École Polytechnique

Thèse présentée pour obtenir le titre de

DOCTEUR DE L'ÉCOLE POLYTECHNIQUE

spécialité : informatique

par

Eric GOUBAULT

Titre : Géométrie du Parallélisme.

soutenue le 13 Novembre 1995 devant le jury composé de :

MM.	Maurice Nivat	Président
	Samson Abramsky	Rapporteur
	Jeremy Gunawardena	Rapporteur
	Vaughan Pratt	Rapporteur
	Gerard Berry	Examinateur
	Patrick Cousot	Directeur de thèse

Acknowledgment

I first became aware of the existence of "Higher-Dimensional Automata" when Samson Abramsky told me about the POPL article by Vaughan Pratt "Modeling Concurrency with Geometry". I was then visiting Imperial College in London, during my DEA, under the supervision of Chris Hankin, and Patrick Cousot (in Paris). When I came back to Paris, Patrick Cousot took me as a Ph.D. student at Ecole Normale Supérieure and gave me both freedom to work on whatever "abstract nonsense" I wanted to, and numerous advices on my research (and teaching). I am very much indebted to the four of them indeed, and to the members of my jury, Maurice Nivat, Jeremy Gunawardena, Vaughan Pratt, Samson Abramsky, Gerard Berry, Patrick Cousot.

* * *

The story is not quite complete yet. To understand why I have been so keen on this geometric and algebraic-topological approach to concurrency theory, I have to give names, and to thank them for this,

- First of all, I have to thank Patrick Cousot for supervising my thesis and teaching me theoretical computer science. Thanks also to Radhia Cousot for supporting me so often (and to Laurent and Thibault).
- Many thanks to Alain Guichardet (Centre de Mathematiques, Ecole Polytechnique) who gave me my first experience in research (Lie groups, representation theory, cohomology of groups), when I was a student at Ecole Polytechnique. I have enjoyed this very much (hence this slight bias towards Lie groups, homology etc.) and this gave me the opportunity to write my first research article [GL92] with my fellow graduate student Claude Lebris.
- IBM La Gaude, for which I have been working four months as a research engineer on digital signal processing. Thanks to all people of the Advanced Technology Systems, in particular to Caroline Fenzy, Cathy Raimondo, Paolo Scotton, Alain Védrenne and Jean Menez.
- The Theory and Formal Methods section, Imperial College, London, in which I have stayed for more than six months. Thanks in particular to

Chris Hankin who has received me so well so many times, Dave Sands, Sebastian Hunt, Lindsay Errington, Dave Clark.

- Marie-Solange Tissier (Corps des Mines) for letting me do research after my graduation at Ecole des Mines.
- CNRS, for having offered me a position at ENS.
- ENS (DMI/LIENS), for backing me when applying for CNRS and more particularly many thanks to Patrick Cousot.

Then, some people had to suffer my extravaganzas,

- Thanks to Régis Cridlig, Bruno Monsuez, Thomas Jensen, Ian Mackie, Maribel Fernandez, Sandra Holzbacher-Jensen, Arnaud Venet, Chris Col–by for listening to my very long and boring seminars. As if it was not enough, Ian, Arnaud, Thomas and Chris have proof-read my dissertation[1]. I am most compassionate to their ordeal. Thanks to Ian for getting me into SemSoc which is going to be a worldwide success.
- Thanks to the Concurrency Group at Stanford (Vaughan Pratt, Rob van Glabbeek, Dusko Pavlovic, Carolyn Brown, Anna Paterson) for such a delightful and inspiring week in June 1995. My apologies to Anna for having disturbed her so many times during her Ph.D.
- Thanks to Jeremy Gunawardena for having discussed with me some important matters about serializability and inviting me so kindly at the Newton Institute.
- Thanks to my parents for thinking I am a good person.
- Special thanks to my brother Jean who has taught me quite a few things. He has also proof-read my dissertation[2].
- Thanks to all my friends. In particular, to Bénédicte Millot (my warmest thanks), Patrick Millot, Christine Mamert, Sophie Berlin, Nathalie Schri–mpf, Charles Grillou, Christian Montès, Pierre-Cyrille and Guiomar (and Antoine!) Hautcoeur, Gilles Joachim, Gilles Royon etc. During the last years I have ruined quite a few parties with my unique ability to talk out loud about mathematics. I apologize for any inconvenience that I may have caused to any of the participants.

Last but not least, thanks a lot to Guinness^{tm} whose faithful support in all countries made my life easier these last three years.

[1] The errors left are mine though.

[2] Same as footnote 1.

La Géométrie du Parallélisme

La théorie des machines séquentielles est bien aboutie. Tous les modèles de calcul connus (et même inconnus, par la thèse de Church) sont équivalents en ce sens qu'ils calculent la même classe de fonctions. Qu'ajoute alors l'étude des machines parallèles à cet état de fait ? En effet un modèle du parallélisme ne pourra pas calculer plus de fonctions qu'un modèle séquentiel. Par contre, il y a un sens en lequel on peut espérer un gain de temps (en général d'un facteur linéaire, parfois superlinéaire) par un calcul parallèle. Sur ce point tous les modèles du parallélisme ne sont pas équivalents. Plus généralement, les modèles existants du parallélisme ne sont pas équivalents quant aux comportements dynamiques dont ils rendent compte.

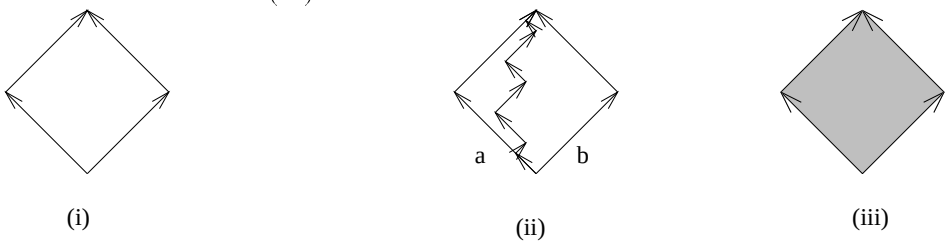
Un des plus anciens modèles aussi bien des machines séquentielles que parallèles, les systèmes de transitions, décrivent bien les états d'une machine, les branchements entre les différents flots d'exécution possibles, mais codent le parallélisme par l'entrelacement d'actions. Cela veut dire en particulier que la confluence forte est indistinguable du parallélisme sur le graphe états transitions.

D'autres modèles ont élaboré sur cette remarque, et ont "décoré" les systèmes de transitions avec des indications sur les comportements permis, à partir de tel ou tel état. Par exemple, les systèmes de transitions asynchrones rajoutent aux systèmes de transitions standards une relation binaire sur les transitions, dite relation d'indépendance. Quand deux actions entrelacées sont indépendantes, elles peuvent être exécutées en parallèle, alors que si elles ne l'étaient pas, cet entrelacement représenterait leur exclusion mutuelle. Les "trace automata", ou les "concurrent automata" sont d'autres formes de cette même idée, ce dernier ayant notamment repris l'idée des "résidus" du λ-calcul pour exprimer l'interférence entre deux actions.

Les systèmes de transitions standards avaient d'attrayant le fait que un certain nombre de comportements dynamiques au cours du calcul (confluence, branchements, états finaux etc.) se lisaient directement sur le graphe états transitions, c'est à dire sur la géométrie du modèle. Ce n'est plus vrai des modèles décorés dans lesquels on ne sait plus représenter la géométrie des exécutions.

Pour remédier à cela, examinons la figure 0.1. Le dessin (i) décrit l'entrelacement de deux actions a et b. On peut imaginer que chacune des deux actions porte un axe, sur lequel on peut lire le temps local d'exécution. Un chemin d'exécution dans lequel un processeur calcule un peu de a tandis qu'un autre calcule un peu de b est figuré en (ii). C'est un chemin croissant en chacune des deux coordonnées partant de l'état initial, arrivant à l'état final, et à l'intérieur

Figure 0.1: Non-déterminisme (i), recouvrement dans le temps (ii) abstrait par une transition de dimension 2 (iii).



du carré délimité par l'entrelacement de a et b. Tous les chemins asynchrones couvrent donc l'intérieur de ce carré. Voulant dans un premier temps n'autoriser que des observations discrètes sur le comportement de systèmes parallèles, on abstrait les états de l'intérieur du carré par l'intérieur du carré lui-même, appelé transition de dimension deux, ou 2-transition (car géométriquement de dimension deux). Alors, la présence de trous contraint l'exécution à rester unidimensionnelle, c'est à dire qu'elle est équivalente à la présence d'exclusions mutuelles. Remplir ces trous revient à autoriser les comportements asynchrones.

Tout ceci se généralise bien évidemment aux dimensions supérieures à deux. Exécuter n actions en parallèle, revient à se trouver sur des chemins à l'intérieur d'un hypercube de dimension n, ou n-cube. Il faut également pouvoir rendre compte des allocations de nouveaux processus sur des processeurs, ainsi que de la composition séquentielle entre processus. Cela nous amène à considérer des formes sur lesquelles se font les exécutions construites en collant des hypercubes de toutes dimensions ensemble, selon leurs bords. Ceci est connu en topologie algébrique combinatoire sous le nom de complexe cellulaire, et plus précisément de complexe cubique. Mais contrairement à la formalisation habituelle, il nous faut également nous souvenir de la direction du temps. On y arrive si on divise les opérateurs bords en deux opérateurs bords, les opérateurs bords début, et les opérateurs bords fin.

Un hypercube de dimension n a $2n$ bords, hypercubes de dimension $n-1$. On a donc $2n$ opérateurs bord agissant sur un hypercube de dimension n, divisés en n opérateurs bord début et en n opérateurs bord fin. Cela se particularise au cas bien connu des automates standards, dans lequel on ne trouve que des transitions de dimension un, et pour lesquels on n'a donc que deux opérateurs bord, un opérateur début, et un opérateur fin.

Une première définition formelle est donc comme suit. On appelle automate semi-régulier M toute suite (M_n) d'ensembles de n-transitions avec leurs opérateurs bords début d_i^0 et fin d_i^1,

$$M_n \overset{d_i^0}{\underset{d_j^1}{\rightrightarrows}} M_{n-1}$$

for all $n \in N$ and $0 \leq i, j \leq n-1$, vérifiant

$$d_i^k \circ d_j^l \quad = \quad d_{j-1}^l \circ d_i^k$$

$(i < j, k, l = 0, 1)$ et $\forall n, m\ n \neq m, \quad M_n \cap M_m = 0$.

Si l'on définit de plus une notion de simulation (et donc de morphisme) entre ces automates, on obtient une catégorie et des constructions algébriques intéressantes sur ces objets. On montre aisément que les automates semi-réguliers forment un topos élémentaire.

Le produit cartésien entre deux automates existe et est par définition le plus grand automate dont chacun des deux automates de départ est une implémentation. On prouve qu'il s'agit du produit synchronisé des automates.

On peut également former l'union (ou coproduit) de deux automates. C'est le plus petit automate implémentant chacun des deux automates de départ.

L'ensemble des simulations d'un automate vers un autre peut également être muni d'une structure naturelle d'automate semi-regulier, c'est à dire en particulier que l'"évaluation" est une simulation. Cela implique que les n-transitions de cet "espace de fonctions" sont essentiellement des évaluations synchrones d'un processus paramétré en dimension n en un argument lui aussi de dimension n (c'est un appel de n procédures classiques, en parallèle).

Il y a aussi un automate "objet de vérité" qui classifie les sous-automates.

On construit également un produit tensoriel représentant l'exécution parallèle sans interférence de deux automates. C'est une opération qui crée du parallélisme et donc qui augmente la dimension des objets en considération.

On a aussi un espace de fonction qui est associé à ce nouveau produit. Une n-transition de cet espace de fonction est maintenant une fonction qui alloue dynamiquement (au moment de l'appel, c'est à dire de l'évaluation) un processus de dimension n (ou n "threads" en parallèle). Ces opérateurs correspondent eux à des fragments de logique linéaire, tandis que les opérateurs "synchrones" correspondaient à une logique intuitioniste.

Dans le deuxième chapitre de la thèse, on définit également des variantes de ce modèle de base.

La première est ce que l'on nomme les automates partiels. L'idée est d'autoriser les fonctions bords à n'être que partiellement définies, et donc géométriquement, à pouvoir considérer des formes non fermées. Cela exprime ainsi la possibilité d'avoir des calculs parallèles en point mort, ou ne bloquant que quelques uns des processeurs disponibles. C'est une généralisation de la méthode classique utilisée en sémantique dénotationnelle pour exprimer les calculs ne terminant pas (symbole $\perp$).

Une deuxième variante consiste à s'autoriser à parler de multi-ensembles d'actions (et donc d'ensembles de chemins) en passant aux sommes formelles engendrées par les ensembles de n-transitions. Pour parler encore plus précisément des chemins, et en particulier pour être à même de décrire la cyclicité ou l'acyclicité d'un automate, on sépare l'indice de dimension n indexant les ensembles M_n en deux indices, un de temps p, un autre q égal à la dimension moins le temps. On obtient ainsi des ensembles $M_{p,q}$ pour définir un automate M, contenant les $p+q$-transitions pouvant être exécutées au temps t. Une première élaboration sur ces idées nous amène à définir les automates réguliers. Une classe plus importante est celle des automates généraux. En utilisant la

structure de module libre engendré par les n-transitions, on peut collecter les bords début en un seul opérateur bord, ∂_0 et tous les bords fin en ∂_1. On obtient ainsi une structure très proche des complexes doubles de modules (à l'exception près de la condition faible sur l'intersection des sous-modules $M_{p,q}$) dont la formalisation suit.

Un automate général est un R-module libre M muni de,

- une décomposition: $M = \Sigma_{p,q\in\mathbb{Z}} M_{p,q}$, telle que $\forall p, q$,

$$M_{p,q} \cap (\Sigma_{r+s\neq p+q} M_{r,s}) = 0$$

- deux différentielles ∂_0 and ∂_1, compatibles avec la décomposition, donnant à M une structure une structure de bicomplexe:

$$\partial_0 : M_{p,q} \longrightarrow M_{p-1,q}$$

$$\partial_1 : M_{p,q} \longrightarrow M_{p,q-1}$$

$$\partial_0 \circ \partial_0 = 0, \quad \partial_1 \circ \partial_1 = 0, \quad \partial_0 \circ \partial_1 + \partial_1 \circ \partial_0 = 0$$

On peut définir de même que pour les automates semi-réguliers une notion de simulation donc de morphisme. Dans le cas des automates généraux, les morphismes sont tout simplement des fonctions linéaires commutant avec ∂_0 et ∂_1, donc sont des morphismes de complexes de modules respectivement pour les complexes (M, ∂_0) et (M, ∂_1). Quand l'automate ne contient pas de cycle, alors M est un vrai complexe double de modules, et les morphismes sont alors des vrais morphismes de bicomplexes.

Remarquons toutefois que toutes ces structures sont non-étiquetées. La fin du chapitre deux y remédie, et définit les étiquetages comme des morphismes à valeur dans l'automate des étiquettes. Cet automate est le plus souvent constitué d'une somme de tores en toutes dimensions.

Rendus à ce point, il est naturel de se poser la question de connaître les rapports entre les modèles anciens, et ceux basés sur les automates de dimension supérieure. C'est là l'objet du troisième chapitre. On y étudie tout d'abord les possibles traductions entre systèmes de transitions ordinaires et les automates semi-réguliers étiquetés, tout du moins d'un type particulier. Si l'on désire conserver l'intuition opérationnelle, c'est à dire interpréter les transitions des automates standards par des 1-transitions de HDA, on a essentiellement à voir si il existe un adjoint à gauche ou à droite au foncteur inclusion ou à des foncteurs qui se réduisent à l'identité sur les systèmes de transitions standards (quand on identifie la catégorie des systèmes de transitions à une certaine sous catégorie Υ^1_{sr} des automates semi-réguliers de dimension un). On prouve qu'en fait, on a les deux.

Le foncteur inclusion $\mathcal{I} : \Upsilon^1_{sr} \to \Upsilon_{sr}$ est adjoint à gauche du foncteur troncation $T_1 : \Upsilon_{sr} \to \Upsilon^1_{sr}$. Ce foncteur "oublie" toutes les transitions de dimension plus élevée que 1, donc creuse des trous dans un automate semi-régulier jusqu'à en faire un automate standard.

Le foncteur troncation (généralisé) $T_n : \Upsilon_{sr} \to \Upsilon^n_{sr}$ est adjoint à gauche du foncteur $\mathcal{G}_n : \Upsilon^n_{sr} \to \Upsilon_{sr}$. Ce dernier, en un sens homologique (et homotopique) très précis, comble tous les trous de dimension supérieure ou égale à n.

En bref, la paire de foncteurs adjoints $(\mathcal{I}, T_n)$ (généralisée de la paire $(\mathcal{I}, T_1)$) correspond à une interprétation des systèmes de transitions dans laquelle tous les niveaux de parallélisme k ($k \geq n$) sont interprétés comme des entrelacements d'exécutions asynchrones de n actions. La paire de foncteurs adjoints $(T_n, \mathcal{G}_n)$ quand à elle correspond à une interprétation dans laquelle toutes les exclusions mutuelles de niveau k (sémaphore initialisé au début à la valeur k, $k \geq n$) sont identifiées au niveau de parallélisme $k+1$.

Ces considérations se transportent au cas des systèmes de transitions asynchrones. On a également une stratégie d'allocation maximale pour laquelle toutes les exclusions mutuelles de niveau k ($k \geq 2$) sont interprétées comme des niveaux de parallélisme $k+1$. La stratégie d'allocation minimale identifie toutes les exclusions mutuelles à des exécutions parallèles. On a les mêmes conclusions avec les traces de Mazurkiewitz dont on se sert pour relier les automates de dimension supérieure aux structures d'événements premières en utilisant les résultats d'adjonctions classiques. Ceux-ci nous permettent également de comparer avec les arbres de synchronisations, les systèmes de transitions déterministes et les langages de Hoare.

Tout cela semble indiquer que le modèle des automates de dimension supérieure semble bien meilleur pour formaliser les propriétés d'allocation du parallélisme que d'autres modèles opérationnels (à l'exception probable des réseaux de Pétri cependant).

Le chapitre quatre ouvre la deuxième partie de la thèse consacrée à l'utilisation des automates pour la définition sémantique de langages.

On commence donc par l'étude générale des propriétés catégoriques des différentes classes de HDA, et plus particulièrement des automates de dimension supérieure généraux. On montre que ces automates forment une catégorie complète et co-complète.

Tout comme dans le cas semi-régulier, le coproduit correspond au choix non-déterministe. Par contre c'est un biproduit, c'est à dire que le produit cartésien est identifié à la somme directe.

Les limites directes permettent de construire des automates infinis par leurs approximations finies (par exemple), et donc permettent de définir des agents récursifs.

On peut également définir un produit tensoriel (le produit parallèle sans interférence) et un foncteur Hom correspondant. Cette fois, la catégorie est *symétrique* monoidale. Si on se restreint aux automates avec un nombre fini d'états et de transitions, on peut même définir un dual, contenant les événements correspondants, de dimension opposée, faisant ainsi de cette sous-catégorie une catégorie $*$-autonome (et même compacte fermée).

Enfin on prouve dans le chapitre trois que les automates semi-réguliers sont une abstraction des automates réguliers qui eux-mêmes sont une abstraction des automates généraux.

On utilise les propriétés catégoriques de ces diverses classes d'automates au chapitre cinq. La catégorie des sous-objets d'un automate D (c'est à dire la sous catégorie de Υ/D composée des monomorphimes, modulo les isomorphismes) est une algèbre de Heyting dans le cas semi-régulier (puisque c'est alors un topos élémentaire) et un treillis complet dans tous les cas. On peut donc s'intéresser à l'utiliser comme treillis de dénotations pour des programmes, avec définitions récursives de domaines (comme D). C'est d'autant plus intéressant que ces dénotations sont naturellement des ensembles de traces avec structure, c'est à dire sont des ensembles dans lesquels on peut lire le temps auquel les branchements, les confluences, les exclusions mutuelles, les allocations de processus etc. sont effectués.

On applique tout cela à un langage similaire à CCS et on donne sa sémantique vraiment parallèle sous deux formes. La première est sous format SOS, par règles d'inférences. La deuxième est une sémantique catégorique.

On peut également généraliser ces domaines sémantiques pour prendre en compte des valeurs de variables. Pour cela, on représente les états par des substitutions de valeurs aux variables (c'est à dire des environnements) et les actions par des homotopies entre la fonction identité sur les environnements et la fonction qu'elle doit calculer sur l'état de la machine. On met cela en pratique sur un simple petit langage impératif parallèle à mémoire partagée. En utilisant une transformation de type "'Continuation Passing Style" on obtient une sémantique pour un langage similaire si ce n'est que le produit parallèle "statique" est remplacé par un opérateur dynamique "fork".

Le chapitre six ouvre la troisième partie dédiée aux propriétés dites géométriques. C'est là la grande originalité du modèle des automates de dimension supérieure, que de voir des propriétés classiques, ou moins classiques à travers une intuition géométrique et d'utiliser ensuite les ressources de la topologie algébrique pour formaliser et résoudre certains problèmes.

On commence dans ce chapitre par définir les propriétés géométriques les plus simples à définir et à calculer. Dans cette catégorie vient tout ce qui peut se définir à partir des groupes d'homologie $H_k(M, \partial_0)$ pour les complexes de modules avec l'opérateur ∂_0 et $H_k(M, \partial_1)$ pour les complexes de modules avec l'opérateur ∂_1. On peut résumer les principaux résultats comme suit.

- l'ensemble des états initiaux engendre $H_0(M, \partial_1)$.
- l'ensemble des états finaux engendre $H_0(M, \partial_0)$, donc un automate diverge si $H_0(M, \partial_0) = 0$.
- l'ensemble des branchements en dimension k union les points morts inverses en dimension k ($k \geq 1$, $k = 1$ constitue les branchements d'automates au sens classique union les transitions n'ayant pas d'état de départ) engendre $H_k(M, \partial_0)$.
- de façon duale, l'ensemble des confluences en dimension k union l'ensemble des points morts de k processeurs engendre $H_k(M, \partial_1)$.

Sachant que l'homologie cellulaire est indépendante de la subdivision choisie, ces propriétés géométriques sont invariantes par raffinement par des processus purement parallèles, sans branchements. De plus on connait beaucoup de moyens de calculs de ces groupes d'homologie.

En particulier, l'homologie du produit tensoriel est donné par la formule de Kunneth, celle de son adjoint à droite, par la formule des coefficients universels, et celle des intersections et unions peut être déterminée en utilisant certaines suites exactes telles la suite exacte de Mayer-Vietoris. On applique tout cela au calcul exact inductif (sur la syntaxe) des branchements et confluences des termes CCS, puisque l'on avait pu écrire sa sémantique avec les opérateurs produit tensoriel, somme etc. dans le chapitre précédent. Les calculs sont un peu longs et techniques mais sont exacts et complets.

On examine ensuite une application directe de ces propriétés, les équivalences sémantiques dites "branching-time" comme la bisimulation, qui caractérisent les temps auquels les choix sont faits dans un automate. On définit tout d'abord des versions de la bisimulation forte tenant compte des stratégies d'allocations d'actions (puisque reliant les n-transitions entre elles) puis on montre qu'elles conservent les branchements modulo l'étiquetage. C'est à dire que modulo des problèmes spécifiques à l'étiquetage des automates, deux HDA bisimulation équivalents ont les même groupes d'homologie pour les complexes en ∂_0 (les branchements). Cela permet de montrer qu'il ne peut y avoir aucun terme CCS representant l'allocation dynamique de trois actions sur deux processeurs.

Le chapitre suivant entreprend l'étude des automates d'un point de vue homotopique. Deux chemins sont homotopes si et seulement si l'un peut se déformer continument en l'autre dans l'automate. Autrement dit, deux chemins sont homotopes si on peut toujours passer localement à travers des 2-transitions, c'est à dire utiliser des relations de commutations entre deux actions indépendantes (puisque sans interférence), pour aller de l'un à l'autre. Donc les classes de chemins modulo homotopie ne sont jamais que les ordonnancements essentiels d'actions (1-transitions) dans un automate, modulo les relations d'indépen–dance.

Remarquons que si la relation d'équivalence "homotopie" est bien la notion classique d'homotopie, les "groupes d'homotopie" qui nous intéressent sont d'une nature quelque peu différente, comme le montre la figure 0.2. Ceci est du à la contrainte de non-inversibilité du temps, ou autrement dit de la croissance des chemins.

En fait, le "groupe d'homotopie orienté" n'est pas naturellement un groupe, c'est plutot un monoide. En effet, on ne peut pas considérer comme dans le cas classique des lacets à partir d'un point base, que l'on peut composer de façon évidente, mais seulement des chemins croissants d'un point à un autre, qui eux ne sont composables que lorsque la fin de l'un est égale au début de l'autre. En fait, on peut plonger ce monoide dans un groupe un peu plus gros, sans toutefois perdre l'information essentielle sur la direction du temps. Ensuite, il faut pouvoir considérer un groupe fondamental orienté pour l'ensemble d'un automate, et pas seulement pour les chemins d'un point fixe à un autre. Cela se fait dans chaque "composante connexe" par amalgamation, en identifiant la

Figure 0.2: Les 4 générateurs du groupe fondamental orienté, les 3 générateurs du groupe fondamental classique

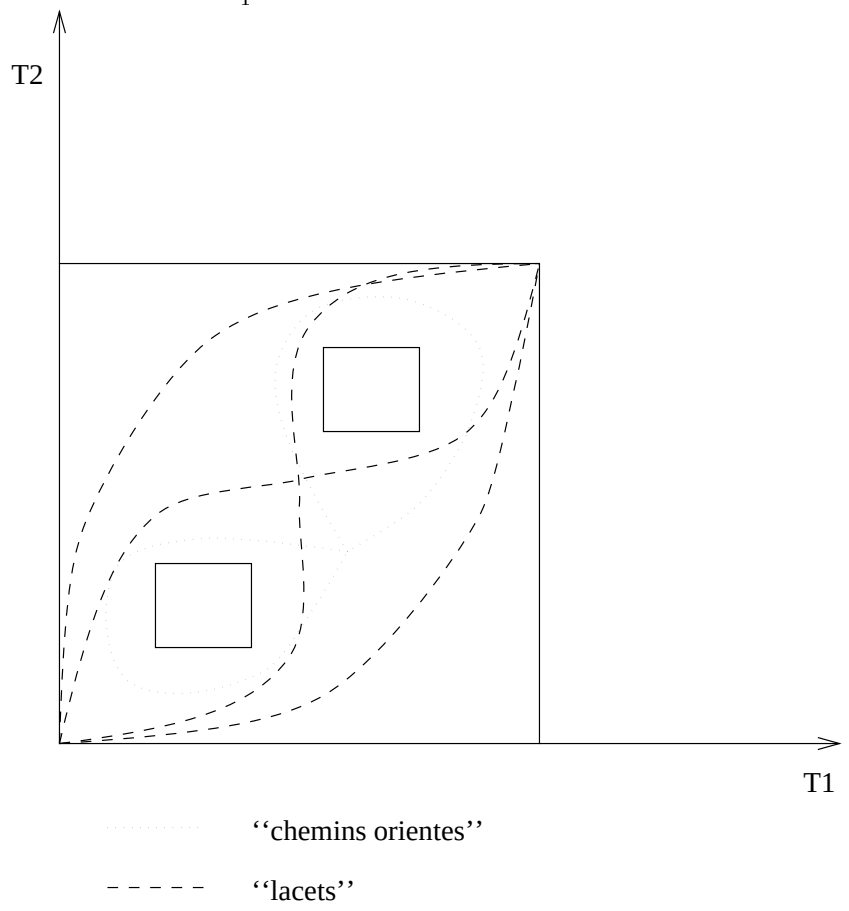
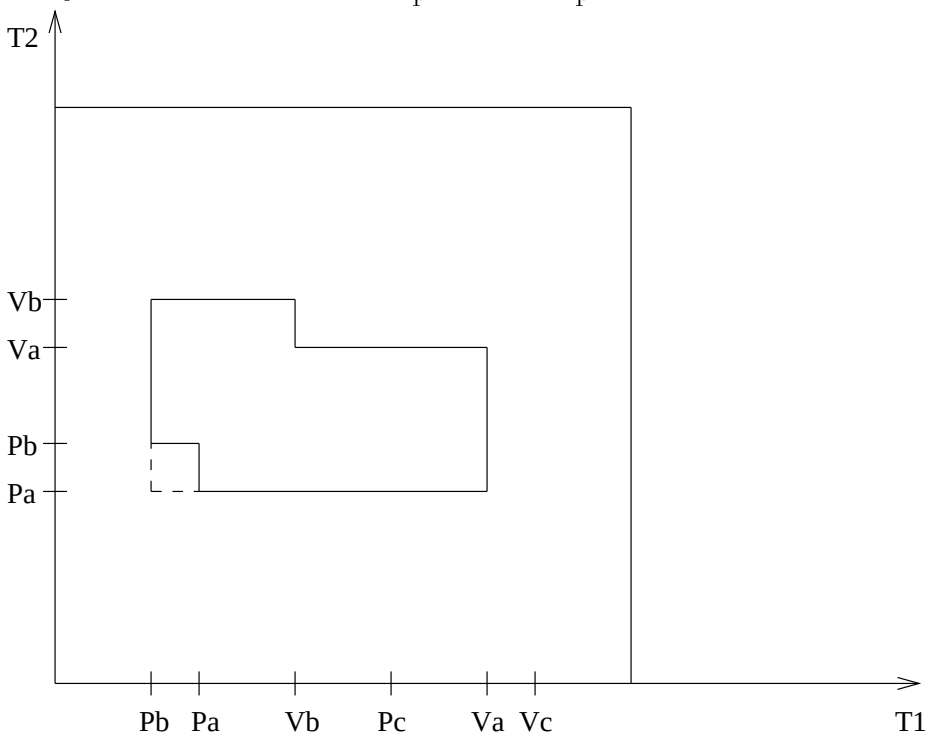


Figure 0.3: Un automate connexe mais pas connexe par chemins croissants



loi de groupe à la concaténation de chemins.

La notion d'automate connexe est elle aussi différente, quand on se restreint à considérer des chemins croissants (voir figure 0.3). Sur l'exemple plus bas, la forme est connexe au sens classique mais ne l'est plus dans le cas orienté.

La contrainte de non-inversibilité pose également des problèmes de définition des groupes d'homotopie de dimension supérieure. Les chemins de dimension deux par exemple sont des surfaces dont les bords sont deux chemins de dimension un d'un même point initial vers un même point final. De façon plus générale, un chemin de dimension n est une collection de n-cubes dont les bords sont deux $(n-1)$-chemins dont les bords sont égaux. La relation d'homotopie sur ces n-chemins est alors la déformation à travers les $(n+1)$-cubes. De même qu'en dimension un, ceci ne forme pas naturellement un groupe, mais plutot un "double monoide", ou même une 2-catégorie. On a en effet deux concaténations possibles. L'une est induite par la concaténation des bords des n-chemins, l'autre est une composition transversale, qui a deux n-chemins de bords p_1, p_2 et p_2, p_3 respectivement associe un n-chemin de bord p_1, p_3. On peut encore plonger cette structure dans un groupe pour lequel ces deux opérations de concaténations sont identifiées à la loi de groupe. De même on peut amalgamer tous ces groupes en un groupe d'homotopie orienté en dimension n.

Ces groupes d'homotopies vérifient des propriétés semblables aux groupes standards. On peut démontrer un analogue des théorèmes d'Hurewicz qui relient les groupes d'homotopies aux groupes d'homologie sous certaines conditions. Ici, les groupes d'homologie correspondant sont les groupes d'homologie avec

les bords totaux ($\partial_0 - \partial_1$) relatifs aux bords des n-chemins considérés.

De même, on a un théorème de Seifert/Van Kampen qui permet de calculer le groupe fondamental orienté d'une union de deux automates en fonction du groupe fondamental de chacun des deux automates et de leur intersection, sous certaines conditions. Une tentative de preuve d'un théorème de Van Kampen pour les groupes d'homotopie de dimension supérieure est faite, à travers l'introduction de groupes d'homotopie définis par suspension. Nous ne savons pas encore si nous sommes en mesure de prouver ce résultat (analogue à ceux de [BH81b, BH81a]).

Le chapitre suivant est consacré aux premières applications de cette théorie homotopique, et en particulier, nous essayons de montrer ses liens avec d'autres problèmes informatiques et mathématiques.

Nous examinons en premier lieu le théorème de Squier qui donne un critère pour savoir si un monoide peut etre présenté par un un système de réecriture canonique fini. C'est en un certain sens le premier résultat de calculabilité que nous considérons. Le résultat est que si un monoide M est présenté par un système de réecriture canonique fini alors ses groupes d'homologie $H_i(M)$ sont tous de type fini. On peut prouver cela en suivant la méthode de Groves et définir une résolution de $\mathbb{Z}$ par des $\mathbb{Z}M$-modules libres, en tant que $\mathbb{Z}M$-module, à partir d'un système de réecriture canonique fini. Rappelons que construire un résolution pour un monoide M est la même chose que construire un espace contractile X sur lequel M agit librement. On peut aisément imaginer que X est la représentation géométrique d'un automate de dimension supérieure, dont le langage est M (ou qui "accepte" ou "reconnait" M). La présence de trous dans cet automate signifirait la non-confluence du système de réecriture. Donc X est naturellement contractile lorsque l'on peut présenter M par un système de réecriture canonique fini.

Une deuxième application de type critère de calculabité peut être trouvé dans le domaine des protocoles de systèmes distribués. Dans le chapitre sur la théorie homotopique, on s'était beaucoup aidé d'un exemple tiré des bases de données parallèles. On dit qu'un système de transactions est séquentialisable si et seulement si pour toute exécution possible, il existe un entrelacement des transactions qui donne le même résultat (c'est à dire que les exécutions sont équivalentes, ou homotopes, aux entrelacements). Dans le cas des monoides présentés par des systèmes de réecritures canoniques finis, on avait une propriété encore plus forte que la séquentialisation. Dans cette partie, on veut savoir si l'on peut calculer certaines fonctions (comme le consensus, le pseudo-consensus etc.) de façon distribuée et robuste. Par robuste, on entend le fait que certains des processus sont autorisés à mourir sans pour autant affecter le déroulement de l'exécution des processus vivants. On impose donc une contrainte de fort découplement (de forte asynchronie) entre les différents processus. Le cas extrême étant le cas "sans attente" où $N-1$ parmi N processus sont autorisés à mourir. Dans ce cas les automates décrivant les exécutions possibles sont N-connexes (au sens de la théorie homotopique orientée). Cela implique que les différentes coupes à temps constant de cet automate sont $(N-1)$-connexes au sens classique du terme. Le consensus, imposant un choix, donc une non-connexité de

la dernière coupe (alors que la coupe au temps initial était connexe) est donc non-calculable sans attente.

On revient également dans ce chapitre sur deux points à peine abordés précedemment. Le premier est l'adjonction entre les systèmes de transition ordinaires et les automates de dimension supérieure, par la stratégie d'allocation maximale. On prouve que l'on a bien n-connexité de l'interprétation qui résulte de l'adjonction, donc que l'interprétation en question correspond à un calcul "sans attente". Le deuxième est dans le même esprit. En utilisant la sémantique de CCS par automate de dimension supérieure donnée précedemment, on prouve que l'on ne peut pas implémenter d'exclusion mutuelle dans un sous-ensemble de CCS purement asynchrone (sans action complémentaire). Cela revient à dire que dans un modèle de machine à mémoire partagée avec accès par lecture/écriture non-atomique, on ne peut implémenter d'exclusion mutuelle (ou de sémaphore).

Jusqu'à présent, on avait essayé d'utiliser des invariants topologiques de manière exacte pour prouver l'impossibilité de calculer certaines fonctions. Dans le chapitre qui suit, on essaie d'approximer le calcul de ces invariants, et en particulier des ordonnanceurs, pour vérifier des protocoles ou des programmes. Cette théorie de l'approximation est basée sur l'interprétation abstraite.

On peut montrer que le calcul des groupes d'homotopie, et donc des ordonnanceurs, est une interprétation abstraite de la sémantique, en utilisant le théorème de Seifert/Van Kampen. Ensuite, en utilisant le théorème d'Hurewicz, on peut même donner un algorithme pour calculer le groupe fondamental (les ordonnanceurs sur un processeur). Ceci permet de verifier des protocoles de systèmes distribués de façon automatique ou de vérifier qu'un programme peut s'implémenter sur une architecture parallèle contrainte. Il suffit de calculer les obstructions si elles existent (qui vivent dans un groupe d'homologie) à la déformation d'une forme (la sémantique du programme) sur une sous-forme (la sémantique du programme sous contraintes).

D'un point de vue dual, si l'on part d'un ordonnanceur séquentiel arbitraire d'un programme, trouver une extension maximale de cette trajectoire dans une variété de trajectoires autorisée est la parallélisation du programme. Nous donnons également un algorithme pour résoudre ce problème (dans le chapitre suivant).

Ces théories de l'ordonnancement peuvent s'inscrire dans un cadre plus général d'approximations de la sémantique des programmes par interprétation abstraite. C'est là l'objet du chapitre 9.

On dit que l'on a un domaine D_a (domaine abstrait) qui est une interprétation abstraite d'un domaine D_c (domaine concret) dès lors que l'on peut exhiber une paire de foncteurs adjoints (α, γ) entre Υ/D_c et Υ/D_a. En fait, en général on se limite à considérer des sous-catégories de Υ/D_c et de Υ/D_a.

En particulier, si l'on se limite à la sous-catégorie des objets de D_c et de D_a respectivement, qui sont des treillis complets, alors on obtient la notion classique d'interprétation abstraite par correspondance de Galois.

C'est le cas pour le repliage sur des états, ou des transitions, où l'on transforme

un HDA en un autre où certains états et transitions sont identifiés. C'est une abstraction utile de la sémantique dans la pratique car elle permet de réduire le nombre d'états et de transitions à considérer pour une analyse de programmes.

C'est également le cas pour la troncation à une dimension maximale de transition donnée (qui permet de se limiter à l'étude du système de transition sur une machine à un nombre borné de processeurs).

Mais la généralisation est utile dans le cas des calculs d'ordonnanceurs car on trouve bien que le calcul du Π_1 sur les élements d'un domaine concret est une interprétation abstraite, mais les ordonnanceurs ne sont en aucun cas les monomorphismes à valeur dans un domaine abstrait mais seulement certains morphismes. L'image du treillis des sous-objets de D_c par cette abstraction n'est qu'un préordre.

Les deux derniers chapitres de cette thèse étudient la possibilité d'étendre le modèle des automates de dimension supérieure de deux façons différentes.

La première est une tentative de définir une algèbre de cubes plus convenable par l'adjonction de fonctions dégénérescences permettant de plonger toute transition de dimension n dans l'ensemble des transitions de dimension $n+1$. Cette construction est utile pour deux raisons. Tout d'abord, elle permettrait (on montre en tout cas que c'est un bon candidat) de définir l'homotopie de façon complètement combinatoire, tout comme cela a été réalisé pour les ensembles simpliciaux. Enfin, cela nous permettrait d'avoir une construction plus classique de la synchronisation (à la Nivat) de deux automates par produit synchronisé, avec un mélange adéquat de synchronisation et de parallélisme (les dégénérescences font en sorte que le produit parallèle est maintenant un produit cartésien). Tout cela n'est encore qu'une tentative, qui devra être comparée aux nombreux travaux sur les algèbres de cubes (en particulier [BH81b]).

La deuxième extension est d'incorporer au modèle une notion de temps. L'idée est maintenant de raisonner dans une géométrie continue. Un HDA avec temps est une sorte de variété topologique, sur laquelle est plaquée une structure observationnelle, un complexe cubique singulier. Localement, cette variété est une variété différentielle. Sur chaque espace tangent, on définit une norme. Cela nous permet de définir la longueur d'un chemin, que l'on prend égal à son temps d'exécution. Le chapitre montre que cela est raisonnable pour donner des sémantiques, à la fois en style SOS et en style catégorique.

Contents

Acknowledgment **3**

La Géométrie du Parallélisme **5**

Index of Notations **31**

Introduction **35**

I Abstract Models for Concurrency **39**

1 Models for concurrency **41**

1.1 A few transition systems 42

1.1.1 Ordinary transition systems 42

1.1.2 Asynchronous transition systems 46

1.1.3 Trace automata 47

1.1.4 Concurrent transition systems 49

1.1.5 Transition systems with independence 50

1.1.6 Petri nets 51

1.2 Behavioural models 53

1.2.1 Event structures 53

1.2.2 Mazurkiewitz traces 57

1.3 A few examples of process algebras and of their semantics 59

1.3.1 CCS 59

1.3.2 Communicating Sequential Processes 63

1.3.3 π-calculus 64

1.4 Real-time systems 65

1.4.1 Models of real-time 66

2 An introduction to Higher Dimensional Automata **71**

2.1 Introduction 71

2.2 Basic definitions 72

2.2.1 Semi-regular HDA 73
2.2.2 (†) Partial HDA 88
2.2.3 Regular automata 91
2.2.4 General HDA 92
2.2.5 Labeled HDA 100

3 Relationship with other models of concurrency 105
3.1 Transition systems and HDA 105
3.2 Asynchronous transition systems and HDA 108
3.3 Mazurkiewitz traces and HDA 112
3.4 Event structures and HDA 115
3.5 Other models 117

II Semantic Definitions 119

4 Categorical properties of HDA 121
4.1 Limits and colimits 121
4.1.1 Zero object 121
4.1.2 Finite limits and colimits 121
4.1.3 (†) Enriched structures 124
4.1.4 Direct and inverse limits 125
4.2 Tensor and Hom 127
4.2.1 Autonomous structure 131
4.2.2 $*$-autonomous structures 133
4.3 A formal comparison of the HDA-based models 137
4.3.1 Semi-regular and general HDA 137
4.3.2 Regular and general HDA 138
4.3.3 Semi-regular and regular HDA 139
4.3.4 The lattice of HDA 139

5 Introduction to semantic domains of HDA 141
5.1 Basic principles 141
5.2 Domains of HDA in order-theoretic form 142
5.3 Recursive domain equations 143
5.4 Example: A CCS-like Language 145
5.4.1 Semantics using semi-regular HDA 145
5.4.2 Semantics using general HDA 149
5.5 Semantics of concurrent languages 151
5.5.1 Introduction 151
5.5.2 An imperative language with shared memory 153
5.5.3 A variant with a Fork operator 158

III Geometric Properties **161**

6 Basic geometric properties **163**

6.1 Homology . . . 163
6.1.1 Initial and final states . . . 164
6.1.2 Deadlocks and initial deadlocks . . . 166
6.1.3 Divergence . . . 167
6.1.4 Branchings and mergings . . . 167
6.2 Refinement of actions . . . 169
6.3 Homology functors . . . 170
6.3.1 Homology of limits and colimits . . . 170
6.3.2 Homology of tensor products and Hom . . . 170
6.3.3 Exact sequences . . . 172
6.4 Branchings and mergings of CCS . . . 173
6.5 Application: Semantic Equivalences . . . 180
6.5.1 Linear-time semantic equivalences . . . 180
6.5.2 Branching-time equivalences . . . 181

7 Serialization and schedulers **191**

7.1 Introduction and motivation . . . 191
7.1.1 Scheduling problems in computer science . . . 191
7.1.2 A geometric approach . . . 193
7.2 The group of connected components . . . 201
7.3 Towards formal definitions . . . 202
7.4 The fundamental group . . . 204
7.4.1 Functors $\Pi_1^{p,q}$, Π_1^{∞} and Π_1 . . . 208
7.5 (†) Homotopy of maps . . . 210
7.6 Higher-order homotopy groups . . . 212
7.6.1 First definition . . . 212
7.6.2 (†) Second definition . . . 215
7.7 Some properties of the homotopy modules . . . 219
7.7.1 Combinatorics of Π_1 . . . 219
7.7.2 Seifert/Van Kampen theorem . . . 221
7.8 Some applications . . . 223
7.8.1 Schedulers . . . 223
7.8.2 Mutual exclusion . . . 225
7.9 Approximation of schedulers, branchings and mergings . . . 225

8 Applications of scheduling properties **229**
8.1 Word problems in monoids . . . 229
8.1.1 Presentation of monoids . . . 229
8.1.2 Homology of monoids . . . 229
8.1.3 An introduction: Squier's method . . . 230
8.1.4 Groves' construction . . . 233
8.2 Results in protocols for distributed systems . . . 240
8.2.1 A quick survey . . . 240
8.2.2 Herlihy's framework . . . 241
8.2.3 Wait-free protocols . . . 243
8.3 Application: no algorithm for mutual exclusion . . . 246
8.4 Some properties of the interpretations of transition systems . . . 248

IV Analysis of Programs **253**

9 Analysis of programs **255**
9.1 Abstract interpretation . . . 255
9.1.1 Introduction . . . 255
9.1.2 Definitions . . . 255
9.1.3 An abstraction: denotational semantics . . . 257
9.1.4 A technical abstraction: the image functor . . . 260
9.1.5 An abstraction: truncation . . . 261
9.1.6 An abstraction: folding . . . 262
9.1.7 Schedulers as an abstract interpretation . . . 264
9.1.8 Dependence orderings and abstract interpretation . . . 265
9.1.9 Verification of protocols . . . 267
9.1.10 Inference of a best parallelization . . . 267
9.2 Algorithmic details . . . 268
9.2.1 Representation of HDA . . . 268
9.2.2 Representation of program semantics and constraints . . . 269
9.2.3 Verification . . . 269
9.2.4 Inference . . . 276
9.2.5 Optimizations . . . 278

V Extensions **281**

10 Combinatorial HDA **283**
10.1 Combinatorial HDA . . . 283

10.2 Homotopy theory 289
10.3 Homotopy theory of general HDA 292
10.4 Relationship with semi-regular HDA homotopy 293
10.5 Construction of wait-free protocols 294

11 Timed Higher-Dimensional Automata 299
11.1 Introduction 299
11.1.1 From the untimed to the timed world 300
11.1.2 Timing a semi-regular HDA 301
11.2 Basic definitions 302
11.2.1 Fairness 307
11.2.2 Correctness Timed/Untimed 308
11.2.3 Zeno behaviours 309
11.2.4 Complexity for constrained parallel machines 310
11.3 THDA as denotational and operational models 310
11.3.1 Limits 311
11.3.2 Colimits 314
11.3.3 Function space 315
11.3.4 Tensor product 316
11.3.5 Labeled timed HDA and THTS 318
11.4 Examples 319
11.4.1 Semi-regular HDA as THDA 319
11.4.2 Semantics of a toy language 320
11.5 Homology and Homotopy 323
11.5.1 Cubical versus simplicial homology 324
11.5.2 Homotopy of oriented paths 324

Future Work 329

A Mathematical background - Rings, modules and complexes 331

B Mathematical Background - Some basic properties of simplicial complexes 333

C Mathematical Background - Some basic concepts of differential geometry 335

List of Figures

0.1 Non-déterminisme (i), recouvrement dans le temps (ii) abstrait par une transition de dimension 2 (iii). 6

0.2 Les 4 générateurs du groupe fondamental orienté, les 3 générateurs du groupe fondamental classique 12

0.3 Un automate connexe mais pas connexe par chemins croissants . 13

1.1 A transition system 43

1.2 A partial morphism of transition systems and its corresponding total morphism 45

1.3 Transition system obtained by interleaving two actions 45

1.4 Condition (3) for asynchronous transition systems 47

1.5 Condition (4) for asynchronous transition systems 47

1.6 Permutation equivalence and residuals 48

1.7 Cube axiom 50

1.8 A Petri net graphical representation: mutual exclusion between a and b 52

1.9 A Petri net graphical representation: truly concurrent execution of a and b 53

1.10 A parallel switch 56

1.11 Configurations of events for the parallel switch example 56

1.12 Coproduct of two Petri nets 61

1.13 Allowed transitions of Figure 1.12 62

1.14 Product of two Petri nets 62

1.15 A timed Büchi automaton 67

2.1 Non-determinism (i) versus overlap in time (ii) abstracted by a transition of dimension 2 (iii). 74

2.2 Glueing of elementary shapes to get a semi-regular HDA. 76

2.3 A path and its inclusion morphism in a semi-regular HDA. . . . 77

2.4 Coproduct of two automata (left) and amalgamated sum of the same automata (right). 82

2.5 A cartesian product (left) and a fibered product (right). 83

2.6 A synchronous function space of HDA. 83
2.7 A subset with two one-transitions of T_2 of the labelling automaton L (torus shaped – dashed lines materialize a 2-transition -). . 102

3.1 Example of minimal allocation and then maximal allocation. . . 108
3.2 Example of co-retract $\mathcal{F} \circ \mathcal{E}$ 110
3.3 Co-retract $\mathcal{H} \circ \mathcal{G}$ 112

7.1 MIPS R4000 floating point unit. 192
7.2 A process graph for two transactions accessing the same shared item. 194
7.3 An example of process graph. 195
7.4 Left and right paths in a mutual exclusion. 196
7.5 A process graph with two mutual exclusions. 197
7.6 A HDA (i) and its set of paths (ii). 200
7.7 Another HDA (i) and its set of paths (ii) 200
7.8 A "non-connected" automaton in the oriented theory 203
7.9 Step by step deformation (curved arrows) of one path onto an other 204
7.10 Example of a fundamental group of oriented paths. 207
7.11 Composition of equivalence classes of paths modulo homotopy. . 209
7.12 A path X of dimension 2 between two paths p_1, p_2 of dimension 1.212
7.13 Two homotopic 2-paths. 213
7.14 $*$ and $\cdot$ operations on n-paths 214
7.15 A picture of P_1^k and $P_1^k \otimes P_1^k = P_2^k$. 215
7.16 2-fpaths and 2-paths compared. 217
7.17 A knot M (i) an "upper approximation" of the paths modulo homotopy and the corresponding M' (ii) that cannot be a retract of M. 220
7.18 Two different configurations of holes: left is "dependent" holes, right is "independent" ones. 220
7.19 Conflict in a shared memory parallel machine/concurrent database.224

8.1 A resolution up to dimension one 232
8.2 Product of two 1-cubes 236
8.3 A parallel reduction machine based on a finite canonical TRS . . 240
8.4 Input complex for the binary consensus. 242
8.5 Output complex for the binary consensus. 242
8.6 The map Δ for the binary consensus. 243
8.7 Input and output complex for some domain of HDA D (a 3-cube C), drawn in Herlihy's way at the right hand side. 245

8.8 The effect of a failure of one process in (i)-a mutual exclusion, (ii)-a truly concurrent execution. 245
8.9 Binary pseudo-consensus input, output complexes and decision map. 246
8.10 Binary pseudo-consensus protocol complex, its deformation onto the output complex and its implementation. 247

9.1 A domain of automata D, a subposet SD of Sub and its abstraction to Sc 266
9.2 Sub (simplified, called SD), the denotation p of the program, its subposet of retracts R and the constraint C. 267
9.3 The triangular algorithm 270
9.4 The algorithm $n - scheduler$ 271
9.5 A process graph discretized as an HDA (8 2-transitions, 24 1-transitions and 16 states). 273
9.6 The 1-step inference algorithm Add_n 277
9.7 A parallelization example. 277

10.1 Action of generating morphisms δ^1, δ^0, σ^1 and σ^0. 285
10.2 Homotopy in a combinatorial HDA (the curved arrows indicate the elementary deformations – left is the HDA, right is the corresponding simplicial complexes) 290
10.3 A 1-path in a combinatorial HDA. 291
10.4 A decision map read from a combinatorial HDA 295
10.5 Wait-free protocols corresponding to a subdivision of a combinatorial HDA 296

11.1 Some paths in a Timed HDA 302
11.2 Deformed cubes 303
11.3 Delay transitions (left) and timeout HDA (right). 304
11.4 A timed HDA representing a billiard ball trajectory (i), and a refined version (ii). 305
11.5 A weakly fair path whose untimed version (right) is not fair . . . 307
11.6 A strongly fair path and its untimed fair version (right) 308
11.7 Typical Zeno behaviour and a hybrid system implementing it. . . 309
11.8 Synchronized product (middle) and coproduct (right) of two transitions (left) 313
11.9 Parallel composition (middle) of two transitions (left) and linear function space (right). 317
11.10A labeled timed HDA 319

B.1 A filled-in triangle seen as a simplicial complex 333

C.1 A manifold and its atlas 336

Index

ACP_ρ, 33
ACP_ρ
  absolute stamp, 33
  integral operator, 33
  relative stamp, 33
  time-lock, 33
ATP, 33
  start-delay operator, 33
  unbounded idling, 33
$TCSP$, 33
  delay operator, 33
  weak timeout operator, 33
TPL, 33
  delay operator, 33
$TeCCS$, 33
  delay operator, 33
  time-lock, 33
  unbounded idling, 33
$TiCCS$, 33
  prefixing operator, 33
$U-LOTOS$, 33
  asap, 33
  delay operator, 33
i-boundary, 125
i-cycle, 125
k-set agreement task, 158
n-scheduler, 218

abstract interpretation, 209
abstract operator, 217
  safe approximation of, 217
action urgency, 33
asynchronous transition system, 11
  morphism of, 12
automaton, 57

biproducts, 86
bisimulation, 144
branching, 125

cartesian closed, 84
co-divergence, 129
cokernels, 85
compact-closed, 96
complementary action, 24
complex
  total, 59
complexity, 261
concurrent transition system, 14
  morphism of, 15
consensus task, 158
  binary, 197
coproduct, 85, 265
cubicalation, 50
  subdivision of, 131

deadlock, 128
  initial, 129
deadlock-free, 33
decision tasks, 158
delay, 256
dependence ordering, 219
differential structure, 254
direct limit, 87
direct system, 87
  map of, 87
distance
  inf, 258
  sup, 258
divergence, 129
domain, 272
dual, 95

environments, 115
ESTEREL, 30
event structure, 20
  configuration of, 20
  morphism of, 19
  prime, 19

expansion law, 218

fairness
  strong, 259
  weak, 259
finite-variability, 34
folding, 216

Galois connection, 210
geodesics, 270

HDA
  bounded above, 60
  bounded below, 60
  definition, 56
  end boundary, 96
  finite state, 56
  morphism of, 62
  source boundary, 56
  start boundary, 96
  sub-, 62
  target boundary, 56
Hom, 91
homology, 125
homotopy, 168
  bicomplex, 168
  chain, 168
  classification theorem, 133
  topological, 168

idle transition, 10
image functor, 214
information, 96
invisible action, 24
isometries, 257
items, 153

Kelley space, 254
kelleyfication, 263
kernels, 83
Künneth, 132

left-module, 281
local dimension, 145
local skeleton, 145, 147
LUSTRE, 30

manifold, 285
  atlas, 285
  chart, 285
  local coordinate, 285
Mazurkiewitz trace language, 23
  generalized, 22
  morphism of, 22
Mazurkiewitz trace theory, 219
merging, 125
module, 281
  complex of, 282
  cyclic, 282
  free, 281
  graded differential, 59
  injective, 282
  projective, 282
  torsion sub-, 282

n-events, 60
  elementary, 60
n-transitions, 60
  elementary, 60
non-deterministic choice, *see* coproduct
norm, 253, 255

object
  initial, 83
  terminal, 83
  zero, 83
operational, 103

parallel composition, *see* tensor product
path, 41, 60
  partial, 41
  semi-partial, 41
  total, 41
persistency, 34
Petri net, 16
  Condition/Event, 17
  morphism of, 18
  Place/Transition, 17
product
  cartesian, 83
  tensor, 89
protocol, 158, 197
  complex, 198
  decision map, 198
  input complex, 197

output complex, 197
wait-free, 198

quotients, 84

renaming task, 158
RTCCS, 271

semantics
denotational, 103, 211, 272
operational, 273
semi-regular
HDA, 39
morphism, 40
sequence
exact, 134
Mayer-Vietoris, 134
short exact, 133
shift operator, 149
SOS, 43, 118, 262
spectral sequence, 182
StateCharts, 30
states, 60
elementary, 60
final, 126
initial, 127
sum
amalgamated, 86
direct, 85
synchronized product, 264

tangent space, 254
tensor product, 267
time, 96
timed automata, 31
accepting criterion, 32
timed transition table, 31
timed HDA, 255
labeled, 258, 269
morphism of, 256
non-expansive morphism of, 257
timed transition system, 31, 258, 270
timed execution sequence, 31
timeout, 256
TLA, 32
topos, 41
trace automaton, 13
permutation equivalence, 13
permutation preorder, 13
transactions, 153
transition system, 8
morphism of, 9
partial morphism of, 9
transition system with independence, 16
morphism of, 16
truncation, 215
two-phase protocol, 219

Van Kampen's theorem, 218
vector space, 281
virtual path, 257

Zeno, 260

Index of Notations

Notation	Name	Section
$(S, i, L, Tran)$	transition system	1.1.1
TS	category of transition systems	1.1.1
TS_A	category of transition systems on alphabet A	1.1.1, 3.1
$(S, i, E, I, Tran)$	asynchronous transition system	1.1.2
ATS	category of asynchronous transition systems	1.1.2
ATS_E	category of asynchronous transition systems on E	1.1.2, 3.2
(E, Q, T)	trace automaton	1.1.3
$(G, \uparrow)$	concurrent transition system	1.1.4
CTS	category of concurrent transition systems	1.1.4
$(S, s^I, L, Tran, I)$	transition system with independence	1.1.5
TSI	category of transition systems with independence	1.1.5
$(P, T, pre, post)$	Petri net	1.1.6
PN	category of Petri nets	1.1.6
$(E, \leq, \#)$	prime event structure	1.2.1
$(E, \leq, \#, l, L)$	labeled event structure	1.2.1
LES	category of labeled event structures	1.2.1
$\mathcal{D}(E, \leq \#)$	configurations of a prime event structure	1.2.1
$\mathcal{D}^0(E, \leq \#)$	finite configurations of a prime event structure	1.2.1
$(E, Con, \models)$	event structure	1.2.1
(M, I, L)	Mazurkiewitz trace	1.2.2
GTL	category of generalized Mazurkiewitz traces	1.2.2
GTL_L	category of generalized Mazurkiewitz traces on L	1.2.2, 3.3

Υ_{sr}	category of semi-regular HDA	2.2.1
$\mathcal{T}_n$	truncation functor at dimension n	2.2.1
Υ^n_{sr}	category of semi-regular HDA of dimension up to n	2.2.1
$\square$	category of n-cubes with no degeneracies	2.2.1
$\square$**Set**	category of functors from $\square$ to **Set**	2.2.1
σ_x	singular cube	2.2.1
$D_{[n]}$	representable functor in $\square$**Set**	2.2.1
$\mid M \mid$	geometric realization of M	2.2.1
Top	category of topological spaces with continuous maps	2.2.1
$\mathcal{S}(X)$	singular cube functor applied to X	2.2.1
$P\Upsilon$	category of partial HDA	2.2.2
$CP\Upsilon$	category of closed partial HDA	2.2.2
Υ_r	category of regular HDA	2.2.3
$\underline{M}$ or $\mathcal{A}(M)$	free general HDA generated by M	2.2.4
$R-Mod$	free R-module functor	2.2.4, A
$Tot(M)$	total complex of M	2.2.4
Υ	category of general HDA	2.2.4
Υ_a	category of acyclic HDA	2.2.4
Υ_F	category of free HDA	2.2.4
Υ_f	category of finite free HDA	2.2.4, 4.2.2
L_A	labelling HDA on alphabet A	2.2.5
$\mathcal{C}$	category of well labeled HDA	3.1
$\mathcal{C}p$	category of well labeled pointed HDA of dimension 1	3.1
$\mathcal{C}p'$	category of well labeled pointed HDA	3.1
$\mathcal{U}$	representation functor from TS_A to $\mathcal{C}p$	3.1
$\mathcal{V}$	representation functor from $\mathcal{C}p$ to TS_A, inverse of $\mathcal{U}$	3.1
$\mathcal{I}$	inclusion functor	3.1
$\mathcal{G}_n$	"parallelizing" functor right-adjoint to $\mathcal{T}_n$	3.1
$\mathcal{U}_{min}$	minimal allocation functor for transition systems	3.1
$\mathcal{U}_{max}$	maximal allocation functor for transition systems	3.1
$\mathcal{V}_{min}$	right-adjoint to $\mathcal{U}_{min}$	3.1
$\mathcal{V}_{max}$	right-adjoint to $\mathcal{U}_{max}$	3.1
$d\mathcal{C}p'$	category of deterministic well labeled pointed HDA	3.2

$\mathcal{F}$	maximal allocation functor for asynchronous transition systems	3.2
$\mathcal{E}$	left-adjoint to $\mathcal{F}$	3.2
$\mathcal{G}$	minimal allocation functor for asynchronous transition systems	3.2
$\mathcal{H}$	right-adjoint to $\mathcal{G}$	3.2
$\mathcal{C}p_2$	category of well labeled pointed HDA of dimension at most 2	3.3
$\mathcal{V}$	representation functor from GTL_L to $\mathcal{C}p_2$	3.3
$\mathcal{W}$	right-adjoint of $\mathcal{V}$	3.3
DES	category of deterministic labeled event structures	3.4
ST	category of synchronisation trees	3.5
HL	category of Hoare languages	3.5
dTS	category of deterministic transition systems	3.5
$Ker\ f$	kernel of f	4.1.2
$Im\ f$	image of f	4.1.2
$\varinjlim$	direct limit (filtered colimit) functor	4.1.4
DG	category of graded differential modules	4.1.4
M^*	dual of M	4.2.2
Υ^2	category of bi-indexed semi-regular HDA	4.3.1
$Forget$	right-adjoint to the $R-Mod$ functor	4.3.3
$\mathcal{L}$	a programming language	5.5.2
Sub	category of subobjects of some domain D	5.2
$[f,g]$	$(\partial_1-\partial_0)$-homotopy between f and g	5.5.1
$\check{f}$	name of f	5.5.1
$\hat{f}$	sequentialization of f	5.5.1
$H_i(Q,\partial)$	ith homology group of Q with respect to ∂	5
G	subdivision operator	6.2
Re	one-step refinement operator	6.2
$\mathcal{P}(M)$	set of semi-partial paths of M	6.5.1
$V_n(x)$	local skeleton of dimension n at x	6.5.2
$P_n(M)$	R-module of n-paths of M	6.5.2
Pa	lock a	7.1.2
Va	release a	7.1.2
$\tilde{\Pi}_0(M)$	reduced group of connected components of M	7.2
$P_1^k(M)$	1-paths of length k in M	7.4
$\Pi_1^k(M)$	fundamental group of M for paths of length k	7.4
$\Pi_1^{\alpha,\beta}(M)$	fundamental group of M from α to β	7.4

$\Pi_1^\infty(M)$	fundamental group of paths of infinite length of M	7.4.1
$\Pi_1(M)$	the full fundamental group of M	7.4.1
$\tilde{\Pi}(M)$	reduced fundamental group of M	7.4.1
$P_n^{p_1,p_2}(M)$	R-module of n-paths of M between p_1 and p_2	7.6.1
$Q_n^{(p_1,p_2;\alpha,\beta)}(M)$	set of normalized n-paths from p_1 to p_2 in M	7.6.1
$\Pi_n^{p_1,p_2}(M)$	nth homotopy group of M between p_1 and p_2	7.6.1
$\tilde{\Pi}_n(M)$	nth reduced homotopy group of M between p_1 and p_2	7.6.1
$\Pi_n(M)$	nth homotopy group of M	7.6.1
$\Pi_n'^k(X)$	nth homotopy group of length k of M	7.6.2
$S(X)$	suspension of X	7.6.2
$\mathbb{Z}M$	ring of M	8.1.2
I	input complex	8.2.2
O	output complex	8.2.2
$\square$	category of cubes with degeneracies	10.1
$\pi_n(M)$	nth homotopy group of the combinatorial HDA M	10.2
$\square\mathcal{C}$	category of cubical objects in $\mathcal{C}$	10.1
D_c	concrete domain	9.1.2
D_a	abstract domain	9.1.2
(α, γ)	abstract interpretation pair	9.1.2
Ω	denotational semantics functor	9.1.3
$\mathcal{I}m$	image functor	9.1.4
$\mathcal{J}$	right-adjoint of the image functor	9.1.4
$\mathcal{M}_p$	folding functor by p	9.1.6
$\mathcal{N}_p$	right-adjoint to $\mathcal{N}_p$	9.1.6
α_n	functor n-scheduler	9.1.7
γ_n	"parallelizing" functor right-adjoint to α_n	9.1.7
$(S, i, E, Tran, l, u)$	timed transition system	1.4.1
$df(u).\dot{u}$	differential of f at u applied to $\dot{u}$	11.2, C
$\|.\|$	norm	11.2
$\mathcal{V}i(u, v)$	set of virtual paths from u to v	11.2
d_i^X	distance inf in X	11.2
d_s^X	distance sup in X	11.2
$T\Upsilon$	category of timed HDA	11.2
$T\Upsilon_=$	category of timed HDA and isometries	11.2
$T\Upsilon_\leq$	category of timed HDA and non-expansive maps	11.2
$\mathcal{F}t$	time forgetful functor	11.2.2

Introduction

Sequential machines can be studied by examining their operational behaviours – that is by looking at their state transition graphs. One of the fundamental properties that we might want to study is confluence of the performed computation. This is obviously a property of a highly geometric nature: we must be able to complete all non-deterministic applications of conflicting reductions by some other reductions that all converge to the same result; i.e. we must have diamond shapes in the state transition graphs describing the sequences of operations of our sequential machines.

For concurrent machines, the geometric properties of computation are more intricate. Purely (interference free) asynchronous executions of two processes are confluent and therefore recognizable geometrically as diamonds (or squares).

There could also be cubical shapes in the transition graph. These could arise in different ways. If there are three processes, then their asynchronous execution will generate cubes just like the asynchronous execution of two processes generates squares. Another way that cubes could arise is if there are fewer than three processes, but the actions of some process may be chosen non-deterministically, yielding squares or cubes; this is similar to the situation that arises when studying the confluence of sequential machines. The intuitive difference between the former way in which cubical shapes can arise and the latter is that in the former situation we are allowed to consider that we have spent any arbitrary amount of time in the respective sequential processes, and so in some sense all paths in the interior of the cubical shapes might be part of valid executions. In contrast, in the latter situation two or more dimensions of the cubic shapes arise by non-deterministic choice, which remains sequential and does not allow us to go in the interior of the cubes (and, to generalize, hypercubes). In order to distinguish these two behaviors, we abstract interiors of squares, cubes, etc. as 2-dimensional transitions, 3-dimensional transitions, and so forth. This leads to a generalization of ordinary automata to what we call Higher-Dimensional Automata (HDA in short), as first proposed in [Pra91b] and [vG91].

The main contribution of this thesis is to develop a few of the possible theories of such automata and to study interesting "geometric properties" of concurrent executions using this model. This is still conceived as an introduction to using geometry for solving problems in concurrency theory. Some sections are attempts – and not "definitive" solutions – to define the necessary concepts.

First of all, we remind the reader of a few models for concurrency (Chapter 1), in particular the operational ones which distinguish the non-deterministic choices from "true concurrency". This chapter will enable us to have the basic concepts of concurrency theory at hand, and we will recognize most of them in the HDA models.

The geometry of the operational models can be made clear by using techniques from combinatorial algebraic topology, (the semi-regular HDA introduced in the first sections of Chapter 2) and from homological algebra (the general HDA introduced at the end of Chapter 2). Semi-regular HDA are collections of cubes of all dimensions glued together: they are cubical complexes, a particular case of combinatorial cell complexes, a standard notion of algebraic topology. These in turn generate a weak form of bicomplexes of modules, the general HDA. Even if the techniques are new in computer-science, we can define notions from standard transition systems' theory: SOS rules, denotational semantics using the categorical structure of the models (see also Chapter 4). Some other HDA-based models are defined which refine the basic semi-regular one: partial HDA which can also express deadlocking behaviours and labeled HDA which add a notion of observation.

The relationships with other models are extensively studied in Chapter 3. It is shown in particular that some formal adjunctions between some transition-system-based models and HDA correspond to the different possible allocations on a certain number of processors. For instance, for asynchronous transition systems, the independence relation I between actions a, b is interpreted as a 2-transition "aIb", but aIb and bIc and cIa can be interpreted either as a 3-transition (maximal allocation strategy) or just as the six 2-transition boundaries of that 3-transition.

Then, we study the categorical properties of general HDA. The categorical constructions bear computer-scientific meanings. As customary since [Win88], they are very much like operators used in process algebra. These constructions are also similar to the standard ones for complexes of modules. The situation is slightly complicated by the fact that the complexes which formalize HDA are not quite bicomplexes. These technical difficulties are addressed in Chapter 4. It is proven that general HDA and labeled HDA form autonomous categories. Interesting subcategories are shown to be $*$-autonomous. We introduce in Chapter 5 the use of some of the categorical properties we have studied in order to give semantics to simple languages, like a CCS-like process algebra.

The field of algebraic topology offers several techniques for giving an algebraic description of topological properties of geometric objects. For instance, we can develop a theory of *homology* of HDA in the standard way (see for instance [CE56] or [ML63]). To each HDA we associate a sequence of modules that characterizes the essential branchings and mergings in the HDA. This is the aim of Chapter 6. These homology modules seem to be more amenable to automated computation than the fundamental groups associated with homotopy theory. The computation of the homology of programs in the language of Chapter 5 (i.e. its branchings and mergings) is done using general techniques from homological algebra. But homotopy is also interesting for many reasons, in

particular for scheduling properties (as shown for the two-phase protocol for concurrent databases in [Gun94]) and so it must be studied in its own right. This is started in Chapter 7. Schedulers are actually derivable from the HDA semantics by abstract interpretation. This gives algorithms and approximation techniques for verifying protocols and even for parallelizing programs, as shown in Chapter 9, where a general theory is sketched.

Various applications come in Chapter 8. First of all we make a link (both formal and intuitive) between serialization issues and word problems in monoids. Monoids whose word problem can be solved by a finite canonical systems are constrained in such a way that their homology groups are of finite dimension. This is a geometric property of the execution in the canonical rewriting systems, very similar to serializability, which fits well into the HDA framework (even if first exposed in other terms, in particular in [Gro91]). Then we show that some of the homotopical properties of concurrent executions enable us to solve some problems about protocols for distributed systems. The idea is again that some geometric properties must be preserved during the execution of purely asynchronous processes and therefore some decision tasks cannot be solved by asynchronous machines. We follow the presentation of M. Herlihy et al., but put it in a semantic perspective. This will be refined in Chapter 10 where a "non-degenerate" homotopy theory (i.e., a homotopy theory which is not a homology theory) will be developed by adapting the homotopy theory of simplicial complexes on cubical complexes. This will extend it in particular to all general HDA, whereas the one discussed in Chapter 7 was only applicable on free general HDA generated by acyclic semi-regular HDA. As an application, we will give a semantic view of the construction of the wait-free protocols of [HS94] in more general terms.

Finally, we give hints in Chapter 11 about a possible extension of the theory of HDA in order to deal with real-time systems.

Remark Some chapters come from articles written by me and co-authors. In particular, the notion of general HDA was developed in [GJ92] where the point of view of homology theory was described and an application to CCS and bisimulation was given. A category of HDA was defined in [Gou93] together with an application to the semantics of a toy imperative language. This was continued in [CG93] with application to Linda-based languages and their abstract interpretation. Finally, a sketch of the theory of schedulers as an abstract interpretation of an HDA semantics was given in [Gou95].

Notations In the following, we use two special symbols at the beginning of sections, definitions or theorems,

- (†): can be saved for second reading
- (∗): indicates that the following is of classical inspiration. If not well-known, it uses at least very classical arguments.

Part I

Abstract Models for Concurrency

Chapter 1

Models for concurrency

Sequential computation can be modeled through many different formalisms. One of the oldest, the Turing machine, gives a very mechanical view on computation [Dav58]. Lambda calculus, and all combinatory logics (see [Bar84] or [HLS72] for a survey) put more the emphasis on passing arguments to functions (β-reduction) and using elementary computations (like δ-rules). Post systems, Random Access Machines, Markov systems are other examples among many more models for sequential computations.

All of them are actually equivalent and Church's thesis asserts that all models for sequential computation we may think of will be equivalent to these as well. The equivalence considered here is rather basic and takes the form of a simple input/output relation: all the models cited above compute the same class of functions on the integers, the class of "computable functions".

This may seem to be the final answer to all questions in computer science but this is not quite the case. Even for sequential machines for which the dynamic properties of the execution do not appear as essential, we may sometimes be interested in measuring the complexity of algorithms as a number of steps of elementary computations in some model. Here the relationship is less obvious [vEB90] and the equivalence between all the models is broken (for linear-time reductions, not for poly-time reductions).

A fundamental motivation for concurrency is gaining time by executing several actions at the same moment. A concurrent machine with N processors will never compute more functions than a sequential machine[1] nor will it change the complexity class of an algorithm since speedup is in general[2] at most linear (by a factor of at most N). But a concurrent machine might exhibit some very important dynamic properties that sequential machines would only implement in a fairly trivial manner. For instance a two processor machine might deadlock because each processor is waiting for a value that the other processor is holding. A two processor machine with shared memory may also have unpredictable behaviour if the concurrent accesses to shared items are not carefully taken

[1] By Church's thesis.

[2] There are combinatorial search problems where one processor can inform other processors to stop searching and for which speedup can be more than linear.

care of. Finally, the speedup can only be determined by modeling in a precise manner which actions may overlap in time.

Sequential models are not very well suited for all this and more specific models have appeared for dealing with some aspects of concurrent computation. Obviously, any of the existing set theories would suffice for modeling concurrent machines: we could well use as many predicates and logical formulas we may think of to derive properties of programs. This was Floyd's view for instance but it is not the view we take here.

We follow here the approach of [SNW94] and [WN94]. In order to compare models and see what kind of property they are describing, we define a notion of observation of programs expressed in a natural manner in the models, through morphisms defining the allowed transformations from an object of the model to another. This makes models into categories and questions then naturally arise about what *natural*[3] constructions we can make in these models. Surprisingly enough, most of the categorical (i.e. natural) constructions bear striking resemblance with combinators of process algebras and give models of fragments of linear logic. We now review some of these models before concentrating on a few basic ones in Chapter 3.

Most of the models somehow include a notion of state of the program or machine described. In some models, a given state can only occur once and for all. In some others, a state can occur repeatedly because it does not include the history of all completed actions but rather contains only the accessible information at the time the program has reached it. For instance the state might be the values of all variables used. The classification used in [SNW94] is between *behaviour* model (states occur once) and *system* model (states occur repeatedly). Examples of the former are all kinds of event based models like event structures [Win88], geometric automata [Gun92], pomsets [Pra86], event spaces [Pra91a] all kinds of trace based models like Mazurkiewitz traces [Maz88], synchronization trees [Mil80], Hoare languages [Hoa81] etc. Examples of the latter kind are Petri nets [BRG87], all kinds of transition systems like transition systems [Kel76], asynchronous transition systems [Bed88], concurrent machines [Shi85], trace automata [Shi85] transition systems with independence [SNW94] etc.

We begin by looking at the most well known models (coming directly from the sequential world) i.e. the transition system based models.

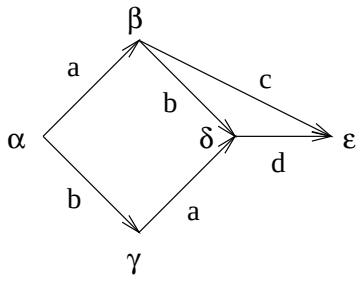
1.1 A few transition systems

1.1.1 Ordinary transition systems

Transition systems are one of the most famous models of computation. They are nothing but state transition graphs and can be pictured as such. Look for instance at Figure 1.1. We have five states α, β, γ, δ and ϵ representing, for example, the value of the variables of a program at different times of its

[3] Not using any kind of coding. The natural constructions should be explainable only in terms of allowed transformations/observations.

Figure 1.1: A transition system



execution. We have also six transitions, two as, two bs, c and d. a, b, c and d are four instructions of a program which change the state of the machine from the source of its corresponding arrow to its target.

This is generally formalized using a transition relation. We use the same notations as in [WN94].

Definition 1 *A transition system is a structure (S,i,L,Tran) where,*

- *S is a set of states with initial state i*
- *L is a set of labels, and*
- *$Tran \subseteq S \times L \times S$ is the transition relation*

Transition systems are made into a category by defining morphisms to be some kind of simulation [WN94]. The idea is that a transition system T_1 simulates a transition system T_0 if as soon as T_0 can fire some action a in some context, then T_1 can fire a as well in some related context. A morphism $f : T_0 \to T_1$ defines the way states and transitions of T_0 are related to states and transitions of T_1.

Definition 2 *Let $T_0 = (S_0, i_0, L_0, Tran_0)$ and $T_1 = (S_1, i_1, L_1, Tran_1)$ be two transition systems. A morphism $f : T_0 \to T_1$ is a pair $f = (\sigma, \lambda)$ where,*

- $\sigma : S_0 \to S_1$,
- $\lambda : L_0 \to L_1$ *are such that* $\sigma(i_0) = i_1$ *and*

$$(s, a, s') \in Tran_0 \Rightarrow (\sigma(s), \lambda(a), \sigma(s')) \in Tran_1$$

This definition differs from the one of [WN94] in that it rules out "partial" morphisms. Partial morphisms allow T_1 to be idle when T_0 carries on computation.

Definition 3 *Let $T_0 = (S_0, i_0, L_0, Tran_0)$ and $T_1 = (S_1, i_1, L_1, Tran_1)$ be two transition systems. A partial morphism (or morphism in [WN94]) $f : T_0 \to T_1$ is a pair $f = (\sigma, \lambda)$ where,*

- $\sigma : S_0 \to S_1$,

- $\lambda : L_0 \rightharpoonup L_1$ *is a* partial *function.* (σ, λ) *are such that*
  - $\sigma(i_0) = i_1$,
  - $(s, a, s') \in Tran_0$ *and* $\lambda(a)$ *is defined implies* $(\sigma(s), \lambda(a), \sigma(s')) \in Tran_1$. *Otherwise, if* $\lambda(a)$ *is undefined then* $\sigma(s) = \sigma(s')$.

As in [WN94], the difference between the two kinds of morphisms can be fixed by adding "idle" transitions to transition systems, very similar in spirit to the lifting of domains in denotational semantics [GS90, Plo84], where partial functions from D to D are considered total (and strict) from $D_\perp$ to $D_\perp$ where $\perp$ is a new element such that $\forall x,\ \perp \leq x$.

An idle transition is a transition $*$ such that $*$ goes from a state s to the same state s: "it does not change the state of the machine". Consider the following completion $T_* = (S_*, i_*, L_*, Tran_*)$ of a transition system $T = (S, i, L, Tran)$,

- $S_* = S$,
- $i_* = i$,
- $L_* = L \cup \{*\}$,
- $Tran_* = Tran \cup \{(s, *, s)/s \in S\}$.

It basically adds idle transitions to each state of an automaton. Now, a morphism $f = (\sigma, \lambda)$ (with λ a total function) from $(T_0)_*$ to $(T_1)_*$ such that $\lambda(*) = *$ is the same as a partial morphism f' from T_0 to T_1 by identifying $*$ with "undefined". Conversely, a partial morphism $f = (\sigma, \lambda)$ from T_0 to T_1 can be identified with $f_* = (\sigma, \lambda_*)$, $\lambda_*(x) = *$ if and only if $\lambda(x)$ is undefined (look at Figure 1.2 for an example). It is then obvious that the categorical constructions with partial morphisms will be the same as the categorical constructions with (total) morphisms on "lifted" transition systems.

For the sake of simplicity, we will not use these extensions to our "total" morphisms, though we will make an attempt to add them in Chapter 10. We write TS for the category of transition systems with "total" morphisms. We name TS_A its subcategory where we restrict to transition systems labeled on an alphabet A.

One of the aspects of the classification of concurrent models of [SNW94] deals directly with concurrency. Some models are only simulating concurrency and are called *interleaving* models whereas others distinguish concurrent executions from simulations on a one processor machine and are called *non-interleaving* or *truly concurrent models*. In transition systems, one can simulate the parallel execution of two actions a and b as the "interleaving" a then b or b then a (see Figure 1.3). Interleaving models rely on the notion of *indivisible* or *atomic* actions, which makes them unsatisfactory for practical use where we would like to be able to abstract away processes from unnecessary details at first, and then refine the semantics when we ask for more precision. Interleaving models are compelled to detail every part of a program execution [vGG89]. Moreover,

Figure 1.2: A partial morphism of transition systems and its corresponding total morphism

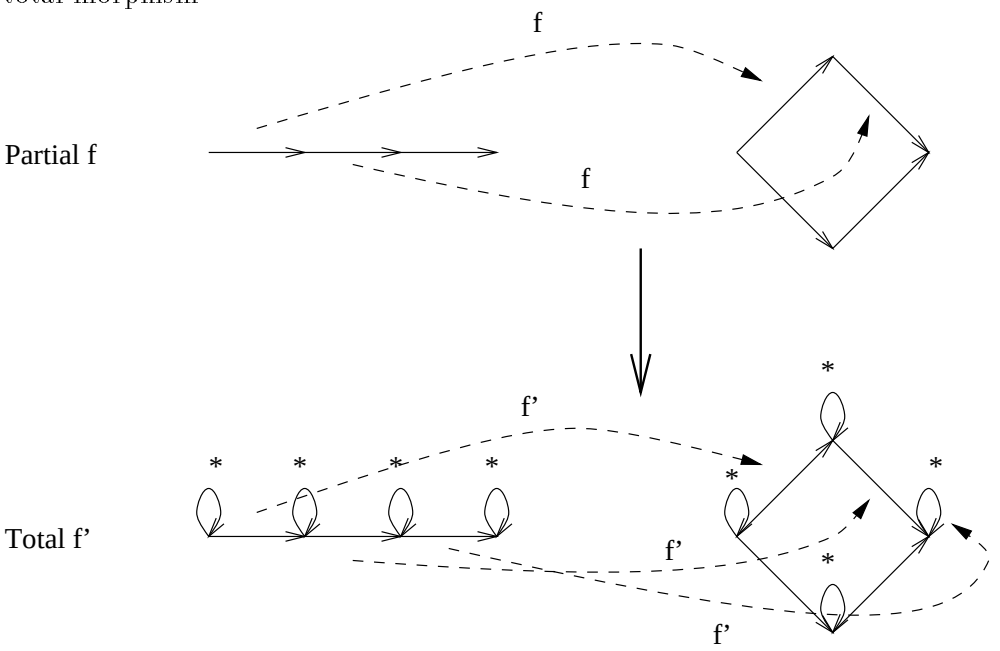
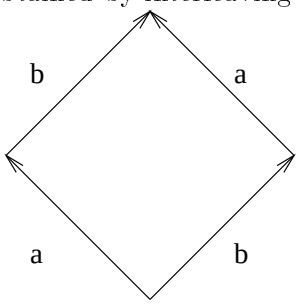


Figure 1.3: Transition system obtained by interleaving two actions



interleaving models are not suited for giving semantics to most distributed programs since equating all the possible executions with some linear ordering of atomic actions means imposing a global clock as a ruler of the system. Finally, no possible discussion of the allocation of processes on the processors is possible since all executions in the interleaving approach are equivalent to some execution on a one processor machine.

The answer to this last problem is not quite given by all truly concurrent methods as we will show in Chapter 3. We really have to define a *level of parallelism* as well as a *level of mutual exclusion*. We propose here to refine this third axis in the classification of Winskel et al. [SNW94] into a real "level of parallelism/level of mutual exclusion" axis. Let us first review some of the truly-concurrent transition systems.

1.1.2 Asynchronous transition systems

Asynchronous transition systems were introduced independently in [Bed88] and [Shi85]. They can be thought of as generalizations of Mazurkiewicz trace languages to be discussed in Section 1.2.2. The important thing is that they actually do distinguish between the interleaving of two actions and their truly-concurrent execution. This is coded using a binary independence relation. The following definition is taken from [WN94].

Definition 4 *An asynchronous transition system is (S,i,E,I,Tran) where*

- (S,i,E,Tran) is a transition system and

- $I \subseteq E \times E$ is an irreflexive symmetric relation (the "independence" relation) such that,

(1) $e \in E \Rightarrow \exists s, s' \in S, (s, e, s') \in Tran$

(2) $(s, e, s') \in Tran \wedge (s, e, s'') \in Tran \Rightarrow s' = s''$

(3) $e_1 I e_2 \wedge (s, e_1, s_1) \in Tran \wedge (s, e_2, s_2) \in Tran \Rightarrow \exists u, (s_1, e_2, u) \in Tran \wedge (s_2, e_1, u) \in Tran$

(4) $e_1 I e_2 \wedge (s, e_1, s_1) \in Tran \wedge (s_1, e_2, u) \in Tran \Rightarrow \exists s_2, (s, e_2, s_2) \in Tran \wedge (s_2, e_1, u) \in Tran$

In the following, we actually relax condition (1) stating that all events should be used.

Condition (2) says that the underlying transition system should be deterministic (e could not be a random generator). Conditions (3) and (4) are pictured respectively in Figures 1.4 and 1.5. They deal with the confluence of transitions coming from independent actions. Many different conditions of this kind can be studied. The next section deals with one of the possible refinements of those.

Morphisms f are then morphisms of transition systems preserving the independence relation I, i.e.

$$aIb \Rightarrow f(a) I' f(b)$$

Figure 1.4: Condition (3) for asynchronous transition systems

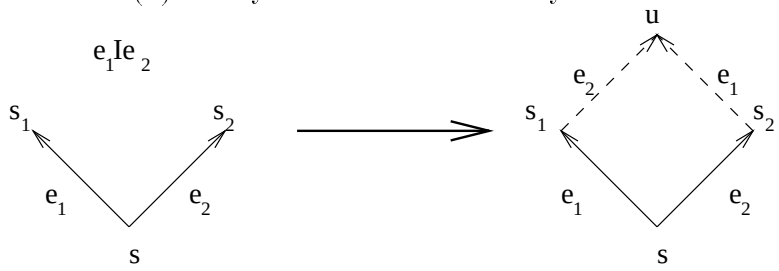
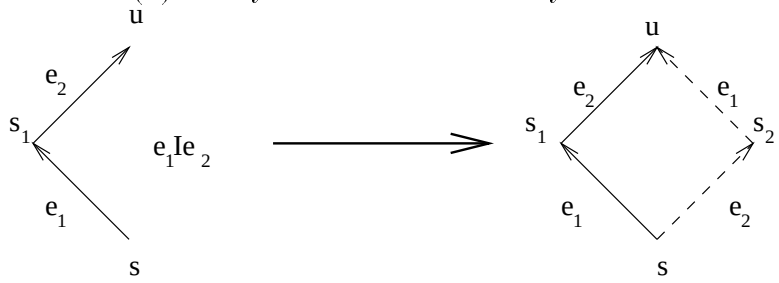


Figure 1.5: Condition (4) for asynchronous transition systems



This makes asynchronous transition systems into a category, written ATS. The category of asynchronous transition systems over an alphabet E is named ATS_E.

1.1.3 Trace automata

Trace automata are very similar to, but slightly more general than the "forward stable asynchronous systems" of [Bed88], an instance of the asynchronous transition systems of the last section. They have been mostly used for giving operational models for non-deterministic dataflow networks [Kah74, KM77].

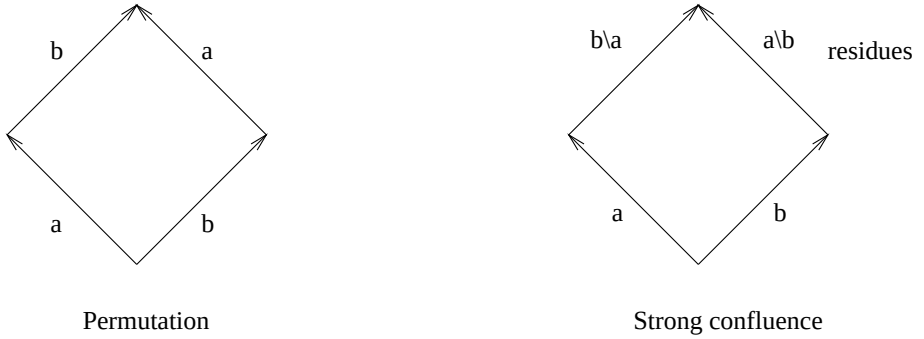
Definition 5 *A trace automaton is a tuple $A = (E, Q, T)$ where,*

- *E is a concurrent alphabet, i.e. a set of events equipped with a symmetric, irreflexive binary relation $\|_E$ called the concurrency relation,*
- *Q is a set of states,*
- *$T \subseteq Q \times (E \cup \{\epsilon\}) \times Q$ is a set of transitions. Transitions are pictured the usual way with arrows.*

These data are required to satisfy the following conditions,

- *$q \xrightarrow{\epsilon} r$ if and only if $q = r$,*
- *if $q \xrightarrow{a} r$ and $q \xrightarrow{a} r'$ then $r = r'$ (similar to condition (2) of asynchronous transition systems),*

Figure 1.6: Permutation equivalence and residuals



- *for all states q and events a, b, if $a\|_E b$, $q \xrightarrow{a} r$ and $q \xrightarrow{b} s$ then for some state p there exist transitions $s \xrightarrow{a} p$ and $r \xrightarrow{b} p$ (similar to condition (3) of asynchronous transition systems).*

We can define a notion of permutation equivalence on traces of these trace automata. This equivalence equates traces which are "essentially the same". As a matter of fact, when two actions a and b are independent, i.e. $a\|_E b$ then ab and ba are just two sequential views of the same parallel execution, and therefore should be equated. This equivalence is defined as the least congruence $\sim$ with respect to concatenation of finite traces such that $q \xrightarrow{a} r \xrightarrow{b} p$ and $q \xrightarrow{b} s \xrightarrow{a} p$ are $\sim$-related if $a\|_E b$. We will see again such equivalence relations between traces later on (Chapter 7). They are related to scheduling problems in concurrent systems and serializability.

The quotient of the traces by $\sim$ with the internal law induced by concatenation is a partially commutative monoid (as we will see in Mazurkiewitz trace theory). The permutation equivalence is actually generated by the more interesting permutation preorder defined as follows.

If $\subseteq$ is the inclusion of traces then define the permutation preorder $\sqsubseteq$ to be the transitive closure of $\subseteq \cup \sim$. The set of equivalence classes of traces of a trace automata with the permutation preorder is then a Scott domain, and even an event domain (a domain of configuration of an event structure, see Section 1.2.1). This gives a relation with the models of λ-calculus [Bar84].

Operationally, this view can be refined with the notion of residual. Given two traces t and u which begin at the same state, the residual of t by u is what is left of t after the part of u that overlaps with it has been "cancelled". In particular, $t \sqsubseteq u$ if and only if the residual of t by u is essentially nothing (an identity in the formalization of the following section).

Again it has much of the flavour of the residuals of λ-calculus as defined in [L$\acute{7}$8] which were designed to help understand confluence (look at Figure 1.6). The residual operation has been formalized and leads to the concurrent transition systems of next section.

1.1.4 Concurrent transition systems

These were introduced as a truly-concurrent operational model for concurrency in [Sta89].

Definition 6 *A concurrent transition system (CTS) is a structure* $(G, \uparrow)$ *where,*

- $G = (O, A, dom, cod, id)$ *is a graph with identities i.e.,*
  - O *is the set of proper states,*
  - A *is the set of proper transitions,*
  - $dom : A \to O$ *maps transitions to their start states,*
  - $cod : A \to O$ *maps transitions to their final states,*
  - $id : O \to A$ *maps each* $s \in O$ *to a distinguished transition (the idle transitions of section 1.1.1)* id_s *such that* $dom(id_s) = s$ *and* $cod(id_s) = s$.
- $\uparrow : Coin(G^\#) \to A^\#$ *is the* residual operation *where,*
  - $G^\#$ *is the augmented graph* $(O^\#, A^\#, dom, cod, id)$ *with,*
    - $O^\# = O \cup \{\Omega\}$ *(Ω does not belong to O),*
    - $A^\# = A \cup \{\omega_q / q \in O^\#\}$, $dom(\omega_q) = q$, $cod(\omega_q) = \Omega$.
  - $Coin(X)$ *(where X is a graph) is the set of* coinitial *transitions, i.e. the set of pairs* (t, u) *of transitions* t, u *of* X *which have the same start states.*

subject to the following conditions,

(1) for all $t \in A^\#$ *and* $u \in A^\#$ *(see Figure 1.6),*

  (a) $dom(t \uparrow u) = cod(u)$,

  (b) $cod(t \uparrow u) = cod(u \uparrow t)$.

(2) for all $t : q \to r \in A^\#$,

  (a) $id_q \uparrow t = id_r$,

  (b) $t \uparrow id_q = t$,

  (c) $t \uparrow t = id_r$.

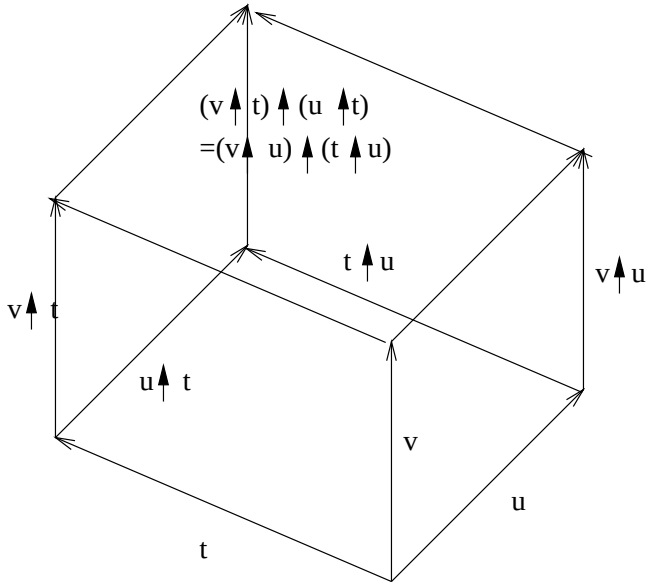
(3) for all coinitial t, u, v *in* $A^\#$, $(v \uparrow t) \uparrow (u \uparrow t) = (v \uparrow u) \uparrow (t \uparrow u)$ *(the "cube axiom", see Figure 1.7)*

(4) for all coinitial t, u *in* $A^\#$, *if* $t \uparrow u$ *and* $u \uparrow t$ *are both identities then* $t = u$.

Coinitial transitions t, u of a CTS are called consistent if $t \uparrow u$ is a proper transition (i.e. is in A and not in $A^\#$).

As usual, morphisms are simulations,

Figure 1.7: Cube axiom



Definition 7 *A morphism of concurrent transition systems is a pair of maps* $\rho = (\rho_O, \rho_A) : (O, A, dom, cod, id, \uparrow) \to (O', A', dom', cod', id', \uparrow')$ *such that,*

- $\rho_O : O \to O'$ *and* $\rho_A : A \to A'$ *are functions such that* $dom' \circ \rho_A = \rho_O \circ dom$, $cod' \circ \rho_A = \rho_O \circ cod$ *and* $\rho_A \circ id = id' \circ \rho_O$ *(simulation of the underlying state transition graph, a "total morphism" of Section 1.1),*
- *if* t, u *are consistent proper transitions of* C *then* $\rho(t \uparrow u) = \rho(t) \uparrow' \rho(u)$.

We extend morphisms to non-proper transitions by taking $\rho(\omega_q) = \omega_{\rho(q)}$.
This makes concurrent transition systems into a category we denote CTS.

1.1.5 Transition systems with independence

Transition systems with independence are transition systems enriched with a notion of concurrency in order to model true concurrency. This model has been introduced in [SNW94] and can be considered as a variation of such decorated transition systems as concurrent automata [Sta89] or asynchronous transition systems [Bed88]. We recall their formal definition, which can be found in [SNW94] or [WN94]. The independence relation between actions is now a function of the state as well. This refines a lot the precision of the model.

Definition 8 *A transition system with independence is a structure* $(S, s^I, L, Tran, I)$ *where* $(S, s^I, L, Tran)$ *is a transition system and* $I \subseteq Tran^2$ *is an irreflexive, symmetric relation such that*

(i) $(s, a, s') \sim (s, a, s'') \Rightarrow s = s''$ *(condition (2) of asynchronous transition systems),*

(ii) $(s,a,s')I(s,b,s'') \Rightarrow \exists u\,(s,a,s')I(s',b,u) \wedge (s,b,s'')I(s'',a,u)$ *(similar to condition (3) of asynchronous transition systems),*

(iii) $(s,a,s')I(s',b,u) \Rightarrow \exists s''(s,a,s')I(s,b,s'') \wedge (s,b,s'')I(s'',a,u)$ *(similar to condition (4) of asynchronous transition systems),*

(iv) $(s,a,s') \sim (s'',a,u)I(w,b,w') \Rightarrow (s,a,s')I(w,b,w')$.

where $\sim$ *is least equivalence on transitions including the relation* R *defined by,*

$$(s,a,s')R(s'',a,u) \Leftrightarrow \begin{cases} (s,a,s')I(s,b,s'') & and \\ (s,a,s')I(s',b,u) & and \\ (s,b,s'')I(s'',a,u) & \end{cases}$$

Morphisms are defined as being the total morphisms defined in [SNW94]: these are morphisms of the underlying transition system which preserve independence, i.e., pair of maps (σ,λ) with σ, a map between states, and λ a map between labels such that

- $(s,a,s') \in Tran \Rightarrow (\sigma(s),\lambda(a),\sigma(s')) \in Tran'$
- $(s,a,s')I(\overline{s},b,\overline{s}') \Rightarrow (\sigma(s),\lambda(a),\sigma(s'))I(\sigma(\overline{s}),\lambda(b),\sigma(\overline{s}'))$

We call TSI the category of transition systems with independence with total morphisms.

Now, to be complete, we review the Petri nets model as they are system models and truly concurrent models of concurrency, as all these transition systems. They can be thought of as "distributed transition systems". This view has been taken in [DDNM88] in order to give a truly concurrent semantics of CCS (Section 1.3) using Petri nets.

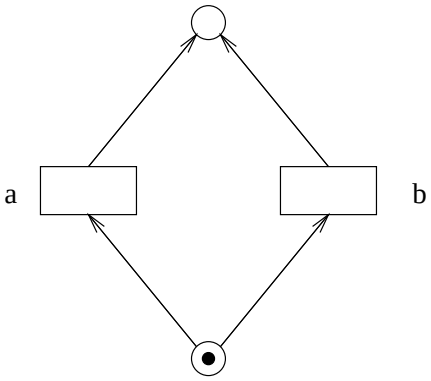
1.1.6 Petri nets

If they can be considered as distributed systems, the formalism they rely on is quite different and is based on a "token game".

Definition 9 *A Petri net* $N = (P,T,pre,post)$ *consists of,*

- P *is a set of places,*
- T *is a set of transitions,*
- $pre : T \to \tilde{P}$ *is the pre-condition map, where* $\tilde{P}$ *denotes the set of multisets of* P*,*
- $post : T \to \tilde{P}$ *is the post-condition map.*

Figure 1.8: A Petri net graphical representation: mutual exclusion between a and b



The places represent resources which might be used by one or more processes. This is formalized by the notion of *marking*. A marking is a multiset of places.

The pre-condition map describes how transitions "consume" resources and the post-condition map shows how transitions "create" new resources. This defines a transition relation between markings. If M and M' are two markings of some net N, and t is a transition of N we write $M[t\rangle M'$ for "t fires from M to M'" if and only if

$$\exists M'' \in \tilde{M}, M = M'' + pre(t) \text{ and } post(t) + M'' = M'$$

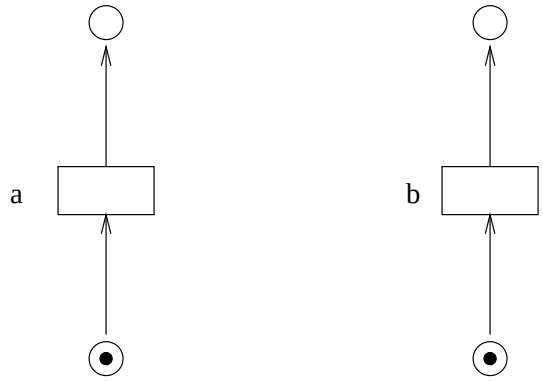
Petri nets are generally represented graphically (see Figure 1.8 and Figure 1.9) as follows,

- places are circles,
- transitions are squares,
- arcs from places to transitions with a suitable multiplicity are used to represent the pre-condition map,
- arcs from transitions to places (again with multiplicity) represent the post-condition map,
- markings are tokens put in suitable places.

As many of the models we have already discussed, Petri nets exist in several versions. The one we have just defined is very often called P/T nets, standing for Place/Transition nets. A particular case of it is C/E nets, i.e. Condition/Event nets. Places are now conditions which can only be true or false. True is identified with one token, and false with no token occupying a place. Therefore, C/E nets are particular P/T nets in which there can be at most one token per condition. A special case of C/E nets is an occurence net. It is a C/E net in which a transition can only be fired exactly once.

P/T, C/E and occurence nets can be made into categories with the notion of morphism that we define below.

Figure 1.9: A Petri net graphical representation: truly concurrent execution of a and b



All Petri nets we consider now will be given with an initial marking M_0, so that we write a net as $N = (P, T, M_0, pre, post)$. Following [WN94] we define morphisms to be maps preserving initial markings and events when defined.

Definition 10 *Let $N = (P, T, M_0, pre, post)$ and $N' = (B', M_0', E', pre', post')$ be two nets. A morphism $f : N \to N'$ consists of,*

- *a relation $\beta \subseteq B \times B'$ such that β^{op} is a partial function from B' to B,*
- *a partial function $\eta : E \to E'$ such that,*
  - $\beta M_0 = M_0'$,
  - $\beta \circ pre(e) = pre \circ \eta(e)$,
  - $\beta \circ post(e) = post \circ \eta(e)$.

As usual in this chapter, we restrict to morphisms which are total functions. This means that β and η in the above definition are total functions. We have an isomorphic category of Petri nets if all nets are "lifted" by adding an idle event $*$ with $pre(*) = post(*) = \emptyset$.

As a direct consequence of the definition, morphisms preserve the transition relation i.e. if $M[e\rangle M'$ in N then $\beta M[\eta(e)\rangle \beta M'$ in N'.

The category of Petri nets with morphisms described as in Definition 10 is named PN, the one with total morphisms is PN_t.

Petri nets have inspired most of the event based models. We present a few of them below.

1.2 Behavioural models

1.2.1 Event structures

Event structures describe a concurrent system through the occurrence of some events, like "some action has taken place", "the state of the machine has been changed" etc. This is done through a partial ordering $\leq$ on the set of events E.

When two events are not related by $\leq$, they are candidates for being executed in parallel, since no event has to precede the other. To be able to model mutual exclusions as well, we use a conflict relation $\#$ between events. $e_0 \# e_1$ if and only if e_0 and e_1 cannot take place at the same moment.

Then events e_0 and e_1 are truly concurrent when one is not before the other and if they are not in conflict. This can be written in symbols as a "concurrency" relation *co*:

$$(e_0 \; co \; e_1) \text{ iff } not\,((e_0 \leq e_1) \text{ or } (e_1 \leq e_0) \text{ or } (e_0 \# e_1))$$

The formal definition is as follows,

Definition 11 *A prime event structure is a structure* $(E, \leq, \#)$ *where* E *is a set of* events *partially ordered by* $\leq$ *called the* causal dependency relation *and where* $\# \subseteq E \times E$ *is a symmetric irreflexive relation, the* conflict relation *satisfying,*

- $\{e'/e' \leq e\}$ *is finite (axiom of "finite causes"),*
- $e \# e'$ *and* $e' \leq e''$ *implies* $e \# e''$ *(conflict is hereditary).*

A labeled event structure $(E, \leq, \#, l, L)$ is composed of an event structure $(E, \leq, \#)$, a set of labels L and a labelling function $l : E \to L$.

Then morphisms of event structures are defined as follows.

Definition 12 *Let* $S = \{E, \leq, \#\}$ *and* $S' = \{E', \leq', \#'\}$ *be event structures. A morphism of event structures from* S *to* S' *is a partial function* $f : E \to E'$ *such that,*

- *if* $f(e)$ *is defined then* $\{e'/e' \leq f(e)\} \subseteq f(\{e''/e'' \leq e\})$,
- *if* $f(e_0)$ *and* $f(e_1)$ *are both defined then* $f(e_0) \# f(e_1)$ *or* $f(e_0) = f(e_1)$ *implies* $e_0 \# e_1$ *or* $e_0 = e_1$.

Here again, we can restrict to "total" morphisms, i.e. to functions $f : E \to E'$ such that,

- $\{e'/e' \leq f(e)\} \subseteq f(\{e''/e'' \leq e\})$,
- $f(e_0) \# f(e_1)$ or $f(e_0) = f(e_1)$ implies $e_0 \# e_1$ or $e_0 = e_1$.

Morphisms of labeled event structures are pairs

$$(\eta, \lambda) : (E_0, \leq_0, \#_0, l_0, L_0) \to (E_1, \leq_1, \#_1, l_1, L_1)$$

such that η is a morphism of event structures from $(E_0, \leq_0, \#_0)$ to $(E_1, \leq_1, \#_1)$ and $\lambda : L_0 \to L_1$ is a function satisfying $\lambda \circ l_0 = l_1 \circ \eta$.

This forms the category LES of labeled event structures.

This is the first "non-operational" model of concurrency we have been considering. But it is actually quite easy to recover the operational intuition in the event based models. We only have to collect "compatible" events, linearly ordered by the time at which they may happen. This leads to the definition of *configurations*.

Let $(E, \leq, \#)$ be an event structure. Its set of configurations $\mathcal{D}(E, \leq, \#)$ is the set of those subsets $x \subseteq E$ which are,

- conflict-free: for all $e, e' \in x$, e is not in conflict with e',
- downwards-closed: for all e, e', $e \in x$ and $e' \leq e$ implies $e' \in x$

Following [WN94] we write $\mathcal{D}^0(E, \leq, \#)$ for the set of finite configurations.

Now the notion of causal dependency in an execution of an event structure is given through the "enabledness" relation.

Let $e \in E$ and $c \in \mathcal{D}(E, \leq, \#)$ then we say that e is enabled at a configuration c, written $c \vdash e$ if,

(i) $e \notin c$,

(ii) $\{e'/e' \leq e \wedge e' \neq e\} \subseteq c$,

(iii) $e' \in E$ and $e' \# e$ implies $e' \notin c$

Finite configurations are traces when we linearly order their elements by causal dependency. $\{e_1 < e_2 < \ldots < e_n\}$ is a *securing* for c if and only if $\{e_1, \ldots, e_{i-1}\} \vdash e_i$ for $i = 1, \ldots, n$. We write also securings as strings $e_1 \ldots e_n$.

Events are one side of a duality [Pra92] for which automata are the other side. A general framework in which automata and schedules (event based models) fit very nicely has been recently introduced by Vaughn Pratt under the name of Chu spaces [Pra94a, Pra94b].

The event structures we have defined are not the most general event structures described in the literature, and the discussion about configurations gives us the generalization we are looking for. The most general ones are as follows. Instead of reasonning only on partial orders of events, we consider directly partial histories, i.e. finite consistent sets of events. This is described by Con in the following definition. Con also takes the information about conflicts into account so we do not need the # relation any longer. But the dynamics has to be described now. It is addressed by the enabling relation $\models$.

Definition 13 *[Win88] An event structure is a triple $(E, Con, \models)$ where,*

- *E is a set of events,*
- *Con is a nonempty set of finite subsets of E, called the* consistency predicate *which satisfies,*

$$X \in Con \wedge Y \subseteq X \Longrightarrow Y \in Con$$

Figure 1.10: A parallel switch

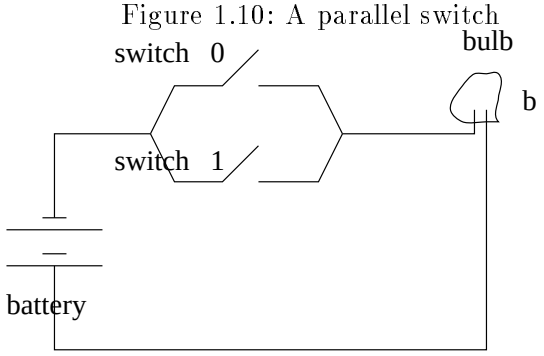
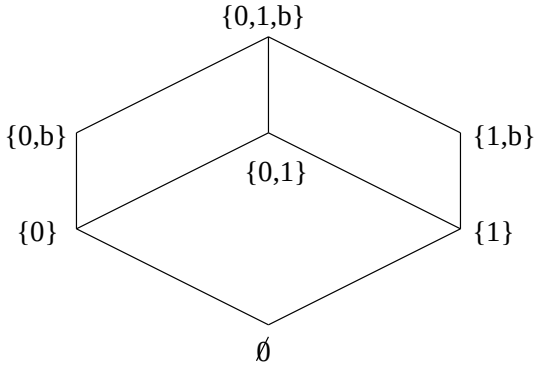


Figure 1.11: Configurations of events for the parallel switch example



- $\models \subseteq Con \times E$ *is the* enabling *relation which satisfies,*

$$X \models e \wedge X \subseteq Y \wedge Y \in Con \Longrightarrow Y \models e$$

What have we gained now?

It can be shown in a very precise manner [Win88] that these event structures are more general in the sense that an event can now be enabled in different ways. An event can be caused by more than one configuration (see Figure 1.10 and Figure 1.11). We say that event structures can exhibit OR causality whereas prime event structures cannot.

This is a problem most of the models for concurrency based on partial orders have (like pomsets [Pra86] etc.). They can exhibit AND causality, i.e. an event can only occur if and only if some other events have all occured before. This distinction between AND and OR causality was used in [Gun92] to analyse Milner's notion of confluence (closer to our notion of serializability than to the standard notion of confluence on transition systems). In short, Confluence = Determinism+{AND,OR} Causality (here determinism is the "determinacy" of Milner, [Mil83]).

Technically, prime event structures have the same domains of configurations (the finitary prime algebraic domains) as the so-called *stable* event structures. These are event structures $(E, Con, \models)$ for which there is a partial order of causal dependency on each configuration. They satisfy the following stability

axiom which ensures that there is always a minimal set of event occurrences enabling any event,

$$X \models e \wedge Y \models e \wedge X \cup Y \cup \{e\} \in Con \Longrightarrow X \cap Y \models e$$

It is not a surprise that the domains of configurations of stable event structures are isomorphic to the dI-domains of Berry [Ber79] since the latter arose in the context of the search for a precise definition of sequentiality. As a matter of fact, one of the ancestors of event structures were precisely the *concrete data structures* of Kahn and Plotkin used to define domains of "sequential" functions (see [Cur86, BC82] for a survey).

Then, as in the sequential case, some models can be *linear* time or *branching time* i.e. some models express only deterministic behaviours while other distinguishes the time when (non-deterministic) choices are made. The event based, the transition based models and the Petri nets models above are all branching time as well as synchronization trees whereas among the trace based models only the Mazurkiewitz traces and Hoare languages are linear time.

We define below a generalised version of Mazurkiewitz trace languages as can be found in [WN94].

1.2.2 Mazurkiewitz traces

In this section, we slightly generalize the asynchronous transition systems in order to allow the "independence" relation to vary according to the local state of the machine. In the literature, this has not been developed for transition systems (except in TSI) but rather on the language side of the automata theory, i.e. the Mazurkiewitz trace theory. We introduce the generalization as it has first been done in [SNW94].

Definition 14 *A generalized trace language is a triple (M, I, L) where,*

- *L is a set of symbols,*
- *$M \subseteq L^*$,*
- *$I : M \to 2^{L \times L}$ is a function which associates to each $s \in M$ a relation $I_s \subseteq L \times L$.*

such that, if we define $\cong$ to be the least equivalence relation on L^* such that $sabu \cong sbau$ if aI_sb, and,

- *for all $s \in M$, I_s is symmetric and irreflexive,*
- *(I is consistent) $s \cong s'$ implies $I_s = I_{s'}$,*
- *(M is I-closed) aI_sb implies $sab \in M$,*
- *(I is coherent)*

  (i) aI_sb and $aI_{sb}c$ and $cI_{sa}b$ implies $aI_{sc}b$,

(ii) aI_sc *and* cI_sb *implies (*aI_sb *if and only if* $aI_{sc}b$*).*

Definition 15 *Let* (M, I, L) *and* (M', I', L') *be generalized trace languages. A partial function* $\lambda : L \to L'$ *defines a morphism from* (M, I, L) *to* (M', I', L') *if and only if,*

- λ *preserves words:* $s \in M$ *implies* $\lambda^*(s) \in M'$,
- λ *respects independence:*

  aI_sb *and* $\lambda(a)$, $\lambda(b)$ *are defined implies* $\lambda(a)I'_{\lambda^*(s)}\lambda(b)$ *where* λ^* *is an extension of* λ *on words defined as follows,*

  - $\lambda^*(\epsilon) = \epsilon$,
  - $\lambda^*(sa) = \begin{cases} \lambda^*(s)\lambda(a) & \text{if } \lambda(a) \text{ is defined} \\ \lambda^*(s) & \text{otherwise} \end{cases}$

They form the category GTL_p. It is proven in [SNW94] that GTL_p is equivalent to the category of deterministic labeled event structure defined in the previous section. We give an account of this proof in Section 3.4.

Once again, we restrict to the category GTL of generalized Mazurkiewitz traces with "total morphisms" i.e. $\lambda : L \to L'$ a function such that,

- λ preserves words i.e. $s \in M$ implies $\lambda^*(s) \in M'$,
- λ respects independence: aI_sb implies $\lambda(a)I'_{\lambda^*(s)}\lambda(b)$ where λ^* is an extension on words defined as follows,
  - $\lambda^*(\epsilon) = \epsilon$,
  - $\lambda^*(sa) = l^*(s)\lambda(a)$

GTL_L is the subcategory of GTL of generalized Mazurkiewitz traces on an alphabet L.

Now, the category of Mazurkiewitz traces can be seen as a full subcategory of GTL. Recall that a Mazurkiewitz trace language is a triple (M, I, L) where,

- L is a set of symbols,
- $M \subseteq L^*$,
- I is a symmetric irreflexive relation on L such that,
  - I is prefix-closed: $sa \in M$ implies $s \in M$ for all $s \in L^*$ and $a \in L$,
  - M is I-closed: $sabt \in M$ and aIb implies that $sbat \in M$ for all $s, t \in L^*$ and all $a, b \in L$,
  - M is coherent: $sa \in M$ and $sb \in M$ and aIb implies $sab \in M$ for all $s \in L^*$ and all $a, b \in L$

To see that Mazurkiewitz traces are a special case of generalized Mazurkiewitz traces as the name suggests, we only have to see that I defines a constant function from M to $2^{L \times L}$ satisfying axioms of Definition 14.

1.3 Semantics of a few process algebras

There has been an attempt to define a calculus for parallel processes which would be as foundational as λ-calculus for functional computation. In some way, this approach has not succeeded yet, but all process algebras introduced since at least twenty years have brought a good understanding of the basic behaviours that concurrent programs may exhibit. Some of them have even been used to give semantics to some real concurrent languages.

We present below a small selection of these process algebras for different purposes. The first one is to have a few examples of toy languages at hand for showing how to use some of the abstract models of concurrency we have described to give semantics. The second one is to show different paradigms for parallel computation that will enable us to recognize or to infer some interesting constructions in Higher-Dimensional Automata.

1.3.1 CCS

In this section we focus on pure CCS. The original language [Mil89] actually had values, variables and channels. Pure CCS is a simplification of it in which communications along channels are abstracted in the following way,

- we forget about actual values traveling on channels,
- we also forget the names of variables, leaving only the name of the channel,
- now, to receive a value into the variable x on channel a, i.e. $a?x$, is abstracted by a,
- to send a value n on channel a, i.e. $a!n$ is abstracted by the *complementary action* $\overline{a}$.

Therefore, communication is abstracted by synchronization of an action and a complementary action. This synchronization is "observable" through the occurrence of an "invisible action" τ.

The syntax of pure CCS is now (where α ranges over actions, complementary actions and τ, and f is a function on actions that commutes with $x \to \overline{x}$),

$$
\begin{array}{rcll}
t & ::= & \mathbf{nil} & \text{(idle process)} \\
 & | & t_1 + t_2 & \text{(choice operator)} \\
 & | & t_1 \mid t_2 & \text{(parallel composition)} \\
 & | & \alpha.t & \text{(prefixing)} \\
 & | & t_1 \backslash c & \text{restriction operator} \\
 & | & p[f] & \text{relabelling} \\
 & | & \mathbf{rec}\ x.t(x) & \text{recursive agent}
\end{array}
$$

Semantics The first semantics [Mil83] was given in terms of synchronization trees (acyclic transition systems). It is very often given using SOS rules [Plo81] describing transition systems as follows,

$$\frac{}{\alpha.t \xrightarrow{\alpha} t} \tag{1.1}$$

$$\frac{t_i \xrightarrow{\alpha} t_i'}{t_1 + t_2 \xrightarrow{\alpha} t_i'} \tag{1.2}$$

$$\frac{t_1 \xrightarrow{\alpha} t_1'}{t_1 \mid t_2 \xrightarrow{\alpha} t_1' \mid t_2} \tag{1.3}$$

$$\frac{t_2 \xrightarrow{\alpha} t_2'}{t_1 \mid t_2 \xrightarrow{\alpha} t_1 \mid t_2'} \tag{1.4}$$

$$\frac{t_1 \xrightarrow{\alpha} t_1' \qquad t_2 \xrightarrow{\overline{\alpha}} t_2'}{t_1 \mid t_2 \xrightarrow{\tau} t_1' \mid t_2'} \tag{1.5}$$

$$\frac{t \xrightarrow{\alpha} t' \qquad \alpha \neq c \text{ and } \alpha \neq \overline{c}}{t\backslash c \xrightarrow{\alpha} t'\backslash c} \tag{1.6}$$

$$\frac{t \xrightarrow{\alpha} t'}{t[f] \xrightarrow{f(\alpha)} t'[f]} \tag{1.7}$$

$$\frac{\mathbf{rec}\ x.t[x] \xrightarrow{\alpha} t'}{t[\mathbf{rec}\ x.t[x]] \xrightarrow{\alpha} t'} \tag{1.8}$$

Rule 1.1 indicates that $\alpha.t$ can fire an α action at first.

Rule 1.2 shows that $t_1 + t_2$ behaves as t_1 or t_2 once and for all.

Rules 1.3 and 1.4 define $t_1 \mid t_2$ by their interleaving.

Rule 1.5 takes care of the synchronization between complementary actions, which when "annihilating" each other produce a silent τ action.

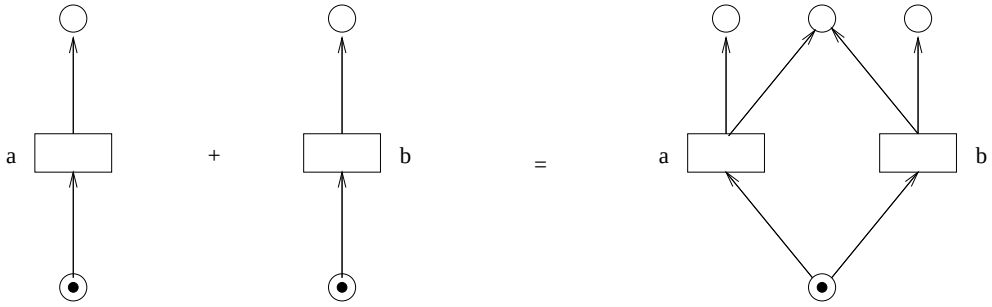
The restriction operator is defined by Equation 1.6. The restriction applies both on an action and its corresponding complementary action.

Rule 1.7 defines in an obvious manner the relabelling operator.

Finally **rec** $x.t[x]$ is similar to the "infinite term" $t[t[\ldots[\mathbf{nil}]]]$. This is Equation 1.8.

A "denotational" semantics can also be given in ordinary transition systems using their categorical properties. We recall here the basic categorical constructs we have with partial morphisms of ordinary transition systems and their interpretations as can be found in [WN94],

Figure 1.12: Coproduct of two Petri nets



- the cartesian product is a form of synchronized product plus interleaving (those transitions with a $*$ transition as one of their components),
- the fibered coproduct is the non-deterministic choice of CCS,
- the CCS restriction operator is obtained through a strong cartesian lifting,
- the CCS relabelling operator corresponds to a strong cocartesian lifting construction,
- the CCS prefixing and parallel operators do not really correspond to any categorical combinator in the category of transition systems.

A "denotational" semantics can also be given using Petri nets even if categorical constructions in PN are not easy to find. But it is shown in [WN94] that at least the following constructs are available,

- the coproduct of two nets is roughly the non-deterministic sum of the nets (see Figure 1.12 and Figure 1.13 for an example). The behaviours are really those of a non-deterministic choice only for the so called *safe* nets (see [WN94]).
- the product of two nets corresponds to a synchronization of the two nets. An example is given in Figure 1.14.

A different approach using Petri nets has been used in [DDNM88]. The view taken there is that Petri nets are nothing but distributed transition systems. Then the CCS-terms are decomposed into local processes which are given a standard kind of operational semantics. Some other approaches with Petri nets account for subsets of CCS only as in [GM84] and [DCDMPS83].

Trying to be as complete as possible, a semantics using event structures has been given in [Win88] and semantics using partial orderings have been given in [DDNM85] and [DM87]. In [Gup94], a semantics is also given in terms of Chu Spaces.

We will see in Chapter 5 how to give a truly-concurrent operational semantics of CCS both in denotational and SOS form with the model we introduce in next chapter.

Figure 1.13: Allowed transitions of Figure 1.12

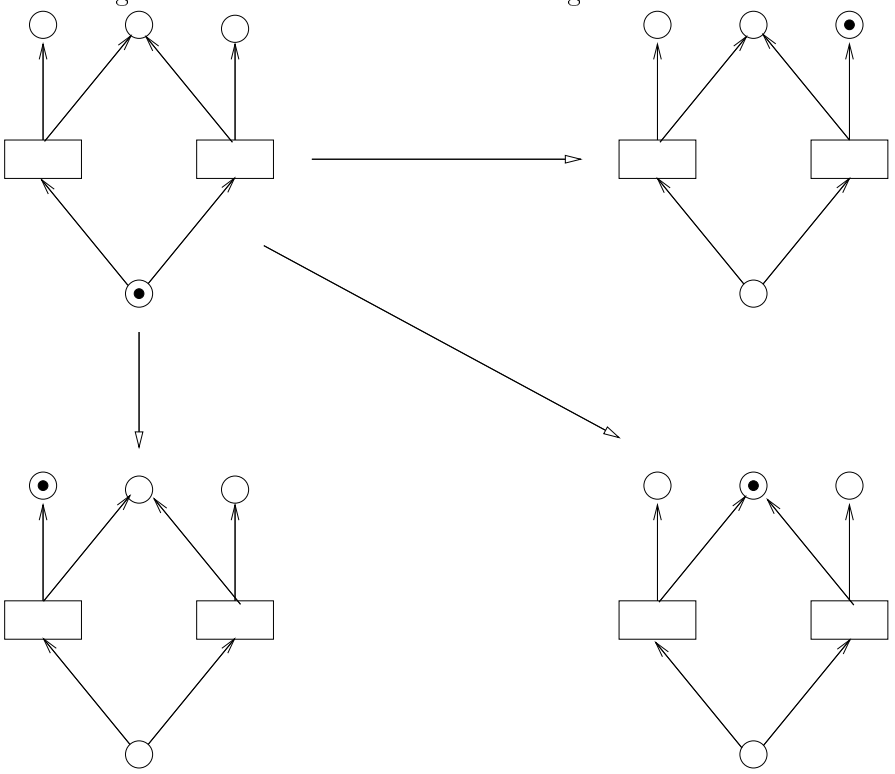
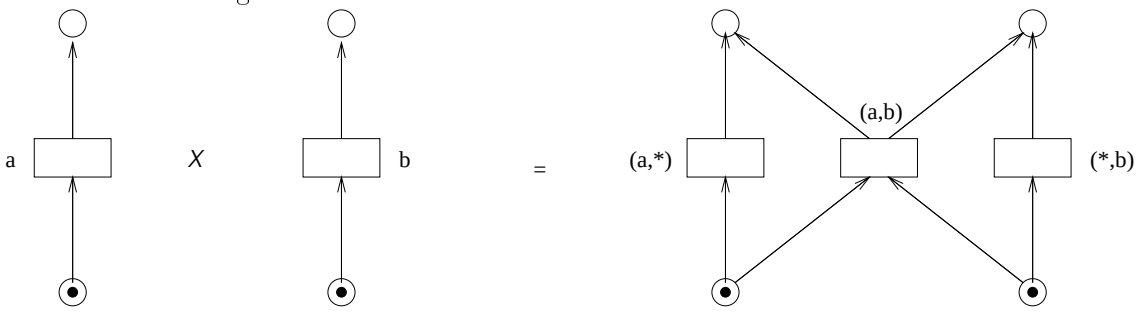


Figure 1.14: Product of two Petri nets



1.3.2 Communicating Sequential Processes

This calculus was introduced in [Hoa85]. Several syntaxes are commonly used (from one near Dijkstra's GC to one resembling CCS). CSP has about the same primitives as CCS, but the synchronization mechanism is slightly different. The syntax of "valued" CSP is as follows,

t	$::=$	$\alpha?X \rightarrow t$	input on channel α and store in variable X, then do t
	$\mid$	$\alpha!a \rightarrow t$	output value a on channel α and then do t
	$\mid$	$t_1 \mid\mid t_2$	(parallel composition)
	$\mid$	$t_1 \Box t_2$	(choice operator)
	$\mid$	$l : t$	labelling
	$\mid$	$\mu x.t(x)$	recursive agent, for t a guarded expression

A semantics in terms of labeled transition systems can be found in [BHR84] and also for instance in [Win93].

In this calculus, we have chosen to represent values and variables. Therefore, the states of the transition system will be of the form $\langle c, \sigma \rangle$, where c is a CSP program and σ is a store (i.e. a function from variables to values).

We do not go through the whole semantics of this CSP language since it is very similar to the CCS one. We specify only the parallel operator,

$$\frac{\langle c_0, \sigma \rangle \xrightarrow{\lambda} \langle c_0', \sigma' \rangle}{\langle c_0 \mid\mid c_1, \sigma \rangle \xrightarrow{\lambda} \langle c_0' \mid\mid c_1, \sigma' \rangle}$$

$$\frac{\langle c_1, \sigma \rangle \xrightarrow{\lambda} \langle c_1', \sigma' \rangle}{\langle c_0 \mid\mid c_1, \sigma \rangle \xrightarrow{\lambda} \langle c_0 \mid\mid c_1', \sigma' \rangle}$$

CSP with no values has the following syntax,

$$\begin{array}{rcccccc} P & ::= & SKIP & \mid & STOP & \mid & a \rightarrow P \quad \mid \\ & & P;Q & \mid & P \cap Q & \mid & P \| Q \quad \mid \\ & & P \mid\mid\mid Q & \mid & P \backslash a & \mid & repeat\ P \end{array}$$

$P \mid\mid\mid Q$ denotes the interleaving of P with Q. $P \| Q$ is the lockstep synchronization of P with Q. This means that this process must synchronize each action of P and Q with the same name.

A denotational (truly-concurrent) semantics of CSP with no values using event structures has also been proposed in [Win88].

1.3.3 π-calculus

Here, we will restrict to the monadic π-calculus. We refer the reader to [Mil91] for details on how to generalize this to the polyadic case.

As Robin Milner says [Mil91],

> "The work on π-calculus really began with a failure, at the time I wrote about CCS, the Calculus of Communicating Systems [Mil80]. This was the failure, in discussion with Mogens Nielsen at Aarhus in 1979, to see how full mobility among processes could be handled algebraically. The wish to do this was motivated partly by Hewitt's actor systems, which he introduced much earlier. Several years later, Engberg and Nielsen succeeded in giving an algebraic formulation. The π-calculus is a simplification and strengthening of their work."

Let X be a set of names. Typical elements are x, y, etc. We use p, q, etc. to range over the set P of processes. They are constructed according to the following syntax,

p	$::=$	$\mathbf{0}$	empty process
	$\mid$	$\sum_{i \in I} \pi_i.p_i$	sum of finitely many processes
	$\mid$	$p \mid q$	parallel composition
	$\mid$	$!p$	replication
	$\mid$	$\nu x p$	restriction operator

The prefixes π_i in the sum above represent atomic actions which can be of the following form,

- $x(y)$ means "input some name, call it y, along the link named x",
- $\overline{x}y$ means "output the name y along the link named x"

Before defining an operational semantics (based on ordinary transition systems) we first define a structural congruence $\cong$ on terms,

- processes which only differ by a change of bound names are identified,
- $+$ and $\mid$ are commutative on the equivalence classes modulo $\cong$,
- $p + \mathbf{0} \cong p$ and $p \mid \mathbf{0} \cong p$,
- $!p \cong p \mid !p$,
- $\nu x \mathbf{0} \cong \mathbf{0}$ and $\nu x \nu y p \cong \nu y \nu x p$,
- if x is not a free name in p then $\nu x(p \mid q) \cong p \mid \nu x q$.

As for CCS, syntactic terms will be states of the transition system, but this time, they will be taken modulo the structural congruence $\cong$.

The transitions are now expressed in SOS form as follows,

$$\frac{}{(\ldots + x(y).p) \mid (\ldots + \overline{x}z.q) \longrightarrow p[z/y] \mid q}$$

$$\frac{p \longrightarrow p'}{p \mid q \longrightarrow p' \mid q}$$

$$\frac{p \longrightarrow p'}{\nu x p \longrightarrow \nu x p'}$$

$$\frac{q \cong p \quad p \longrightarrow p' \quad p' \cong q'}{q \longrightarrow q'}$$

Example 1 *The following example (taken from [Mil91]) exemplifies the kind of mobility that can be achieved using the π-calculus.*

Consider the terms $P = \overline{x}y.\mathbf{0}$, $Q = x(u).\overline{u}v.\mathbf{0}$ and $R = \overline{x}z.\mathbf{0}$ and the process $X = P \mid Q \mid R$.

P can send y to Q and R can send z to Q, but not both. Therefore the two alternatives for the result are $\mathbf{0} \mid \overline{y}v.\mathbf{0} \mid \overline{x}z.\mathbf{0}$ or $\overline{x}y.\mathbf{0} \mid \overline{z}v.\mathbf{0} \mid \mathbf{0}$. R has thus become $\overline{y}v.\mathbf{0}$ or $\overline{z}v.\mathbf{0}$. The communication has determined which channel R can next use for output, y or z.

1.4 Real-time systems

By real-time system we mean here a sequential or concurrent machine whose states depend on one or several *clocks*, i.e. whose evolution is constrained by (some measures of) *time*. Examples are everywhere in real life: alarm clocks, coffee machines (which give you back your money if you are too long to choose your beverage), "real-time languages" and of course ordinary computers, which always have a clock to trigger signals etc.

Restricting to real-time software and languages would give a rather partial view. In synchronous languages such as ESTEREL [BC85], LUSTRE [CHPP87] and StateCharts [Har87] the execution times of actions are considered to be insignificant with respect to the time constants of the external signals (which are actually the inputs to the programs). This approach has many benefits. In particular, it is much easier to design applications and to prove them correct with respect to timed specifications.

This simpler view has much influenced the semantic models. There are but a very few models considering that actions take time (see [Jos89] though). Here we follow [Jos89],

> "A crucial aspect of "real-ness" of many real-time systems is that they have a limited set of computational resources such as

processors, memory, channels etc. whose use must be scheduled appropriately for the real-time program to meet its deadlines. In these cases, the real-time program must be seen as a concurrent program which is executed on a system with limited resources and it is necessary for the limitations to be represented in the associated semantic models."

1.4.1 Models of real-time

In [Hen91], real-time models were considered good enough if they were **refinable**, **digitizable**, and **operational**. This means in particular that we should be able to look at a real-time system at different levels of precision (this rules out formalisms depending on a base of time) and that its description should be based on systems of transitions. We focus on these kinds of models below.

Timed transition systems [HMP93] assume a global fictitious real-valued clock. They are based on ordinary transition systems on which they add requirements about the minimal and maximal delays between which actions have to be enabled in order to be fired. In this model, transitions take no time. Formally, a Timed Transition System (TTS) is $S = (S, i, E, Tran, l, u)$ where,

- $(S, i, E, Tran)$ is a transition system[4],
- l is a collection $l_\tau \in \mathbb{N}$ of minimal delays for actions $\tau \in E$,
- u is a collection $u_\tau \in \mathbb{N} \cup \{\infty\}$ of maximal delays for actions $\tau \in E$

Traces, called *timed execution sequences* in [HMP93] are sequences $(\sigma_i, T_i)_{i\in\mathbb{N}}$, $\sigma_i \in S$ and $T_i \in \mathbb{R}$ such that,

- $(\sigma_i)_{i\in\mathbb{N}}$ is a trace of the underlying transition system,
- "time never decreases" i.e.

$$\forall i \in \mathbb{N}, (T_{i+1} = T_i) \vee ((T_{i+1} > T_i) \wedge (\sigma_{i+1} = \sigma_i))$$

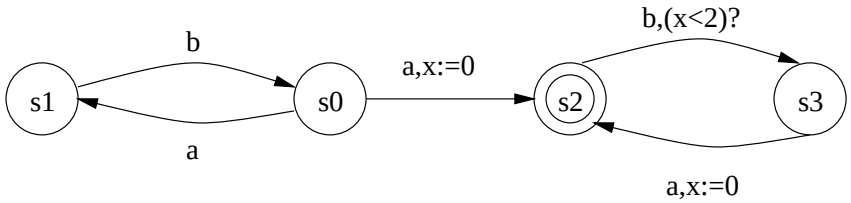
- "time diverges" i.e.

$$\forall t \in \mathbb{R}, \exists i \geq 0, T_i \geq t$$

- "a transition τ has to be enabled at least l_τ time units in order to be fired" i.e. for every transition $\tau \in E$ and $i \geq 0, j \geq i$ with $T_j < T_i + l_\tau$, if τ is taken at position j of σ then τ is enabled on σ_i,
- "a transition τ cannot be enabled for more than u_τ time units without being taken" i.e. for every transition $\tau \in E$ and $i \geq 0$, there exists $j \geq i$ with $T_j \leq T_i + u_\tau$ such that either τ is not enabled on σ_j or τ is taken at position j of σ.

[4] In fact it has been introduced in a different way in [HMP93] where states are assignments of values to variables. The transition system is also required to contain all idle transitions, i.e. transitions from state s to s, for any s.

Figure 1.15: A timed Büchi automaton



Timed automata [AD91] generalize finite state machines over infinite strings. They consist mainly of finite sets of locations and finite sets of real-valued clocks. Again, transitions take no time and states are waiting periods for clock constraints to be satisfied.

Formally, if X is a set of clocks, we define the set $\Phi(X)$ of clock constraints δ inductively as follows,

$$\delta := x \leq c \mid c \leq x \mid \neg\delta \mid \delta_1 \wedge \delta_2$$

where x is a clock in X and c is a constant in the time domain.

Then a *timed transition table* is a tuple $\langle \Sigma, S, S_0, C, E \rangle$ where,

- Σ is a finite alphabet,
- S is a finite set of states,
- $S_0 \subseteq S$ is a set of start states,
- C is a finite set of clocks and,
- $E \subseteq S \times S \times \Sigma \times 2^C \times \Phi(C)$ gives the set of transitions.

Edges, or transitions, are $(s, s', a, \lambda, \delta) \in E$ going from state s to state s' on input symbol a. $\lambda \subseteq C$ gives the clocks to be reset with this transition and δ is a clock constraint over C. Several kinds of accepting criteria can be given: Büchi, Muller etc. to these automata. Then they accept timed languages which are pairs of a language of the underlying untimed automaton and timing constraints (see Figure 1.15) which accepts the language $\{((ab)^\omega, \tau)/\exists i, \forall j \geq i, (\tau_{2j} < \tau_{2j-1} + 2\}$.

Finally, a very general and quite ad hoc model[5] for real-time systems advocated by Leslie Lamport [AL91a] consists in using TLA (Temporal Logics of Actions) as a base for specifying (and verifying) programs and in adding a new variable, *now*, denoting the value of the global clock. This is more a coding than anything else. In particular, extra-TLA formulas are necessary to describe the essential properties of *now* and some problems about non-Zenoness are difficult to solve (see the discussion in [AL91a]).

As we have seen, most of the semantic models for real-time systems assume that transitions do not take time, and states bear time changes. This view has

[5]Even if it is not directly based on transition systems, this model has strong connections with operational semantics in general.

deeply influenced the numerous process algebras that have appeared in the last few years. Before choosing one which will exemplify the use of timed higher-dimensional automata for giving semantics to toy languages, we review some of them as well as discuss their differences. We will follow the very good survey [NS92] as a basis for discussion.

All process algebras rely on a "two-phase functioning scheme". This means that their executions alternate from a synchronous part where all the components agree for the time to progress to an asynchronous part during which the progress of time is blocked. This view, which might be close to the "real" behaviour of some hybrid systems lead to technical difficulties hiding some of the real issues of real-time systems modeling. In particular *deadlock-freeness*, i.e. the fact that no process can block the progress of time is a technical difficulty of the semantical model and is certainly not an actual property of interest for real-time systems.

Some of the real-time process algebras are simple extensions of well-known untimed process algebras,

- $TCSP$ [RR88a, Sch91, DS89] is a simple extension of CSP [Hoa85] with
  - a delay operator $t.P$ (it behaves as P after exactly t time units),
  - a weak timeout operator: behaves as P and then executes Q after d time units.
- $TeCCS$ [MT90], $TiCCS$ [Yi90] are extension s of CCS [Mil89] with
  - a time-lock 0 (for $TeCCS$),
  - a delay operator which we write $(t).P$ (for both): it behaves as P after exactly d time units,
  - unbounded idling: δP can act as P after any amount of time (for $TeCCS$),
  - a prefixing operator $a@vP$ which executes a and then behaves as P with the time variable v by any time ($TiCCS$).
- ACP_ρ [BB90] based on ACP [BK84] with
  - $\delta(d)$ a time-lock at time d,
  - time-stamped actions: absolute stamps like $a(d)$ performing action a at time d or relative ones like $a[d]$ ($d \in \mathbb{R}$) performing action a d time units after the previous action has been performed,
  - an integral operator $\int_{v \in V} P(v)$ which behaves as P with the time variable v replaced by any value of V.
- ATP [NRSV90] with
  - unbounded idling: $\lfloor P \rfloor^\omega$ may perform actions from P or idle as much as it wants,

  - a start-delay operator: $\lfloor P \rfloor^d(Q)$ behaves as P for at most d time units and then executes Q.

- TPL [HR91] with
  - a delay operator $(t).P$: it behaves as P after exactly t time units,
- $U-LOTOS$ [BL91] with
  - a delay operator $(t).P$: it behaves as P after exactly t time units,
  - an operator $asap$ (as soon as possible). It enforces the urgency of a set of actions in the whole execution of a process.

All of them are "deadlock-free" in the sense that no process can block the progress of time, except in ACP_ρ, $TeCCS$ and $U-LOTOS$ in which "time locks" are used to detect inconsistencies in specifications.

All of them have also the "action urgency" property. This means that some actions must be fired without delay. In $TCSP$, $TiCCS$ and TPL these are only the invisible actions.

$TCSP$, $TiCCS$ and $U-LOTOS$ are the only ones which satisfy the counter-intuitive "persistency" property, i.e. the fact that the progress of time cannot suppress the ability to perform an action.

Finally, only $TCSP$ has the "finite-variability" [BKP86] or "non-Zenoness" or "well-timed–ness" property. This prevents any Zeno process to be representable in that only finitely many actions can be performed in a finite time interval. It is realized at the expense of a complicated theory by enforcing a system delay between two actions of a sequential process.

We will propose a model for real-time systems based on a truly concurrent operational model in Chapter 11 which does not need complicated assumptions on time to describe plausible timed behaviours.

Summary We have described some of the models of concurrency that are used for giving semantics to languages and for the description of the basic concepts of concurrency theory. In particular, we have focused on the operational models for true concurrency, i.e. those models which have actions as basic bricks and which distinguish non-determinism from concurrency. In order to be as complete as possible, we have followed the classification of [WN94] around the notions interleaving/truly-concurrent, behaviour/model and linear/branching. We will follow in the next chapters their use of category theory for comparing models and constructing process algebras from the structure of the models. Other views on the notion of morphism can be put forward [Abr93a, Abr93b], this should be applied to the HDA model in some future work. We have also written some examples of the use of these models of concurrency for giving semantics to toy languages, like the process algebras CCS and CSP. We have ended this survey chapter by discussing some of the extensions of classical semantic models to deal with real-time systems. We will also propose an extension of the HDA model to real-time HDA in the last chapter of this thesis.

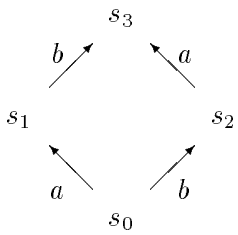
Chapter 2

An introduction to Higher Dimensional Automata

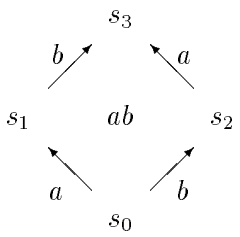
2.1 Introduction

Geometry has been suggested as a tool for modeling concurrency using higher-dimensio–nal objects to describe the concurrent execution of processes. This contrasts with earlier models based on interleaving of computation steps to capture all possible behaviours of a concurrent system. Such models must necessarily commit themselves to a specific choice of *atomic action* which makes them unable to distinguish between the execution of two truly concurrent actions and of two mutually exclusive actions as these are both modeled by their interleaving. This constrasts also with models of true concurrency for which the asynchronous executions are not "first-class" transitions. This was the case of asynchronous transition systems (see Chapter 1) for which we have an independence relation but no notion of "real" asynchronous execution.

In [Pra91b] and [vG91] Pratt and van Glabbeek advocate a model of concurrency based on geometry and in particular on the notion of a higher–dimensional automaton (HDA). Higher–dimensional automata are generalizations of the usual non–deterministic finite automata as described in e.g. [HU79]. The basic idea is to use the higher dimensions to represent the concurrent execution of processes. Thus for two processes, a and b, we model the mutually exclusive execution of a and b by the automaton



whereas their concurrent execution is modeled by including the two–dimensional surface delineated by the (one–dimensional) a– and b–transitions as a transition in the automaton. This is pictured as



A computation is a path in this higher–dimensional automaton.

We begin by giving a first combinatorial description of HDA under the name "semi-regular HDA". They are very practical since their definition is very simple. However they are not powerful enough for us to speak about some important geometric properties of the computations.

As a matter of fact, several properties of computational relevance are determined by the topology of the HDA. For example a HDA is deterministic if for any two paths in the automaton one can be transformed into the other in a continuous fashion, i.e. non–determinism arises from *holes* in the automaton that prevent the transformation of one path into another. Furthermore certain differences in the topologies of two HDA imply that a computation is possible in one HDA but not the other, i.e. information about the topology of HDA can be used to answer questions about, for instance, *bisimulation* (see Section 6.5) between the HDA. To be able to speak about all these properties, we have to introduce the notion of general HDA. We add up a few useful decorations, like labelling at the end of this chapter.

2.2 Basic definitions

Here we present two algebraic formalizations of the "geometric" transition systems we are interested in. These are inspired by many well-known mathematical techniques from algebraic topology and homological algebra so we present a short note first for mathematically oriented readers.

The first model, called semi-regular HDA has its direct inspiration in the simplicial techniques for describing geometric shapes. The name itself comes from the so-called "semi-simplicial complexes" [HW60] since we have only face operators and no degeneracy ones. Instead of using simplicial complexes, we use cubical ones (like in [Ser51]). Mathematicians will note that here we are really interested in the combinatorial complex itself and not just as a tool to compute invariants of some geometrical shape. This view will be postponed until Section 2.2.4 where the cubical complex generates a double complex. Logicians and category theoreticians will easily recognize that this combinatorial model is an instance of what is called a topos, or an "intuitionistic set theory".

The second one, called general HDA is the fully algebraic treatment of the combinatorial structure of the semi-regular HDA. This means that the combinatorics is made hidden by looking at some induced "weak" double complex structure, but some of the geometric characteristics of the HDA are made apparent through homology functors. The aim of the remaining chapters will then be to show why these notions are of importance in computer science.

2.2.1 Semi-regular HDA

We begin by presenting a very simple geometric model for true concurrency, based on ideas by Vaughan Pratt and Rob van Glabbeek [Pra91b, vG91] and formalized in different ways in [Gou93, GJ92].

Operational models for concurrency start with (ordinary) transition systems as we have seen in Chapter 1. This definition has already some geometry in it since we are all used to represent them as arrows (transitions) between states (points or small circles). But this does not provide us with a semantics stable by refinement [vGG89] nor does it distinguish non-determinism from truly concurrent (or asynchronous) execution.

As we have seen, a possible answer is to decorate the transition systems with some relation prescribing the independence of some actions (or transitions). This can be done in more than one way; just to mention a few: asynchronous transition systems [Bed88, Shi85], concurrent automata [Sta89] and transition systems with independence [WN94]. We comment on the former only and refer the reader to Chapter 1 for a detailed discussion of these operational models.

The decoration (the independence relation I) added to ordinary transition systems is enough to make the distinction between non-determinism and true concurrency. Suitable refinement operators can be defined as well on these structures.

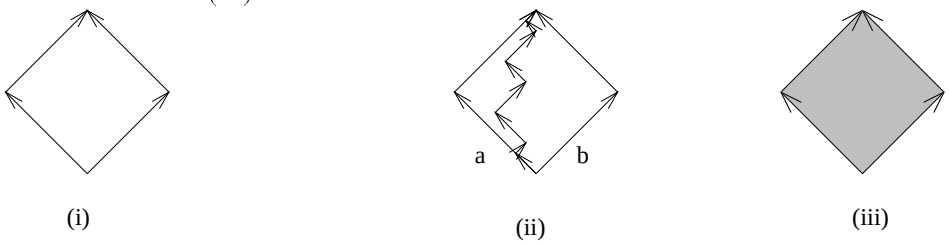
There is a slight problem though. The level of parallelism is not defined in a very precise manner. This is due to the fact that the independence relation is only a binary one. We can interpret "aIb and bIc and cIa" *once and for all* as either "a, b and c can be run asynchronously" (*maximal parallelism* assumption) or "no more than two among the three actions a, b and c can be run asynchronously" (*minimal parallelism* assumption). We insist on the "once and for all" in the last sentence, since changing the interpretation of the independence relation for different transition systems would amount to assume implicit (external to the model) conventions. We come back to these interpretation issues in Chapter 3.

Of course, a straightforward generalization would be to replace the binary relation I by an n-ary relation. This could be done (though we do not have any pointers in the literature) but we have in mind to be able to add some features to our model like real-time (Part V) and the generalization then seems too complicated.

This problem can be tackled if we get back to our geometric intuition. Things have been made overly unnatural by adding an object (the independence relation) which is not of the same nature as transitions and states. Just think of aIb as an abstraction of all possible asynchronous executions of a and b. As in [Pra91b], this can be pictured as the filled-in square at the right-hand side of Figure 2.1, distinguishing it in a striking manner with the interleaving at the left-hand side of the same figure. Notice that geometrically, the interior of the square consists of the union of all paths where executions of a and b overlap "in time" (middle picture of Figure 2.1). Time already makes its way into the model, though not quantified yet.

As a direct generalization, asynchronous execution of n transitions give rise

Figure 2.1: Non-determinism (i) versus overlap in time (ii) abstracted by a transition of dimension 2 (iii).



to hypercubes of dimension n, called n-transitions (ordinary transitions are 1-transitions, states are 0-transitions). Interestingly enough, all this has a very neat algebraic formulation.

We present the geometric shapes we are interested in as unions of points, segments, squares,..., hypercubes, i.e., as collections of n-transitions ($n \in \mathbb{N}$). We glue them together by means of boundary relations (see Figure 2.2), given by two boundary operators: d^0, the start boundary operator and d^1 the end boundary operator. They generalize the source and target functions for ordinary automata.

Consider first a segment,

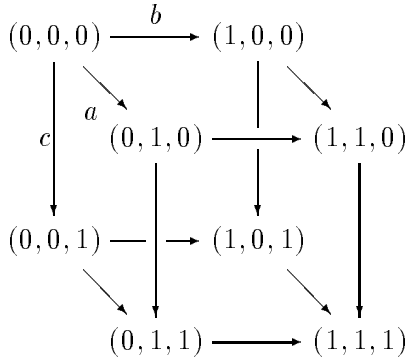
$$0 \xrightarrow{\quad I \quad} 1$$

The object of dimension one I has as source boundary $d^0(I) = 0$, and as target boundary $d^1(I) = 1$. What should we do for the square?

$$\begin{array}{ccc} (0,0) & \xrightarrow{a} & (0,1) \\ {\scriptstyle b}\downarrow & A & \downarrow{\scriptstyle b'} \\ (1,0) & \xrightarrow{a'} & (1,1) \end{array}$$

This corresponds to the asynchronous execution of actions a and b (a' and b' are copies of transitions of label a and b respectively). The object of dimension 2 "interior of the square" A should certainly have two source boundaries, up to the order on $\{a, b\}$, $d^0_0(A) = a$ and $d^0_1(A) = b$ since from state $(0,0)$ we can fire a and b. Similarly, it should have two target boundary operators $d^1_0(A) = a'$ and $d^1_1(A) = b'$ since from the parallel execution of a and b (represented by A) we can first end action a (giving "residue" b') or action b (giving "residue" a'). We will see this again when speaking about paths. Notice that with this ordering on vertices, we have, $d^0(d^0_1(A)) = (0,0) = d^0(d^0_0(A))$ and $d^1(d^0_1(A)) = (1,0) = d^0(d^1_0(A))$.

This generalizes easily to the cube,



The object of dimension 3, "interior of the cube" D, has three source boundaries, the three faces containing $(0,0,0)$, and three target boundaries, the three faces containing $(1,1,1)$.

Let A, B and C be the faces (respectively)

$$((0,0,0),(1,0,0),(0,0,1),(1,0,1))$$

$$((0,0,0),(0,1,0),(0,0,1),(0,1,1))$$

$$((0,0,0),(1,0,0),(0,1,0),(1,1,0))$$

Let A', B' and C' the faces parallel to A, B and C respectively.

Set $d^0_0(D) = A$, $d^0_1(D) = B$, $d^0_2(D) = C$ and $d^1_0(D) = A'$, $d^1_1(D) = B'$, $d^1_2(D) = C'$. Then $d^0_0(A) = b$, $d^0_1(A) = c$, $d^0_0(B) = a$, $d^0_1(A) = c$, $d^0_0(C) = a$, $d^0_1(C) = b$. We verify that,

$$d^0_i(d^0_j(D)) = d^0_{j-1}(d^0_i(D))$$

for all $i < j$.

This algebraic relation can be seen as corresponding to the cube axiom of concurrent transition systems (Section 1.1.4) describing the (strong) confluence of actions a, b and c. This is a bit more general here since the cube axiom corresponds to the view that we run every program on no more than two processors, identifying the strong confluence of any two actions among three with the parallel execution on three processors. Here the decomposition of the cube shows that the boundary of an asynchronous execution of three processes is the "interleaving" of all possible asynchronous executions on two processors.

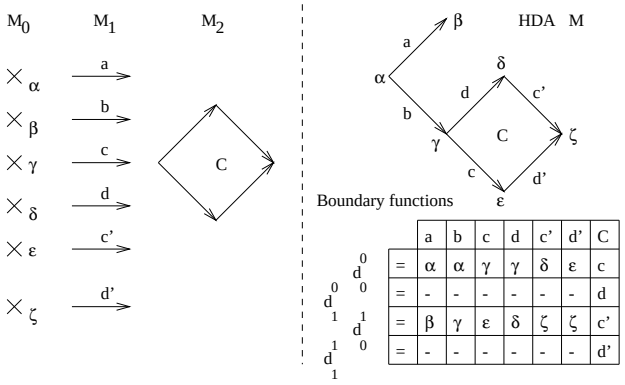
This can be generalized to higher levels of parallelism. We can show that for any hypercube of dimension n, we can choose an ordering on vertices, squares etc. such that the $2n$ boundary operators verify the commutation rules[1],

$$d^k_i \circ d^l_j = d^l_{j-1} \circ d^k_i$$

for $k = 0, 1$, $l = 0, 1$ and $i < j$.

Now we can **glue** these elementary shapes in order to get HDA. This is exemplified in Figure 2.2. We verify on the example the commutation rule between the source and target boundary operators d^0 and d^1 respectively.

Figure 2.2: Glueing of elementary shapes to get a semi-regular HDA.



		a	b	c	d	c'	d'	C
d^0_0	=	α	α	γ	γ	δ	ε	c
d^0_1	=	-	-	-	-	-	-	d
d^1_0	=	β	γ	ε	δ	ζ	ζ	c'
d^1_1	=	-	-	-	-	-	-	d'

We can then introduce these formally under the name of unlabeled semi-regular HDA.

Definition 16 *An unlabeled semi-regular HDA is a collection of sets M_n ($n \in \mathbb{N}$) together with functions*

$$M_n \underset{d^1_j}{\overset{d^0_i}{\rightrightarrows}} M_{n-1}$$

for all $n \in \mathbb{N}$ and $0 \leq i, j \leq n-1$, such that

$$d^k_i \circ d^l_j \quad = \quad d^l_{j-1} \circ d^k_i$$

$(i < j and k, l = 0, 1)$ and $\forall n, m \; n \neq m, \quad M_n \cap M_m = \emptyset$.

Elements x of M_n ($dim\ x = n$) are called n-transitions (or states if $n = 0$).

In order to be able to study "natural" constructions on HDA, we define a notion of **morphism** between them. As customary in recent work in concurrency [WN94], morphisms look like **simulations**. We set morphisms to be structure-preserving maps. In geometrical terms, morphisms preserve shapes, time (every transition is mapped onto a transition), and orientation.

Definition 17 *Let M and N be two semi-regular HDA, and f a family $f_n : M_n \to N_n$ of functions. f is a morphism of semi-regular HDA if and only if*

$$f_n \circ d^0_i = d^0_i \circ f_{n+1}$$

$$f_n \circ d^1_i = d^1_i \circ f_{n+1}$$

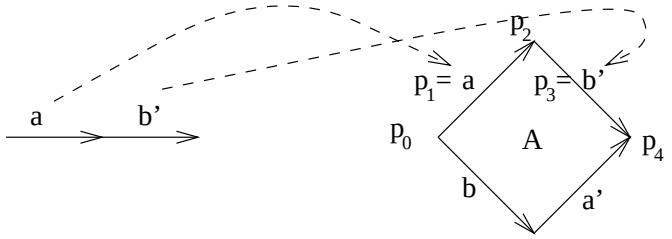
for all $n \in \mathbb{N}$ and $0 \leq i \leq n$.

This defines the category Υ_{sr} of semi-regular HDA.

We write Υ^n_{sr} for the full subcategory of Υ_{sr} consisting of semi-regular HDA whose elements are transitions of dimension less than or equal to n.

[1] Very much like the ones we have for simplicial complexes.

Figure 2.3: A path and its inclusion morphism in a semi-regular HDA.



There is a truncation functor $T_n : \Upsilon_{sr} \to \Upsilon^n_{sr}$ defined by, $T_n(M)_m = M_m$ if $m \leq n$ and $T_n(M)_m = \emptyset$ if $m > n$. Its effect can be interpreted as restricting to behaviour on n processors.

Now, traces of execution are described as sequences of states and transitions satisfying certain properties. A **path** is to be understood as a sequence of *allocation* (case (ii) below) of one action at a time on a new processor or *deallocation* (case (i) below) of one action at a time (i.e. its execution has ended on a given processor). An example of a path in an automaton M is given in Figure 2.3 together with its inclusion morphism into M (M simulates all of its paths).

Definition 18 *A path in a semi-regular HDA M is $p = (p_0, \ldots, p_n)$ such that p_0 and p_n are states and*

$$\forall k, 0 < k < n, \exists j, \begin{cases} p_{k-1} = d^0_j(p_k) & (i) \\ or, & \\ p_{k+1} = d^1_j(p_k) & (ii) \end{cases}$$

The definition of paths explains why the morphisms are (higher-dimensional) simulations. The commutation with the start boundary operator d^0 for example can be seen as asserting: "whenever M fires a new action, N fires a similar one".

For $k = 1$ we are always in case (i) and for $k = n - 1$ we are always in case (ii). n is the length of p. If p does not verify any particular condition on the states p_0 and p_n then p is called partial. If p_0 is an initial state, i.e. a state such that there is no 1-transition t with $d^1_0(t) = p_0$, then p is a semi-partial path. If p_n is a final state as well, i.e. a state for which there is no 1-transition t with $d^0_0(t) = p_n$, then p is a total path.

Categorical and Combinatorial Properties

We first note that semi-regular HDA form a higher-order type theory (see [LS86]).

Proposition 1 *Υ_{sr} is an elementary topos. Moreover it is complete and cocomplete.*

PROOF. There is actually a better way to formulate the definitions of semi-regular HDA, in order to study their algebraic properties.

Let $\Box$ be the free category[2] whose objects are $[n]$, where $n \in \mathbb{N}$, and whose morphisms are generated by,

$$[n] \underset{\delta_j^1}{\overset{\delta_i^0}{\rightrightarrows}} [n-1]$$

for all $n \in \mathbb{N}^*$ and $0 \leq i, j \leq n-1$, such that $\delta_i^k \delta_j^l = \delta_{j-1}^l \delta_i^k \quad (i < j)$. Now, the category $\Box\mathbf{Set}$ of functors from $\Box$ to **Set** (morphisms are natural transformations) is isomorphic to Υ_{sr}. Therefore ([LS86] and [MM92]) it is an elementary topos. It is complete and co-complete because **Set** is complete and co-complete. $\Box$

This formulation of the definition of Υ_{sr} enables us to describe combinatorially the shapes we are dealing with. Let $D_{[n]}$ be the semi-regular HDA $Hom_{\Box}([n], \cdot)$ (where $Hom_{\Box}$ is the Hom functor in the category $\Box$).

(*) Definition 1 *A singular n-cube of a HDA M is a morphism $\sigma : D_{[n]} \to M$.*

(*) Lemma 1 *The set of singular n-cubes of a semi-regular HDA M is in one-to-one correspondence with M_n. The unique singular n-cube corresponding to a n-cube $x \in M_n$ is denoted by $\sigma_x : D_{[n]} \to M$. It is the unique singular n-cube σ such that $\sigma(Id_{[n]}) = x$.*

PROOF. By Yoneda's lemma. Recall that Υ_{sr} is isomorphic to the category of functors from $\Box$ to **Set**. Then the $D_{[n]}$ are the representable functors and $Nat(D_{[n]}, M) \cong M([n])$, where M is a functor from $\Box$ to **Set**. This translates to $\Upsilon_{sr}(D_{[n]}, M) \cong M_n$. $\Box$

(*) Proposition 1 *Let M be a semi-regular HDA. The following diagram is co-cartesian (for $n \in \mathbb{N}$),*

$$\begin{array}{ccc} \coprod_{x \in M_{n+1}} \dot{D}_{[n+1]} & \xrightarrow{\coprod_{x \in M_{n+1}} \dot{\sigma}_x} & T_n(M) \\ \subseteq \downarrow & & \subseteq \downarrow \\ \coprod_{x \in M_{n+1}} D_{[n+1]} & \xrightarrow{\coprod_{x \in M_{n+1}} \sigma_x} & T_{n+1}(M) \end{array}$$

where $\dot{D}_{[n+1]} = T_n(D_{[n+1]})$ and $\dot{\sigma}_x = \sigma_{x|\dot{D}_{[n+1]}}$.

[2]Such a category exists by general theorems. It is actually isomorphic to a poset category which we will not describe here.

PROOF. It suffices to prove that the diagram below (in the category of sets) is cocartesian for all $p \leq n+1$,

$$\begin{array}{ccc} \coprod_{x \in M_{n+1}} (\dot{D}_{[n+1]})_p & \xrightarrow{\coprod_{x \in M_{n+1}} (\dot{\sigma}_x)_p} & (T_n(M))_p \\ \subseteq \downarrow & & \subseteq \downarrow \\ \coprod_{x \in M_{n+1}} (D_{[n+1]})_p & \xrightarrow{\coprod_{x \in M_{n+1}} (\sigma_x)_p} & (T_{n+1}(M))_p \end{array}$$

since colimits (hence pushouts) are taken pointwise in a functor category into **Set**.

For all $p < n+1$, the inclusions are in fact bijections, and the diagram is then obviously cocartesian.

For $p = n+1$, the complement of $\coprod_{x \in M_{n+1}} (\dot{D}_{[n+1]})_p$ in $\coprod_{x \in M_{n+1}} (D_{[n+1]})_p$ is the set of copies of cubes $Id_{[n+1]}$, one for each cube of M_{n+1}. This means that the map $\coprod_{x \in M_{n+1}} (\sigma_x)_p$ induces a bijection from the complement of $\coprod_{x \in M_{n+1}} (\dot{D}_{[n+1]})_p$ onto the complement of $(T_n(M))_p$. This implies that the diagram is cocartesian for $p = n+1$ as well. □

This lemma states that the truncation of dimension $n+1$ of a semi-regular HDA M is obtained from the truncation of dimension n of M by attaching some standard $(n+1)$-cubes $D_{[n+1]}$ along the boundary $\dot{D}_{[n+1]}$ of $n+1$ dimensional holes. This is the basic property of combinatorial cell complexes [LW69] and this will be used when passing from discrete to continuous geometry in Part V (real-time systems).

Computer-scientifically, the previous proposition states that the shapes described by the class of semi-regular HDA is a sensible one since we go from a skeleton of dimension n to a skeleton of dimension $n+1$ by adding some independence relations, or dually, by cancelling some mutual exclusions.

We proceed by decribing the main categorical combinators in two ways. We present their definition in terms of sets of transitions and boundary operators and in a SOS-like metalanguage we now define.

In this meta-language, we wish to enumerate the transitions of higher-dimensional transition systems in a format similar to the usual SOS one.

Semi-regular HDA can be seen as the union of its sub-HDA generated by a single n-transition. From that point of view, enumerating the n-transitions can be seen as enumerating these sub-HDA. We choose[3] to abstract these by the pair of their initial and final states and the generating n-transition, that is, for

[3] There are other choices which in particular describe higher-dimensional traces in a more accurate way. We prefer to use this abstraction here because it is simpler and relates to Hoare-like formalisms for proving partial correctness of programs.

t a n-transition of M,

$$(s \xrightarrow{t} s') \quad = \{d^{\alpha_1}_{\beta_1} \ldots d^{\alpha_k}_{\beta_k}(t) / k \leq n, \alpha_i = 0,1\} \subseteq M$$

where $s = d^0_0 d^0_1 \ldots d^0_{n-1}(t)$ and $s' = d^1_0 d^1_1 \ldots d^1_{n-1}(t)$ are the initial and final states of t in M respectively. Then, we define an entailment relation $\models$ to relate M to its sub-HDA, and we write,

$$M \models \quad s \xrightarrow{a} s' \qquad \Leftrightarrow \qquad (s \xrightarrow{a} s') \quad \subseteq M$$

Notice that for all M and states s of M we may write,

$$M \models s \xrightarrow{s} s$$

Conversely, given a set of SOS-like rules of the form

$$\frac{P_1 \models u_1 \xrightarrow{a_1} v_1 \ \ldots \ P_n \models u_n \xrightarrow{a_n} v_n}{P \models u \xrightarrow{a} v}$$

we can get back to a semi-regular HDA: this is called the *interpretation* of the SOS rules. This is done by considering the set of rules as a positive inductive definition [CC92b].

We say that a set of SOS rules is *adequate* with respect to a categorical combinator if and only if its interpretation is isomorphic to the application of the categorical combinator.

This being settled, we can describe the categorical combinators.

All limits and colimits are computed "pointwise" (see [ML71]) in a functor category. Translating this back from the functor category to the category of semi-regular HDA we obtain,

- the cartesian product of two semi-regular HDA F, G is the semi-regular HDA $F \times G$ with

  $$(F \times G)_n = F_n \times G_n$$

  and

  $$d^\epsilon_k[F \times G] = d^\epsilon_k[F] \times d^\epsilon_k[G]$$

  In SOS form, only one rule is required,

  $$\frac{Q \models u \xrightarrow{t} v \qquad Q' \models u' \xrightarrow{t'} v' \qquad dim\ t = dim\ t'}{Q \times Q' \models (u,u') \xrightarrow{(t,t')} (v,v')}$$

  It can be interpreted as the synchronized product of Q and Q', where their respective transitions are forced to be executed in a synchronous manner. It is adequate.

- the coproduct of two semi-regular HDA F, G is

$$(F \coprod G)_n = F_n \cup G_n$$

$$d^\epsilon_k[F \coprod G](x) = \begin{cases} d^\epsilon_i[F](x) & \text{if } x \in F_n \\ d^\epsilon_k[G](x) & \text{if } x \in G_n \end{cases}$$

In SOS form, we need two rules,

$$\frac{Q \models u \xrightarrow{t} v}{Q \coprod Q' \models u \xrightarrow{t} v}$$

$$\frac{Q' \models u' \xrightarrow{t'} v'}{Q \coprod Q' \models u' \xrightarrow{t'} v'}$$

These show that the coproduct acts like the non-deterministic choice + of CCS (Section 5.4).

- In the isomorphic functor category, the Hom-functor right-adjoint to the cartesian product is given by Yoneda's lemma. If we write $D_{[n]}$ for the representable functor (where $Hom_\square$ is the Hom-functor in the category $\square$) $D_{[n]} = Hom_\square([n], \cdot)$, the right adjoint $\Rightarrow$ is,

$$G \Rightarrow H([n]) = Nat(D_{[n]} \times G, H)$$

and for $f \in G \Rightarrow H([n])$,

$$\begin{aligned} G \Rightarrow H(\delta^\epsilon_k)(f) : D_{[n-1]} \times G &\longrightarrow H \\ (u, v) &\longrightarrow f(u \circ \delta^\epsilon_k, v) \end{aligned}$$

(Nat denotes here the set of natural transformations). This translates to,

$$(G \Rightarrow H)_n = \{f : D_{[n]} \times G \to H / f \text{ morphism}\}$$

$$\begin{aligned} d^\epsilon_k[G \Rightarrow H](f) : D_{[n-1]} \times G &\longrightarrow H \\ (u, v) &\longrightarrow f(u \circ \delta^\epsilon_k, v) \end{aligned}$$

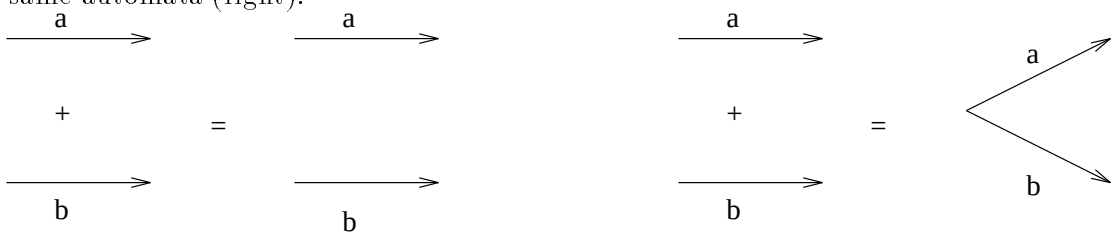
In SOS form,

$$\frac{G \models u \xrightarrow{t} v \qquad G \Rightarrow H \models u' \xrightarrow{t'} v' \quad dim\ t = dim\ t'}{H \models u'(u) \xrightarrow{t'(t)} v'(v)}$$

Note that it looks like the elimination rule for $\Rightarrow$ in intuitionistic logic. This is due to general constructions of categorical logic. It is not fully adequate since we would need an introduction rule (but this in turn needs judgments including variables).

Before carrying on deeper into the categorical structure of semi-regular automata, we give a few pictures and examples.

Figure 2.4: Coproduct of two automata (left) and amalgamated sum of the same automata (right).



Example 2

- *In Figure 2.4, we have drawn the coproduct of two HDA and an amalgamated sum, where the initial points have been identified. This amalgamated sum (pushout) represents an internal choice*[4] *as in TS (Section 1.3.1) whereas the coproduct (or direct sum) is an external choice. It can be seen as a coproduct in the category of semi-regular HDA equipped with a basepoint, and morphisms preserving them. This category is interesting since it is mimicking the one of standard automata, where the basepoint is the initial state.*

- *In Figure 2.5, we have pictured a cartesian product of two semi-regular HDA as well as a fibered product (pullback). The fibered product here only synchronizes actions that are mapped by l onto the same transition. This will be one of the basic remarks for defining the category of labeled HDA. l corresponds to a labelling function and the fibered product is a synchronization à la CSP (Section 1.3.2).*

- *Finally, we have drawn a very simple example of a "function space" HDA (Figure 2.6). It can be seen to represent at least* synchronous function calls. *Let $f : Var \times Prog \to Res$ be a morphism between semi-regular HDA Var (for values of "variables"), $Prog$ (for "program" traces) and Res (for "result"). f computes the value of the result when the program is executed with initial values represented by Var. Now this morphism is "equivalent" to $Proc = curry(f) : Var \to (Prog \Rightarrow Res)$ which gives the different "specializations" of f when applied to particular values. On the other side of the adjunction, we have a synchronous evaluation morphism $eval : (Var, Proc) \to Res$.*

The category Υ_{sr} also has features from linear logic [Gir87].

Proposition 2 *Υ_{sr} is a monoidal closed category.*

As a sketch of proof we construct a tensor product and its right-adjoint. Define a tensor product $F \otimes G$ of two semi-regular HDA F and G (to represent the parallel composition with no interference as in [Gou93]) to be

$$(F \otimes G)_n = \bigcup_{i+j=n} F_i \times G_j$$

[4] There is a parameter "hidden" in the initial state which makes the choice between the two branches.

Figure 2.5: A cartesian product (left) and a fibered product (right).

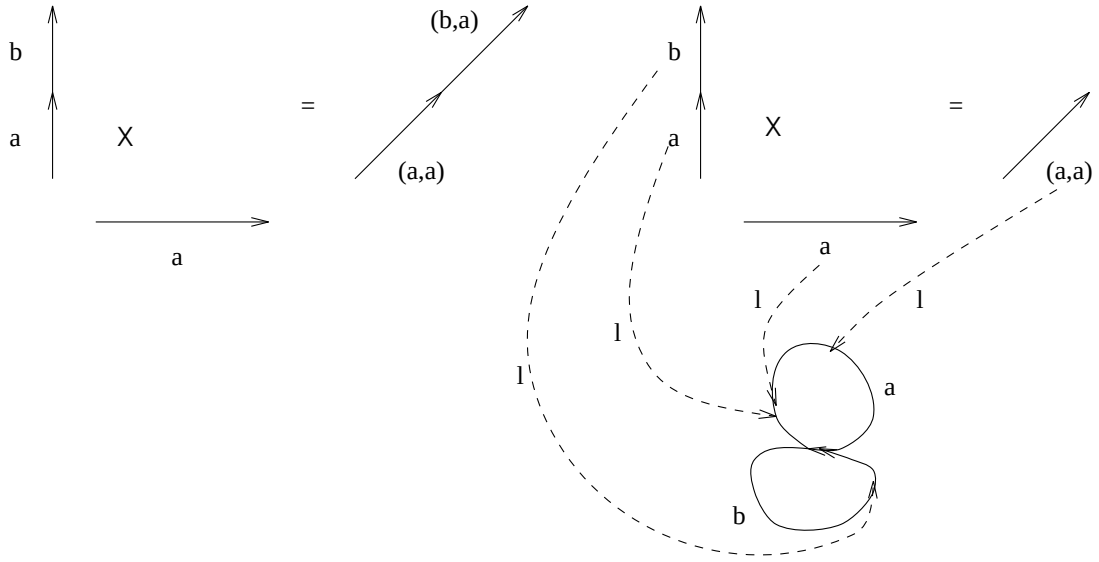
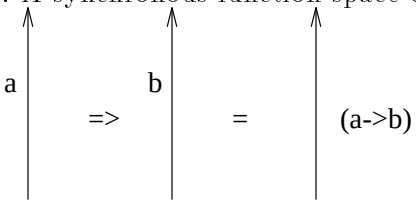


Figure 2.6: A synchronous function space of HDA.

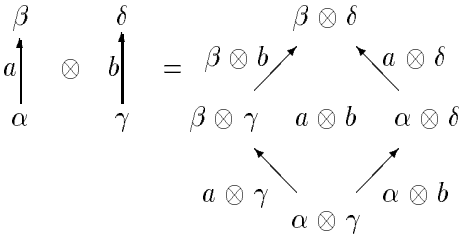


and, for $x \in F_i, y \in G_j$,

$$d^\epsilon_k[F \otimes G](x,y) = (d^\epsilon_k[F](x), y) \quad \text{if } k \leq i-1$$

$$d^\epsilon_k[F \otimes G](x,y) = (x, d^\epsilon_{k-i}[G](y)) \quad \text{if } k > i-1$$

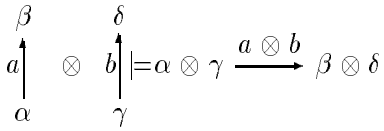
For example,



The corresponding (adequate) rule for the tensor product is

$$\frac{Q \models u \xrightarrow{\ t\ } v \qquad Q' \models u' \xrightarrow{\ t'\ } v'}{Q \otimes Q' \models u \otimes u' \xrightarrow{\ t \otimes t'\ } v \otimes v'}$$

Notice that the tensor product creates transitions of higher-dimensions (it is different from the "synchronous" product) that precisely express the asynchrony in their execution. In the example above we have



The tensor product has a right-adjoint, because it commutes with colimits (by proposition II.1.3 of [GZ67]), which is given again by Yoneda's lemma. We note it by $\multimap$, and it is defined by,

$$(G \multimap H)_n = \{f : D_{[n]} \otimes G \to H / f \text{ is a morphism}\}$$

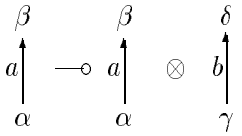
$$\text{and for } f \in (G \multimap H)_n,\ d^\epsilon_k[G \multimap H](f) : D_{[n-1]} \otimes G \longrightarrow H$$

$$(u,v) \longrightarrow f(u \circ \delta^\epsilon_k, v)$$

Finally, its SOS rule is

$$\frac{G \models u \xrightarrow{\ t\ } v \qquad G \multimap H \models u' \xrightarrow{\ t'\ } v'}{H \models u'(u) \xrightarrow{\ t'(t)\ } v'(v)}$$

In $G \multimap H$ we have functions which fork new actions (dynamically). In



there is for instance a 1-transition "fork the b action" (using the λ notation for functions)

$$\lambda x.x \otimes \gamma \xrightarrow{\ \lambda x.x \otimes b\ } \lambda x.x \otimes \delta$$

More generally, using the same kind of arguments that we had on the synchronous function space, we can call $G \multimap H$ the *asynchronous function space*. It contains in particular analogues to Remote Procedure Calls in a quite abstracted way, since its elements are functions which evaluate their arguments in parallel with their own execution. This is a quite powerful kind of mobility of processes even if it is difficult to see the exact relationship with the mobility attained in π-calculus (Section 1.3.3) or CHOCS [Tho89].

We will not need the exponentials of linear logic [Gir87] in the sequel. Nevertheless, we give one possible construction here for completeness of this chapter. For A a semi-regular HDA, let $!A$ be defined by $(!A)_n = \emptyset$ for all $n \neq 0$ and $(!A)_0 = A_0$. It defines an endofunctor on Υ_{sr}. Let now $d :! \rightarrow !\circ !$ and $e :! \rightarrow Id$ be the natural transformations defined by: $d(A)$ is the identity morphism on $!A$, $e(A)$ is the inclusion morphism of A_0 into A. $(!, d, e)$ is a comonad. Now, standard results (see [AL91b] for instance) give the interpretation of all non-commutative intuitionistic linear logic in Υ_{sr}. The intuitionistic logic, retract of this linear logic in the comonad $(!, d, e)$ is the classical logic of the powerset of states (on which Hoare logics are based).

(†) Geometric realization

We are trying now to give an explicit geometric representation of semi-regular HDA. This is done in the same style as the geometric realization functor between simplicial sets and CW-complexes (see for instance [GZ67] or [May67]).

This will prove useful, not only for formalizing the way we picture HDA but also in the future for giving hints about how to go from discrete time ($\mathbb{N}$ or $\mathbb{Z}$) to continuous time ($\mathbb{R}$).

Let $\square_n$ be the standard cube in $\mathbb{R}^{n+1}$ $(n \geq -1)$,

$$\square_n = \{(t_0, \ldots, t_n) / \forall i, 0 \leq t_i \leq 1\}$$

$$\square_{-1} = \{0\}$$

and let δ_i^k, $0 \leq i \leq n$, be the continuous functions $(n \geq 0)$,

$$\begin{array}{ccc} \square_n & \xleftarrow{\delta_i^0} & \square_{n-1} \\ \uparrow{\scriptstyle \delta_i^1} & & \\ \square_{n-1} & & \end{array}$$

defined by,

$$\delta_i^k(t_0, \ldots, t_{n-1}) \quad = \quad (t_0, \ldots, t_{i-1}, k, t_i, \ldots, t_{n-1})$$

And, for $n = 0$,

$$\begin{aligned} \delta_0^0(0) &= (0) \\ \delta_0^1(0) &= (1) \end{aligned}$$

Then,

Lemma 1

$$\delta_i^k \delta_j^l = \delta_{j+1}^l \delta_i^k \qquad (i \leq j)$$

PROOF. Let $i \leq j$, and $(t_0, \ldots, t_n) \in \square_n$. Then,

$$\begin{aligned} \delta_i^k(\delta_j^l(t_0, \ldots, t_n)) &= \delta_i^k(t_0, \ldots, t_{j-1}, l, t_j, \ldots, t_n) \\ &= (t_0, \ldots, t_{i-1}, k, \ldots, l, t_j, \ldots, t_n) \\ &= \delta_{j+1}^l(\delta_i^k(t_0, \ldots, t_n)) \end{aligned}$$

□

We notice that δ^k verify the dual equations that d^k verify in all semi-regular HDA.

Consider now, for a semi-regular HDA M (on **Set**), the set $\mathcal{R}(M) = \coprod_{n, x \in M_n} (x, \square_n)$. Each $(x, \square_n)$ inherits a topology given by the standard one on $\mathbf{R}^{n+1}$, thus $\mathcal{R}(M)$ is a topological space with the disjoint sum topology.
Let $\equiv$ be the equivalence relation induced by the identities:

$$\forall k, i, n, \forall x \in M_{n+1}, \forall t \in \square_n, n \geq 0, (d_i^k(x), t) \equiv (x, \delta_i^k(t))$$

Let $\mid M \mid = \mathcal{R}(M)/\equiv$. It has a structure of topological space induced by $\mathcal{R}(M)$. $\mid M \mid$ is called the *geometric realization* of M.

(*) Lemma 2 *Let $f : X \longrightarrow Y$ be a morphism between the two semi-regular HDA X and Y. Then f induces a continuous map $\mid f \mid$ from $\mid X \mid$ to $\mid Y \mid$.*

(*) PROOF. Define $\mathcal{R}(f) : \mathcal{R}(X) \longrightarrow \mathcal{R}(Y)$ by: $\mathcal{R}(f)((x,t)) = (f(x), t)$. It is obviously a continuous map.

Suppose $(x,t) \equiv (y,s)$. Then there exists $(y_1, s_1), \ldots, (y_u, s_u)$ such that $(y_1, s_1) = (x,t)$, $(y_u, s_u) = (y,s)$ and $\forall g$, $\exists k, j$, $d_j^k(y_g) = y_h$ and $s_g = \delta_j^k(s_h)$ with $h = g+1$ or $h+1 = g$.

We show by induction on u that $\mathcal{R}(f)((x,t)) \equiv \mathcal{R}(f)((y,s))$, thus inducing a map from $\mid X \mid$ to $\mid Y \mid$. We just have to show that $\mathcal{R}(f)((x,t)) \equiv \mathcal{R}(f)((y_2, s_2))$, the result will be proven using the induction hypothesis.

Suppose $\exists k, j$, $d_j^k(x) = y_2$ and $t = \delta_j^k(s_2)$. But $d_j^k(f(x)) = f(d_j^k(x))$. Thus, $d_j^k(f(x)) = f(y_2)$ and t=$\delta_j^k(s_2)$, which proves the result. □

(*) Proposition 2 $\mid \cdot \mid$ *is a functor from Υ_{sr} to* **Top**, *the category of topological spaces with continuous maps.*

(*) PROOF. By Lemma 2 we see that we just need to prove that, for any two semi-regular morphisms f and g, $\mid f \circ g \mid = \mid f \mid \circ \mid g \mid$. This is straightforward. □

We can define an analogue of the singular complex functor (called here "cubicalation"), which is well-known to be a right-adjoint to the geometric realization functor between simplicial sets and CW-complexes.

Definition and lemma 1 *For $X \in$ **Top**, let $\mathcal{S}(X)$ be the semi-regular HDA defined as follows:*

- *$\mathcal{S}(X)_n$ is the set of singular cubes, i.e. the maps f: $\square_n \to X$,*
- *the operators d_i^k are defined by the following equations, for $f \in \mathcal{S}(X)_n$,*

$$d_i^k(f) \quad = \quad f \circ \delta_i^k$$

PROOF. The result is a direct consequence of Lemma 1. □

Proposition 3 *$\mathcal{S}$ induces a functor from **Top** to Υ_{sr}.*

PROOF. We first have to define the action of $\mathcal{S}$ on morphisms in **Top**. Let f be a morphism from X to Y in **Top**. Define $\mathcal{S}(f)$ on elements of $\mathcal{S}(X)_n$, i: $\square_n \to X$, to be $\mathcal{S}(f)(i) = f \circ i : \square_n \to Y$. Therefore the image of an element of $\mathcal{S}(X)_n$ by $\mathcal{S}(f)$ is an element of $\mathcal{S}(Y)_n$. We now have to verify that $\mathcal{S}(f)$ commutes with d_j^k to show that $\mathcal{S}(f)$ is a semi-regular morphism:

$$\begin{aligned} d_j^k(\mathcal{S}(f)(i)) &= d_j^k(f \circ i) \\ &= f \circ i \circ \delta_j^k \\ &= \mathcal{S}(f)(d_j^k(i)) \end{aligned}$$

Then, for any two morphisms f and g, we have for all i element of $\mathcal{S}(X)_n$, $\mathcal{S}(g \circ f)(i) = g \circ f \circ i = \mathcal{S}(g) \circ \mathcal{S}(f)(i)$. $\mathcal{S}$ is a covariant functor. □

We call *cubicalation* of an object X of **Top** any sub-semi-regular HDA M of $\mathcal{S}(X)$ with $\mid M \mid = \mathcal{S}(X)$. Any such sub-HDA corresponds to a choice of a time-flow.

(*) Theorem 1 *$\mathcal{S}$ is right-adjoint to $|.|$.*

(*) PROOF. We prove that there exist two natural transformations

$$\eta : Id \to \mathcal{S}(\mid . \mid)$$

$$\epsilon : \mid \mathcal{S} \mid \to Id$$

(respectively the unit and counit of the adjunction) such that

$$\mathcal{S} \xrightarrow{\eta\mathcal{S}} \mathcal{S}(|\ \mathcal{S}\ |) \xrightarrow{\mathcal{S}\epsilon} \mathcal{S}$$

$$|\ \cdot\ | \xrightarrow{|\ \cdot\ |\ \eta} |\ \mathcal{S}(|\ \cdot\ |)\ | \xrightarrow{\epsilon\ |\ \cdot\ |} |\ \cdot\ |$$

are the identity.

We can first show that:

$$(A): \quad M \hookrightarrow \mathcal{S}(|\ M\ |)$$

$$(B): \quad |\ \mathcal{S}(X)\ | \hookrightarrow X$$

in a natural manner for all M semi-regular HDA and X object of **Top**. We begin by (A). For all n, we have the identity arrows on $\square_n$ which induce the isomorphisms: for all x, $Id : \square_n \to (x, \square_n)$. These in turn induce injective morphisms $f_x : \square_n \to |\ M\ |$, because M is an amalgamated sum of the $(x, \square_n)$. The $(f_x)_x$ form a subset N of $\mathcal{S}(|\ M\ |)$. It is an easy exercise to show that N is closed under the action of the d_i^k. Thus N is a sub-semi-regular HDA of $\mathcal{S}(|\ M\ |)$. The naturality of the inclusion arrow $M \hookrightarrow \mathcal{S}(|\ M\ |)$ is most obvious. This defines what is to be the unit of the adjunction.

Now, we come to (B). Elements of $\mathcal{S}(X)_n$ are $f : \square_n \to X$. Now, $|\ \mathcal{S}(X)\ |$ is an amalgamated sum of $(x, \square_n)$, $x \in \mathcal{S}(X)_n$. The x induce on $\coprod_x (x, \square_n)$ and then on $|\ \mathcal{S}(X)\ |$ an injective morphism in **Top**. It is an easy exercice to show that these arrows are natural in X. This defines what is to be the counit of the adjunction.

Then, we have to verify that two compositions of natural transformations are the identity. This is easy verification. □

This implies that $|\ \cdot\ |$ commutes with all colimits. In particular ($+$ is the amalgamated sum):

$$|\ M + N\ | = |\ M\ | \cup |\ N\ |$$

2.2.2 (†) Partial HDA

Up to now, all transitions were required to terminate. Nothing in semi-regular automata represents deadlocking behaviour. Partial HDA are defined for this purpose: they are semi-regular HDA with "missing" boundaries, meaning that some transitions may never terminate if evaluated (deadlocking behaviour).

Partiality is very much used in all areas of semantics, and many studies have been published on the properties that some categories of partial maps have or do not have. To mention but a few, Plotkin [Plo85], Moggi [Mog86], Curien (partial categories), Carboni (bicategories of partial maps, [Car86]) and Robinson, Rosolini ([RR88b], categories of partial maps and p-categories) have all studied different versions of categories of partial maps. It is the case in all these very general and quite constrained definitions that not all categorical properties are preserved when going from a category A to a category PA of partial maps on A constructed out of A. Here we use a very pragmatic approach. The formal definitions are as follows,

Definition 19 *A partial HDA (or PHDA in short) is a collection of sets M_n ($n \in \mathbb{N}$) together with partial functions*

$$M_n \underset{d_j^1}{\overset{d_i^0}{\rightrightarrows}} M_{n-1}$$

for all $n \in \mathbb{N}$ and $0 \leq i, j \leq n-1$, such that

$$d_i^k d_j^l \quad = \quad d_{j-1}^l d_i^k \qquad (i < j, k, l = 0, 1)$$

whenever both *sides of the equality are well defined*[5] *and*

$$\forall n, m,\ n \neq m, \quad M_n \cap M_m = \emptyset$$

We write $d_i^k(x) = \perp$ when d_i^k is undefined in x. In this case, we say that x has no boundary d_i^k.

Then we define the category of partial HDA $P\Upsilon$ by giving a notion of morphism,

Definition 20 *Let M and N be two partial HDA, and f a family of partial functions $(f_n)_n : M_n \to N_n$. f is a morphism of partial HDA if and only if*

$$f_n \circ d_i^0 = d_i^0 \circ f_{n+1}$$

$$f_n \circ d_i^1 = d_i^1 \circ f_{n+1}$$

for all $n \in \mathbb{N}$, whenever both sides of the equalities are defined[6].

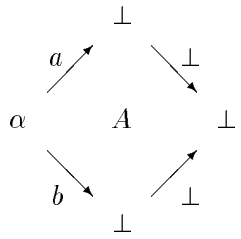
$f(x) = \perp$ corresponds to mapping a n-transition to an idle transition.

In the following example, we label missing boundaries by $\perp$.

Example 3

- $\alpha \xrightarrow{a} \perp$

  a is an action that deadlocks on one processor,

- 

  A deadlocks two processors.

[5] M is called *closed partial* if moreover when one side is undefined, the other is undefined as well. They form a full subcategory of partial HDA isomorphic to the category of functors from $\square$ to $\tilde{\mathbf{Set}}$, the category of sets with partial maps. I owe Régis Cridlig the idea that non-closed partial HDA are necessary for giving semantics to some languages (see [Cri95]) and that partial HDA in general should be studied in their own right.

[6] For closed partial HDA we restrict to morphisms for which both sides of the equality should be undefined at the same time. This gives us the subcategory $CP\Upsilon$.

The study of their categorical properties is made easy by what we know on semi-regular automata.

Definition and lemma 2 *The coproduct of M, N partial HDA is the partial HDA*

$$P_n = M_n \cup N_n$$

$$d_i^k[P] = d_i^k[M] \coprod d_i^k[N]$$

PROOF. We have two canonical morphisms $in_l : M \to P$ and $in_r : N \to P$ defined in a straightforward manner. We verify that if we have two morphisms $f : M \to Q$ and $g : N \to Q$ then there exists a unique $h : P \to Q$ such that $h \circ in_l = f$ and $h \circ in_r = g$. There is no other way than setting $h(x) = f(x)$ if $x \in M$, $h(x) = g(x)$ if $x \in N$. h is a morphism. □

Definition and lemma 3 *The cartesian product of two partial HDA M and N is the partial HDA $P = M \times N$,*

$$P_n = M_n \times N_n$$

$$d_i^k(x,y) = \begin{cases} (d_i^k(x), d_i^k(y)) & \text{if both boundaries are defined} \\ \perp & \text{otherwise} \end{cases}$$

PROOF. Let $p_1 : P \to M$ and $p_2 : P \to N$ be the two projections. They are morphisms of partial HDA since for instance $p_1(d_i^k(x,y)) = p_1(d_i^k(x), d_i^k(y)) = d_i^k(p_1(x,y))$ if both boundaries are defined and if not $p_1(d_i^k(x,y)) = \perp = d_i^k(p_1(x,y))$.

Now if $f : Q \to M$ and $g : Q \to N$ are two morphisms then $h : Q \to P$ defined by $h(x,y) = (f(x), g(y))$ is a morphism. □

In the subcategory of closed partial HDA cartesian products may not exist.

Definition 21 *For M and N two partial HDA define $P = M \otimes N$ by,*

$$P_n = \bigcup_{i+k=n} M_i \times N_k$$

and for $(x,y) \in M_i \times N_k$,

$$d_v^u(x,y) = \begin{cases} (d_v^u(x), y) & \text{if } d_v^u(x) \text{ defined and } v < i \\ (x, d_{v-i}^u(y)) & \text{if } d_{v-i}^u(y) \text{ defined and } v \geq i \\ \perp & \text{otherwise} \end{cases}$$

We define $P = M \Rightarrow N$ as,

$$P_n = \{f : D_{[n]} \times M \to N / f \text{ is a morphism}\}$$

$$\begin{aligned} d^\epsilon_k[M \Rightarrow N](f) : & D_{[n-1]} \times M \longrightarrow N \\ & (u, v) \longrightarrow f(u \circ \delta^\epsilon_k, v) \end{aligned}$$

Similarly, we define $P = M \multimap N$ as,

$$P_n = \{f : D_{[n]} \otimes M \to N / f \text{ is a morphism}\}$$

$$\begin{aligned} d^\epsilon_k[M \Rightarrow N](f) : & D_{[n-1]} \otimes M \longrightarrow N \\ & (u, v) \longrightarrow f(u \circ \delta^\epsilon_k, v) \end{aligned}$$

Lemma 2 *$\Rightarrow$ (resp. $\multimap$) is right-adjoint to $\times$ (resp. $\otimes$) in $P\Upsilon_{sr}$.*

Proof. Let $f \in P\Upsilon(M \times N, P)$. Let g be the function defined on M with range the set of partial functions from N to P by $g(m)(n) = f(m, n)$ $(m \in M, n \in N)$. We prove that g actually takes values in $N \Rightarrow P$.

Let $m \in M$, suppose $m \in M_i$. Let $\downarrow m = \{d^{\epsilon_1}_{i_1} \ldots d^{\epsilon_k}_{i_k}(m) / 0 \leq k \leq i\}$. Then there is a unique morphism $\alpha_m : D_{[i]} \to \downarrow m$ with $\alpha_m(Id) = m$ similarly to the total case. It is also called the singular cube associated with the cube m. Now, $g(m)(n) = f \circ (\alpha_m \times Id)(Id, n)$. By definition $f \circ (\alpha_m \times Id) \in N \Rightarrow P$, and $n \to (Id, n)$ is an isomorphism. Therefore $g(m)$ takes values in $N \Rightarrow P$.

Now we have to prove that $m \to g(m)$ is a morphism of partial HDA, i.e. that $g(d^\epsilon_k(m)) = d^\epsilon_k(g(m))$ whenever both sides are well defined. $g(d^\epsilon_k(m)) = f \circ (\alpha_{d^\epsilon_k(m)} \times Id)$. Now by definition of the boundary operators in $N \to P$,

$$\begin{aligned} d^\epsilon(g(m))(Id, n) &= g(m)(\delta^\epsilon_k, n) \\ &= f \circ (\alpha_{d^\epsilon_k(m)} \times n)(Id, n) \\ &= g(d^\epsilon_k(m)) \end{aligned}$$

The proof for $\multimap$ goes along the same lines. □

A further generalization is to consider transitions that belong to more general categories than **Set**. We can also separate collections of objects of dimension n into collections of objects indexed by two indices p, q with dimension $p + q$, in order to be able to trace the propagation on paths. This will prove useful when speaking about homology and homotopy.

2.2.3 Regular automata

Here, we choose n-transitions to be elements of modules or vector-spaces. This will enable us to speak about finite collections of transitions internally (see Chapter 7).

In all the rest of the text, R is a principal commutative domain

(we refer to Appendix A for details).

Looking again at Figure 2.2, we see that we can decompose M as follows. Set $M_{0,0} = (\alpha)$, $M_{1,0} = (a) \oplus (b)$, $M_{1,-1} = (\beta) \oplus (\gamma)$, $M_{2,-1} = (d) \oplus (c)$, $M_{2,-2} = (\delta) \oplus (\epsilon)$, $M_{3,-2} = (c') \oplus (d')$, $M_{3,-3} = (\zeta)$ and $M_{3,-1} = (C)$ (where (x) is the module generated by x and $\oplus$ is the direct sum of modules). The 1-path (b, c, d') can now be conveniently identified with the formal sum $b + c + d'$.

Definition 22 *A regular HDA is a direct decomposition of a free R-module M as*

$$M = \bigoplus_{p,q \geq 0} M_{p,q}$$

together with boundary operators

$$d^0_i : M_{p,q} \to M_{p-1,q}$$

$$d^1_j : M_{p,q} \to M_{p,q-1}$$

$(0 \leq i, j \leq p + q - 1)$ such that

$$d^k_i \circ d^l_j = d^l_{j-1} \circ d^k_i$$

(for all $i < j$ and $k = 0, 1$, $l = 0, 1$).

Morphisms of regular HDA are $f : M \to N$ with $f = (f_{p,q})_{p,q}$, where $f_{p,q} : M_{p,q} \to N_{p,q}$ are module homomorphisms such that $f_{p,q} \circ d^0_i = d^0_i \circ f_{p+1,q}$ and $f_{p,q} \circ d^1_j = d^1_j \circ f_{p,q+1}$ for all i, j with $0 \leq i, j \leq p + q$. The category of regular HDA is denoted by Υ_r.

We will also consider cyclic regular automata which are regular HDA in which some elements of $M_{p,q}$ and $M_{p',q'}$, $p' + q' = p + q$ may be identified.

They form a category Υ_{cr}.

The formal relationships with the semi-regular and partial models will be postponed until Part II.

2.2.4 General HDA

General HDA are a generalization of semi-regular HDA that abstract away from the combinatorics of transitions. They will prove also to be the right place in which we can speak of the geometry of a higher-dimensional transition system.

Definition and lemma 4 *Let $\mathcal{A}$ be the function from Υ_{sr} to diagrams in the category of free R-modules with,*

$$\mathcal{A}(M) = \begin{array}{ccc} \mathcal{A}(M_{p,q}) & \xrightarrow{\partial_0} & \mathcal{A}(M_{p-1,q}) \ldots \\ \Big\downarrow{\scriptstyle \partial_1} & & \vdots \\ \mathcal{A}(M_{p,q-1}) \ \ldots & & \end{array}$$

such that $\mathcal{A}(M_{p,q})$ is the free module generated by $M_{p,q}$ and,

$$\partial_0 = \sum_{i=0}^{i=p+q-1} (-1)^i d_i^0$$

$$\partial_1 = \sum_{i=0}^{i=p+q-1} (-1)^i d_i^1$$

then,

- $\partial_0 \circ \partial_0 = 0$
- $\partial_1 \circ \partial_1 = 0$
- $\partial_0 \circ \partial_1 + \partial_1 \circ \partial_0 = 0$
- *$\mathcal{A}$ lifts to morphisms f and $\mathcal{A}(f) \circ \partial_0 = \partial_0 \circ \mathcal{A}(f)$, $\mathcal{A}(f) \circ \partial_1 = \partial_1 \circ \mathcal{A}(f)$.*

PROOF. Easy verification. □

We will generally write $\underline{M}$ for the free R-module generated by M and by an abuse of notation, it will also mean the general HDA generated by a semi-regular, or a regular automaton M.

Notice also that closed partial HDA give rise to the same algebraic structure through a slightly generalized functor A which maps "missing boundaries" (the undefined $\perp$) onto 0. It seems that non-closed HDA are somewhat too unstructured for deriving an interesting algebraic structure. Regular HDA obviously provide us with boundary operators ∂_0 and ∂_1 verifying $\partial_0^2 = \partial_1^2 = \partial_0\partial_1 + \partial_1\partial_0 = 0$.

All this motivates the generalization,

Definition 23 *A (unlabeled)* higher dimensional automaton *(HDA) is a R-module M with two gradings associated to two boundary operators ∂_0 and ∂_1, that is, consists in:*

- *a decomposition: $M = \sum_{p,q \in \mathbb{Z}} M_{p,q}$, such that*

$$\forall n, \left(\sum_{p+q=n} M_{p,q} \right) \cap \left(\sum_{r+s \neq n} M_{r,s} \right) = 0$$

- *two differentials ∂_0 and ∂_1, compatible with the decomposition, giving M a structure of bicomplex:*

$$\partial_0 : M_{p,q} \longrightarrow M_{p-1,q}$$

$$\partial_1 : M_{p,q} \longrightarrow M_{p,q-1}$$

$$\partial_0 \circ \partial_0 = 0, \quad \partial_1 \circ \partial_1 = 0, \quad \partial_0 \circ \partial_1 + \partial_1 \circ \partial_0 = 0$$

There are many more relations between semi-regular HDA, partial HDA and general HDA. These will be developed in Part II. Note that here we have generalized as well to negative dimensional transitions. Their interest will be shown in Chapter 4. For our mathematically oriented readers, note that a general HDA is only a "weak" bicomplex (or double complex or complex of complex, see Appendix A) in the sense that we do not have a direct decomposition of M onto the $M_{p,q}$.

Sometimes we explicitly write the boundary operators with the HDA: (M, ∂_0,∂_1). ∂_0 is called the *source boundary operator* and ∂_1 is the *target boundary operator*. When we want to specify the domain and codomain of these boundary operators, we write $\partial_0^{p,q}$ for $\partial_0 : M_{p,q} \longrightarrow M_{p-1,q}$ and $\partial_1^{p,q}$ for $\partial_1 : M_{p,q} \longrightarrow M_{p,q-1}$. If M is in fact a direct sum of $M_{p,q}$, that is, when M is a free bigraded bidifferential R-module, then M is said to be an *acyclic HDA*, name which will be justified in Lemma 12.

If M is a finite-dimensional module, then M is called a *finite state automaton.*

Remark: A "standard" unlabeled automaton can be given the structure of a (unlabeled) higher-dimensional automaton. Let (A,Σ,δ,I,F) be an automaton; A is a set of states, Σ is a set of transitions, δ is the transition relation, $\delta \subseteq \mathcal{P}(A \times \Sigma \times A)$, I is the set of initial states, F is the set of final (or accepting) states. Define M by

$$\forall p, q,\ p+q = 0, M_{p,q} = R - Mod(A)$$

$$\forall p, q,\ p+q = 1, M_{p,q} = R - Mod(\Sigma)$$

Let D_F be the set $D_F = \{a \in A/\ \nexists \sigma, a', \quad (a, \sigma, a') \in \delta \ and \ a \notin F\}$ (it is the set of deadlocks of the automaton). Let $D_I = \{a' \in A/\ \nexists (a, \sigma, a') \in \delta \ and \ a' \notin I\}$ Then,

$$\partial_0(\sigma) = a \Leftrightarrow (\exists a' \in A, \quad (a, \sigma, a') \in \delta) \wedge (a \in A \backslash D_I)$$

$$\partial_1(\sigma) = a' \Leftrightarrow (\exists a \in A, \quad (a, \sigma, a') \in \delta) \wedge (a' \in A \backslash D_F)$$

Last but not least, ∂_j are null functions on M_0.

This construction projects all deadlocks onto 0 and all "false" initial states onto 0.

Example 4

(1)

$$\begin{array}{ccc} M_{0,1} = (a) & \xrightarrow{\partial_0} & M_{-1,1} = (1) \\ \partial_1 \downarrow & & \partial_1 \downarrow \\ M_{0,0} = (\alpha) & \xrightarrow{\partial_0} & M_{-1,0} = 0 \end{array}$$

with $\partial_0(a) = 1$ and $\partial_1(a) = \alpha$, is an acyclic finite state HDA. It comes from the standard automaton $(A, \Sigma, \delta, I, F)$ with $A=\{1, \alpha\}$, $\Sigma = \{a\}$, $\delta = \{(1, a, \alpha)\}$, $I = \{1\}$ and $F = \{\alpha\}$.

(2)

$$\begin{array}{ccc} M_{0,1} = (a) & \xrightarrow{\partial_0} & M_{-1,1} = (1) \\ \partial_1 \downarrow & & \partial_1 \downarrow \\ M_{0,0} = (1) & \xrightarrow{\partial_0} & M_{-1,0} = 0 \end{array}$$

with $\partial_0(a) = 1$ and $\partial_1(a) = 1$, is a finite state HDA which is not acyclic.

(3)

$$\begin{array}{ccccc} M_{1,1} = (A) & \xrightarrow{\partial_0} & M_{0,1} = (a) \oplus (b) & \xrightarrow{\partial_0} & M_{-1,1} = (1) \\ \partial_1 \downarrow & & \partial_1 \downarrow & & \partial_1 \downarrow \\ M_{1,0} = (a') \oplus (b') & \xrightarrow{\partial_0} & M_{0,0} = (\alpha) \oplus (\beta) & \xrightarrow{\partial_0} & M_{-1,0} = 0 \\ \partial_1 \downarrow & & \partial_1 \downarrow & & \partial_1 \downarrow \\ M_{1,-1} = (\gamma) & \xrightarrow{\partial_0} & M_{0,-1} = 0 & \xrightarrow{\partial_0} & M_{-1,-1} = 0 \end{array}$$

with $\partial_0(A) = a - b$, $\partial_1(A) = a' - b'$, $\partial_0(a) = \partial_0(b) = 1$, $\partial_1(a) = \partial_0(b') = \alpha$, $\partial_1(b) = \partial_0(a') = \beta$ and $\partial_1(a') = \partial_1(b') = \gamma$. It is an acyclic finite state HDA.

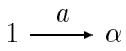
(4)

$$\begin{array}{ccccc} & & M_{0,1} = (a) \oplus (b) & \xrightarrow{\partial_0} & M_{-1,1} = (1) \\ & & \partial_1 \downarrow & & \partial_1 \downarrow \\ M_{1,0} = (a') \oplus (b') & \xrightarrow{\partial_0} & M_{0,0} = (\alpha) \oplus (\beta) & \xrightarrow{\partial_0} & M_{-1,0} = 0 \\ \partial_1 \downarrow & & \partial_1 \downarrow & & \partial_1 \downarrow \\ M_{1,-1} = (\gamma) & \xrightarrow{\partial_0} & M_{0,-1} = 0 & \xrightarrow{\partial_0} & M_{-1,-1} = 0 \end{array}$$

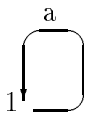
with $\partial_0(a) = \partial_0(b) = 1$, $\partial_1(a) = \partial_0(b') = \alpha$, $\partial_1(b) = \partial_0(a') = \beta$ and $\partial_1(a') = \partial_1(b') = \gamma$. It is an acyclic finite state HDA.

Actually, all these HDA are the result of the application of functor A to semi-regular HDA. We can give pictures of their geometric realization (respectively):

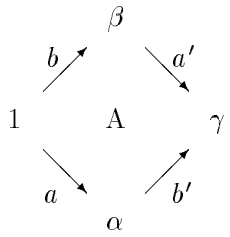
(1)



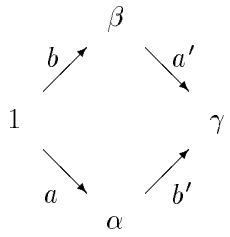
(2)



(3)



(4)



In general we consider free R-modules (see Appendix A). But non-free ones have an interest of their own, as the example below demonstrates.

Example 5 *Let $R = \mathbb{Z}$. All ideals of R are then of the form $n\mathbb{Z}$, $n \in \mathbb{Z}$. Let M be the following HDA (using the notations of Appendix A),*

- $M_{1,0} = (a)_2$,
- $M_{0,0} = (\alpha)_2$,
- *all others are null.*

with boundary operators $\partial_0(a) = \alpha$ and $\partial_1(a) = \alpha$. Then a is a cyclic transition such that $2a = 0$, i.e. such that we only look at the number of times we go through modulo 2. This is a form of built-in congruence analysis [Gra90].

(*) Lemma 3 *Let M be a HDA. Let N be the module M with the following decomposition:*

$$N_n = \sum_{p+q=n} M_{p,q}$$

Then $\partial_0 - \partial_1$ gives N the structure of graded differential module. We write N=Tot(M) (for total complex).

PROOF. Obviously $(\partial_0 - \partial_1)(N_{n+1}) \subseteq N_n$. Moreover, $(\partial_0 - \partial_1) \circ (\partial_0 - \partial_1) = \partial_0 \circ \partial_0 + \partial_1 \circ \partial_1 - (\partial_0 \circ \partial_1 + \partial_1 \circ \partial_0) = 0$. Finally, we have to verify that N_n is a grading of N. We compute:

$$N_n \cap N_m = \sum_{p+q=n} M_{p,q} \cap \left(\sum_{r+s=m} M_{r,s} \right)$$

thus, if $n \neq m$, $N_n \cap N_m = 0$.□

Conversely, one can reconstruct the two gradings of a HDA M (this translation is precisely defined in Part II, Chapter 5), given the grading of $Tot(M)$, up to a translation of the indexes of a multiple of (1,-1). We will use this to abbreviate, when possible, the two indexes to one (the one given by Tot). This extends to the indexes one can give to ∂_0, ∂_1.
For x in $M_{p,q}$, we say that x is of *dimension* $p+q$, denoted by $dim\ x = p+q$. Elements of dimension 0 are called *states*, elements of positive dimension n are *n-transitions*, elements of negative dimension n are *n-events* (see Chapter 4 for a justification of the name event).
If we have decided on a generating set, or even a basis B for M, which will be often the case, we call elementary states, transitions, and events, the states, transitions, events respectively which are elements of B. It is the case for instance when a general HDA comes from the application of functor A to a semi-regular HDA or a partial HDA.
M is said to be *bounded below* (resp. above) if there exists N such that all elements of dimension lower (resp. greater) than N are null.

Example 6 *Examples (1), (2), (3) and (4) are bounded below and above. In example (3), $dim\ A = 2$, $dim\ a = dim\ b = dim\ a' = dim\ b' = 1$ and $dim\ 1 = dim\ \alpha = dim\ \beta = dim\ \gamma = 0$. a, b, a', b', A, α, β, γ form a basis B.*

Definition 24 *A path (of length n) in a HDA M is a sequence of elements of $B=\{b_i\}$, a given generating set of M, $p=(p_i)_{0\leq i\leq n}$ such that:*

$$p_0, p_n \in M_0$$

$$dim\ p_i \geq 0$$

$$\forall i,\ \partial_0(p_i) = \sum_j \alpha_j b_j, \quad with \quad b_k = p_{i-1} \quad and \quad \alpha_k \neq 0 \quad or$$

$$\partial_1(p_i) = \sum_j \alpha_j b_j, \quad with \quad b_k = p_{i+1} \quad and \quad \alpha_k \neq 0$$

A n-dimensional path is a path whose elements are of dimension lower than (or equal to) n.

Notice that the basis appears directly from the application of the functor $\mathcal{A}$ (from semi-regular HDA to general HDA) to paths of semi-regular HDA. It is the same notion as the paths for regular automata when choosing a basis stable by the application of the boundary operators (this is always possible to construct such bases)

Example 7 *In example (3), the different paths (for the basis B=*$\{1,\alpha,\ \beta,\gamma,a,b,\ a',b',A\}$*) are subsequences of:*

(i)

$$(1, a, \alpha, b', \gamma)$$

(ii)

$$(1, a, A, b', \gamma)$$

(iii)

$$(1, a, A, a', \gamma)$$

(iv)

$$(1, b, \beta, a', \gamma)$$

(v)

$$(1, b, A, a', \gamma)$$

(vi)

$$(1, b, A, b', \gamma)$$

This means that:

- *we have chosen B as the set of observable actions*
- *(i) describes the sequential execution of a then b′*
- *(ii): suppose process a is executed on processor 1, and process b is executed on processor 2. Then this path reads:*
  from the idle state 1, processor 1 fires a
  then processor 2 fires concurrently b (transition A)
  then while processor 2 computes, processor 1 terminates a (transition b′: copy of b)
  then processor 2 also goes to an idle state, making the whole system halt to state γ.
- *(iii): keeping the same assumptions as in (ii) about the processors, this path reads:*
  from the idle state 1, processor 1 fires a
  then processor 2 fires concurrently b (transition A)
  then while processor 1 computes, processor 2 terminates b (transition a′: copy of a)
  then processor 1 also goes to an idle state, making the whole system halt to state γ.

- *(iv) describes the sequential execution of b then a'*

- *(v): from the idle state 1, processor 2 fires b*
  then processor 1 fires concurrently a (transition A)
  then while processor 1 computes, processor 2 terminates b (transition a': copy of a)
  then processor 1 also goes to an idle state, making the whole system halt to state γ.

- *(vi): from the idle state 1, processor 2 fires b*
  then processor 1 fires concurrently a (transition A)
  then while processor 2 computes, processor 1 terminates a (transition b': copy of b)
  then processor 2 also goes to an idle state, making the whole system halt to state γ.

  Notice that (ii),(iii),(v) and (vi) are maximal parallelism paths.

Paths as they are defined are not very easy to use. A useful notion is that of n-path, where we restrict actions to be of dimension n and where we collect all possible ways in which n-transitions can end.

Definition 25 *A n-path p is a finite sequence $(p_i)_{i=1,\dots,k}$ of n-transitions such that (for all $1 \leq i < k$) $\partial_1(p_i) = \partial_0(p_{i+1})$.*

To end the first part of this algebraic formulation of HDA, we need a notion of morphism to specify the "allowed" observations. For bicomplexes, there is a standard definition of morphism of bidegree (r, s) where r and s are integers. We will restrict to $r = s = 0$: observations are then some kind of simulations.

Definition 26 *(see [Lan93a]) Let (r, s) be a pair of integers. Let f be a function between two HDA $(M, \partial_0, \partial_1)$ and $(N, \partial'_0, \partial'_1)$, union of linear functions f_i: $M_{p,q} \to N_{p+r,q+s}$ (f is bigraded). Then f is called a morphism (of HDA) of bidegree (r, s) if the f_i verify:*

$$\forall p, q, \forall x \in M_{p+1,q}, \forall y \in M_{p,q+1}, f_{p,q}(\partial_0(x)) = (-1)^{r+s}\partial_0(f_{p+1,q}(x)),$$

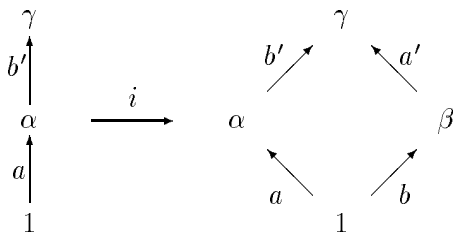
$$f_{p,q}(\partial_1(y)) = (-1)^{r+s}\partial_1(f_{p,q+1}(x)).$$

A morphism of bidegree (0,0) is just called a morphism.

The category whose objects are HDA and whose morphisms are morphisms of degree $(0, 0)$ is denoted by Υ. Its restriction to acyclic HDA is Υ_a. The restriction to free modules is Υ_F. Υ_F and Υ coincide when R is a field. The lower index f is reserved throughout this text to HDA whose underlying module is finitely generated.

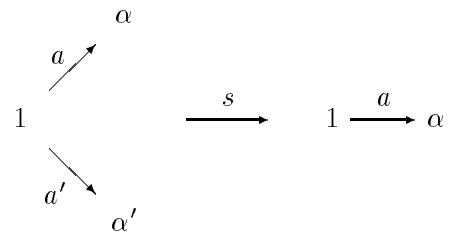
Example 8

- *A typical monomorphism (injection) is an inclusion,*



with $i(a) = a$, $i(b') = b'$, $i(1) = 1$, $i(\alpha) = \alpha$ *and* $i(\gamma) = \gamma$.

- *A typical epimorphism (surjection) is a folding,*



with $s(a) = s(a') = a$.

These examples give a pretty much accurate picture of what morphisms are, since as in the category **Set**, all morphisms can be written as the composition of an epimorphism and of a monomorphism[7]. They also provide us with a hint about the labelling of HDA.

Before coming to labels, we define the notion of subHDA ,

Definition and lemma 5 *Let* $(M, \partial_0, \partial_1)$ *and* $(N, \partial'_0, \partial'_1)$ *be two HDA. Then* N *is a sub-HDA of* M *if and only if* $\forall p, q$, $N_{p,q}$ *is a sub-module of* $M_{p,q}$ *and* $\partial'_{j|N_{p,q}} = \partial_{j|N_{p,q}}$ $(j = 0, 1)$. *Sub-HDA of* M *can be identified with monomorphisms into* M.

PROOF. Easy. □

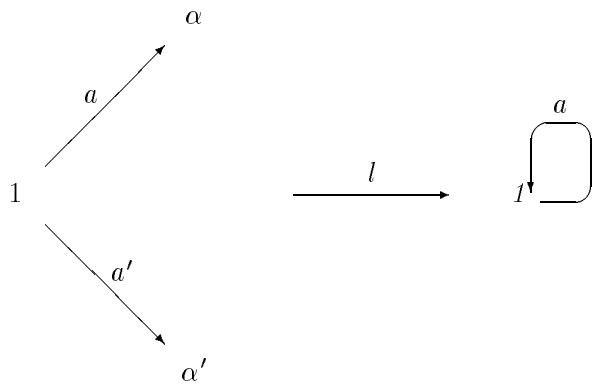
2.2.5 Labeled HDA

A category of labeled HDA

Let P be a (unlabeled) HDA. Labelling P consists in identifying some transitions of P to a common token. This identification of "physical" transitions (those of P) by labels can be thought of as a folding, or as a projection morphism onto a "labelling" HDA L. Thus labels are transitions of L, equivalence classes of transitions of P.

[7] This will be entailed by the existence of quotient objects (see Chapter 3).

Example 9



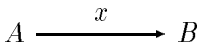
with $l(a) = l(a') = a$.

Definition 27 *A labeled HDA (over L) is a pair* (M, l) *composed of an unlabeled HDA* M*, and a morphism* $l : M \longrightarrow L$*. A morphism* $f : (M, l) \longrightarrow (M', l')$ *of labeled HDA is a morphism of HDA between* M *and* M' *such that* $l' \circ f = l$.

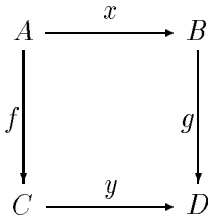
Hence the category of labeled HDA over L is the slice category Υ/L. The categories of labeled semi-regular HDA, labeled acyclic HDA, labeled regular and labeled partial HDA are the subcategories of Υ/L formed by changing Υ into Υ_{sr}, Υ_a, Υ_r and $P\Upsilon_{sr}$ respectively in the above definition.

Example 10 *let L be the HDA such that* $L_0 = (\mathbf{1})$, $L_1 = (\mathbf{a}) \oplus (\mathbf{b})$ *with* $\partial_j(\mathbf{a}) = \partial_j(\mathbf{b}) = \mathbf{1}$*. Let M be the HDA of example (4). Define a module homomorphism* l *by* $l(a) = l(a') = \mathbf{a}, l(b) = l(b') = \mathbf{b}$ *and* $l(1) = l(\alpha) = l(\beta) = l(\gamma) = \mathbf{1}$*. Then* l *is a morphism, and* (M, l) *is a labeled HDA over L.*

More generally, we can be interested in transformations between labels as well (like restriction and relabelling of CCS-like process algebras, Section 1.3.1). For this to be expressible in our framework, we have to consider the category $\Upsilon^{\rightarrow}$ of arrows of Υ. Objects are arrows

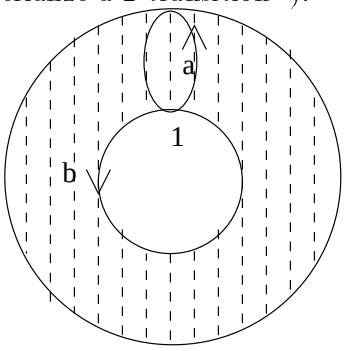


and morphisms between x and y are pairs of morphisms (in Υ_f) (f, g) such that the following diagram commutes,



g is a relabelling function.

Figure 2.7: A subset with two one-transitions of T_2 of the labelling automaton L (torus shaped – dashed lines materialize a 2-transition -).



Some labellings

We will often have a set A of actions and corresponding to that set there will be a "natural" labelling HDA. Let L_A (or L when the context makes it clear what set of actions it is based on) be the semi-regular HDA defined by (see Figure 2.7),

- $L_0 = \{1\}$, $L_1 = A$, $L_k = A^k$,
- $\forall x \in L_1$, $d^0_0(x) = d^1_0(x) = 1$
- $\forall k \geq 2, \forall (x_1, \ldots, x_k) \in L_k$, $d^0_i(x_1, \ldots, x_k) = (x_1, \ldots, x_i, x_{i+1}, \ldots, x_k)$, and $d^1_i = d^0_i$

By extension, we will call L_A the, respectively, regular, general HDA generated by the previous semi-regular HDA.

L_A is natural in that, on the one hand, it gives the ordinary labelling to standard automata as one can see in Example 10 and, on the other hand, it is a "natural" generalization (see Figure 2.7). We will come back to that more formally in the next chapter.

Finally, notice that the SOS format generalizes to labeled HDA. We define an entailment relation $\models$ to relate $l : M \to L$ to its sub-HDA, and we write,

$$M \models \quad s \xrightarrow{\quad a \quad} s' \quad \Leftrightarrow \quad \exists t,\ l(t) = a,\ (s \xrightarrow{\quad t \quad} s') \quad \subseteq M$$

Notice that for all M and states s of M we may write,

$$M \models s \xrightarrow{\ l(s)\ } s$$

Summary In this chapter, we have introduced the main categories of HDA which we will use in the following chapters. First, we introduced the semi-regular HDA and showed that they correspond geometrically to unions of hypercubes of all dimensions, i.e. computer-scientifically to sequences of allocations and deallocations of processes on processors, making them suitable for

expressing dynamic properties of interest of concurrent systems. It was shown in particular that synchronized products are cartesian products, parallel compositions are tensors, forking processes are in function spaces and non-deterministic choices correspond to sums. Together with the "denotational" or categorical semantics approach, we developed a SOS-like notation for the description of all semi-regular HDA. It was shown that some SOS-rules did match perfectly the categorical constructions.

Then we introduced some refinements of this base model. Partial HDA add a "natural" notion of deadlocking behaviours to semi-regular HDA. The main categorical properties were shown to be preserved. Regular HDA are semi-regular HDA "over R-modules". This means that they add some deadlocking behaviours (by using the zero of R-modules) and they internalize the notion of finite collection of transitions, hence of finite path. Last but not least, the general HDA abstract away from the combinatorics of regular HDA and generalize them by adding the notion of event. The structure they give is a "weak" double complex of R-modules structure. We ended by giving a way to label these transition systems in a natural manner (as "labelling" morphisms).

Chapter 3

Relationship with other models of concurrency

In this chapter, we show that some kinds of transition systems (like ordinary ones, asynchronous ones) can be interpreted within the HDA model in different natural ways according to the level of parallelism and mutual exclusion properties we are willing to observe. At the end of the chapter we also use some of the adjunctions of [WN94] to relate these interpretations with event structures and Mazurkiewitz traces.

3.1 Transition systems and HDA

Consider the category $\mathcal{C}$ of labeled Higher-Dimensional Automata consisting of morphisms $l : M \to L$ (L fixed once and for all) such that

$$(H) : (\forall i,\ d_i^0(x) = d_i^0(x')) \wedge (\forall i,\ d_0^1(x) = d_0^1(x')) \wedge l(x) = l(x') \Leftrightarrow x = x'$$

i.e. of "well" labeled HDA such that there is only one representative of a given action between two given states. This does in no way restrict the power of expression of HDA if we keep in mind that labels and states are the only observable objects.

The morphisms in this category are as usual $f = (g, h) : (l : M \to L) \to (l' : M' \to L)$ with $g : M \to M'$ and $h : L \to L$ such that the following diagram is commutative,

$$\begin{array}{ccc} M & \xrightarrow{g} & M' \\ \downarrow{\scriptstyle l} & & \downarrow{\scriptstyle l'} \\ L & \xrightarrow{h} & L \end{array}$$

By abuse of notation, we will identify f, g and h in the following.

We will also consider in the following the category $\mathcal{C}p$ (respectively $\mathcal{C}p'$) of pairs $(l : M \to L, s)$ with $l \in T_1(\mathcal{C})$ (respectively $l \in \mathcal{C}$) and $s \in M_0$ (the "initial" state) and morphisms preserving initial states ("pointed" HDA). $\mathcal{C}p'$ is the category of Higher-dimensional Transition Systems (HTS).

We prove that $\mathcal{C}p$ is isomorphic to TS_A. As a matter of fact, the categories are defined in quite similar terms. States of ordinary transition systems are of the same nature as states of labeled HDA and source and target representation of transitions is nothing but a functional interpretation of the relation $Tran$.

This is done formally by constructing two functors $\mathcal{U} : TS_A \to \mathcal{C}p$ and $\mathcal{V} : \mathcal{C}p \to TS_A$ inverse of each other,

- $(M, l : M \to L, i) = \mathcal{U}(S, A, Tran, j)$ with
  - $M_0 = S$,
  - $M_1 = \{a_{s,s'} / a \in A, s \xrightarrow{a} s' \in Tran\}$,
  - $i = j$,
  - $d^0_0(a_{s,s'}) = s$, $d^1_0(a_{s,s'}) = s'$,
  - $l(a_{s,s'}) = a$, $l(s) = 1$,
- $(S, A, Tran, j) = \mathcal{V}(M, l : M \to L, i)$ with,
  - $S = M_0$,
  - $j = i$,
  - $s \xrightarrow{a} s' \in Tran$ iff $\exists x \in M_1$, such that $l(x) = a$, $d^0_0(x) = s$ and $d^1_0(x) = s'$,

Action of the functors on morphisms is as follows,

- if $f = (\sigma, \lambda) : (S_0, A_0, Tran_0, j_0) \to (S_1, A_1, Tran_1, j_1)$ is a morphism of transition systems then
  - $\mathcal{U}(f)(a_{s,s'}) = \lambda(a)_{\sigma(s),\sigma(s')}$,
  - $\mathcal{U}(f)(s) = \sigma(s)$ $(s \in M_0)$
- if $f : (l_0 : M_0 \to L, i_0) \to (l_1 : M_1 \to L, i_1)$ is a morphism in $\mathcal{C}p$ then $\mathcal{V}(f) = (\sigma, \lambda) : \mathcal{V}(l_0 : M_0 \to L, i_0) \to \mathcal{V}(l_1 : M_1 \to L, i_1)$ with
  - $\sigma(s) = f(s)$ (s any state of $\mathcal{V}(l_0 : M_0 \to L, i_0)$),
  - $\lambda(a) = f(a)$ (a any label in $\mathcal{V}(l_0 : M_0 \to L, i_0)$)

Now, in order to compare the category of higher-dimensional transition systems with ordinary transition systems we only have to look at how to retract $\mathcal{C}p'$ onto its subcategory $\mathcal{C}p$. This boils down to looking at the different adjunctions we have between $\Upsilon^1_{sr} = T_1(\Upsilon_{sr})$ and Υ_{sr}.

We have mainly two different adjunctions between Υ^1_{sr} and Υ_{sr} using T_1 (to keep the underlying ordinary transitions unchanged in the interpretation) among all the possible ones. These adjunctions are nothing but comparisons of models by abstract interpretations [CC92a].

Lemma 3 *The inclusion functor $\mathcal{I} : \Upsilon^1_{sr} \to \Upsilon_{sr}$ is left-adjoint to the truncation functor $T_1 : \Upsilon_{sr} \to \Upsilon^1_{sr}$.*

The truncation functor $T_n : \Upsilon_{sr} \to \Upsilon^n_{sr}$ is left-adjoint to a functor $\mathcal{G}_n : \Upsilon^n_{sr} \to \Upsilon_{sr}$.

PROOF. For the first part of the lemma, we associate to every morphism of Υ_{sr}, $f : \mathcal{I}(M) \to N$ a morphism of Υ^1_{sr}, $T_1(f) : T_1(\mathcal{I}(M)) = M \to T_1(N)$. This is actually a bijective mapping between these two kinds of morphisms since for every morphism $g : M \to T_1(N)$ in Υ^1_{sr}, the composite

$$f : \quad \mathcal{I}(M) \xrightarrow{\mathcal{I}(g)} \mathcal{I}(T_1(N)) \xrightarrow{j} N$$

where $j : \mathcal{I}(T_1(N)) \to N$ is the natural inclusion of $\mathcal{I}(T_1(N))$ into N, is its inverse mapping.

For the second part of the lemma, notice that Υ_{sr} is,

- small co-complete (see Chapter 2),
- well-co-powered,
- has small hom-sets,
- and has a small generating set (the $D_{[n]}$).

Moreover, Υ^n_{sr} has also small hom-sets, so it is enough to verify, by Freyd's special adjoint functors theorem (see [ML71]) that T_n commutes with colimits. This is obvious. □

The functor $\mathcal{G}_n$ has actually a nice interpretation.

Lemma 4 *(m-connectedness[1] of $\mathcal{G}_n(X)$ for all $m \geq n$) Let $X \in \Upsilon^n_{sr}$. $\mathcal{G}_n(X)$ is the least (for the inclusion ordering) semi-regular HDA such that,*

- $X \subseteq \mathcal{G}_n(X)$,
- *any morphism $f : \dot{D}_{[m]} \to \mathcal{G}_n(X)$ $(m \geq n)$ can be extended to a morphism $D_{[m]} \to \mathcal{G}_n(X)$ where $\dot{D}_{[m]} = T_{m-1}(D_{[m]})$.*

This means that all $(m-1)$-interleavings[2] $(m \geq n)$ are interpreted under $\mathcal{G}_n$ as (i.e. mapped by $\mathcal{G}_n$ onto) truly concurrent executions of m actions (see Chapter 8).

These adjunctions now induce the adjunctions

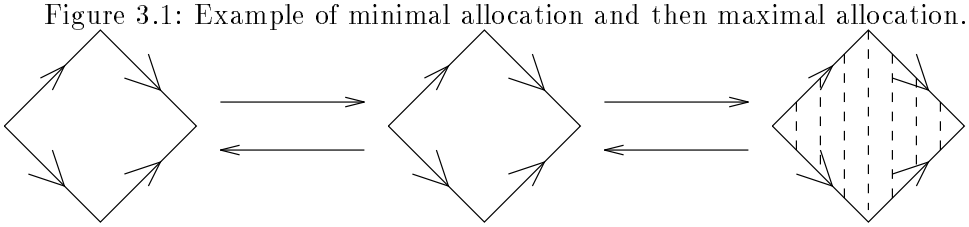
$$TS_A \underset{\mathcal{V}_{min}}{\overset{\mathcal{U}_{min}}{\rightleftarrows}} \mathcal{C}p' \underset{\mathcal{U}_{max}}{\overset{\mathcal{V}_{max}}{\rightleftarrows}} TS_A$$

$\mathcal{U}_{min}$ represents ordinary transition systems of TS_A literally, i.e. every transition is mapped by $\mathcal{U}_{min}$ onto a 1-transition. The implicit parallelism that may have been in a transition system is discarded by $\mathcal{U}_{min}$ and is just interpreted as interleaving. This corresponds to the traces on a one processor machine.

[1] See for instance [Spa66].

[2] Or $(m-1)$-mutual exclusion, that is the execution of m actions under the constraint that no more than $m-1$ actions can be run asynchronously. Its traces are represented by $\dot{D}_{[m]}$. This behaviour is easily programmed using semaphores [Dij68].

Figure 3.1: Example of minimal allocation and then maximal allocation.



On the contrary, $\mathcal{U}_{max}$ **detects all possible interleavings and interprets them uniquely as purely concurrent executions**. No bound on the dimension of transitions generated is put: this corresponds to traces on a machine with infinitely many processors. $\mathcal{U}_{min}$ **describes the minimal allocation (on a multi-processor machine) strategy**, given an ordinary transition system, whereas $\mathcal{U}_{max}$ **describes the maximal allocation strategy**[3] (see Figure 3.1).

Intermediate interpretations (or allocations) of ordinary transition systems can be found if we use the similar adjunctions $(m < n)$

$$\Upsilon^m_{sr} \underset{T_m}{\overset{I}{\rightleftarrows}} \Upsilon^n_{sr} \underset{\mathcal{G}^n_m}{\overset{T_m}{\rightleftarrows}} \Upsilon^m_{sr}$$

Then under the interpretation induced by $(T_m, \mathcal{G}^n_m)$,

all k-mutual exclusions are identified with level of parallelism equal to $k+1$ if $k \geq m$.

We will actually prove this in a more general context, using methods from homological algebra in Chapter 8.

Under the interpretation induced by (I, T_m),

all levels of parallelism k $(k \geq m)$ are interpreted as interleavings of asynchronous executions of m actions.

3.2 Asynchronous transition systems and HDA

Let ATS_E be the full subcategory of ATS consisting of asynchronous transition systems on a given set of events E. We show that the pair of adjoint functors $(\mathcal{U}_{min}, \mathcal{V}_{min})$ (resp. $(U_{max}, \mathcal{V}_{max})$) induces a pair of adjoint functors $(\mathcal{E}, \mathcal{F})$ (resp. $(\mathcal{G}, \mathcal{H})$) between ATS_E and the full subcategory $d\mathcal{C}p'$ of $\mathcal{C}p'$ consisting of deterministic higher-dimensional transition systems, i.e. HDA satisfying

$$l(t) = l(t') \wedge d^0_0(t) = d^0_0(t') \Rightarrow d^1_0(t) = d^1_0(t')$$

(for t, t' 1-transitions). It corresponds to a minimal allocation strategy (resp. maximal allocation strategy).

[3] I owe Alan Mycroft (at WSA'93) the idea of developping the interpretation of such adjunctions.

We first define functors $\mathcal{E}, \mathcal{F}$,

$$d\mathcal{C}p_2' \underset{\mathcal{F}}{\overset{\mathcal{E}}{\rightleftarrows}} ATS_E$$

($d\mathcal{C}p_2$ is the full subcategory of $d\mathcal{C}p'$ consisting of transition systems of dimension less than or equal to two) by,

- (P, j, l, L)=$\mathcal{F}(S, i, E, I, Tran)$ with,
  - $j = i$, $P_0 = S$,
  - $P_1 = \{t_{s,s'} / s \xrightarrow{t} s' \in Tran\}$,
  - $d_0^0(t_{s,s'}) = s$, $d_0^1(t_{s,s'}) = s'$ and $l(t_{s,s'}) = t$,
  - $P_2 = \{ab_{s,s',s'',u} / aIb \wedge a_{s,s'} \in P_1 \wedge b_{s,s''} \in P_1 \wedge b_{s',u} \in P_1 \wedge a_{s'',u} \in P_1\}$,
  - $d_0^0(ab_{s,s',s'',u}) = a_{s,s'}$, $d_1^0(ab_{s,s',s'',u}) = b_{s,s''}$, $d_0^1(ab_{s,s',s'',u}) = b_{s',u}$, $d_1^1(ab_{s,s',s'',u}) = a_{s'',u}$ and $l(ab_{s,s',s'',u}) = (a, b)$.
- $\mathcal{E}(P, P \xrightarrow{l} L, j) = (S, i, E, I, Tran)$ with,
  - $i = j$, $S = P_0$,
  - $s \xrightarrow{t} s' \in Tran \Leftrightarrow (\exists x \in P_1, l(x) = t \wedge d_0^0(x) = s \wedge d_0^1(x) = s')$
  - aIb if and only if $\exists C \in P_2$, $l(C) = (a, b)$

$\mathcal{F}$ has the same action on the underlying ordinary transition system of an asynchronous transition system as functor $\mathcal{U}$. Similarly for $\mathcal{E}$ which acts as $\mathcal{V}$ on the underlying ordinary transition systems. $\mathcal{F}$ fills in all interleavings of two independent actions by 2-transitions. $\mathcal{E}$ imposes two actions to be independent if and only if there exists a truly concurrent execution of them somewhere in the labeled HDA.

The action on morphisms is again easy to define.

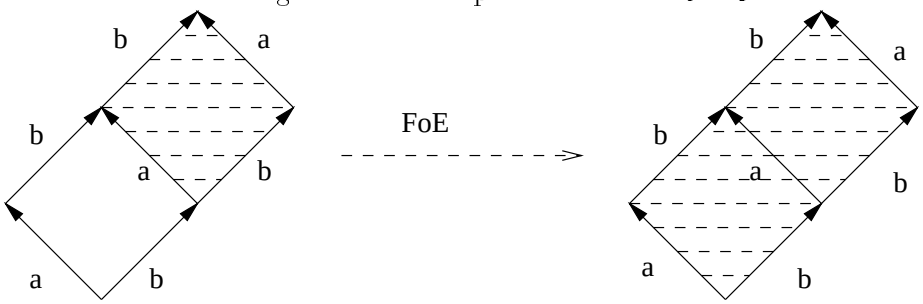
Let $f = (\sigma, \lambda) : (S, i, E, I, Tran) \to (S', i', E', I', Tran')$ be a morphism of asynchronous transition systems. Then $g = \mathcal{F}(f) : \mathcal{F}(S, i, E, I, Tran) \to \mathcal{F}(S', i', E', I', Tran')$ is defined by,

- $g(s) = \sigma(s)$ for $s \in \mathcal{F}(S, i, E, I, Tran)_0$,
- $g(t_{s,s'}) = \lambda(t)_{\sigma(s),\sigma(s')}$ for $t_{s,s'} \in \mathcal{F}(S, i, E, I, Tran)_1$,
- $g(ab_{s,s',s'',u}) = \lambda(a)\lambda(b)_{\sigma(s),\sigma(s'),\sigma(s''),\sigma(u)}$ for $ab_{s,s',s'',u} \in \mathcal{F}(S, i, E, I, Tran)_2$.

Finally, for $g : (P, P \xrightarrow{l} L, j) \to (P', P' \xrightarrow{l'} L, j')$ a morphism of $d\mathcal{C}p_2$ we define $f = (\sigma, \lambda) : \mathcal{E}(P, P \xrightarrow{l} L, j) \to \mathcal{E}(P', P' \xrightarrow{l'} L, j')$,

- $\sigma(s) = g(s)$ for s state of $\mathcal{E}(P, P \xrightarrow{l} L, j)$,

Figure 3.2: Example of co-retract $\mathcal{F} \circ \mathcal{E}$



- let $s \xrightarrow{t} s'$ be a transition in $\mathcal{E}(P, P \xrightarrow{l} L, j)$. Then by definition of $\mathcal{E}$ there exists $x \in P_1$ with $l(x) = t$, $d_0^0(x) = s$ and $d_0^1(x) = s'$. Then we can set $\lambda(t) = l \circ g(x)$. This definition does not depend on s, s',

Proposition 4 *$(\mathcal{E}, \mathcal{F})$ is a pair of adjoint functors.*

PROOF. We verify easily that $\mathcal{E} \circ \mathcal{F} = Id$. We now have to show that the identity morphism is co-universal (case (iv) of Theorem 2 page 81, [ML71]) more precisely that for all $(S, i, E, I, Tran) \in ATS_E$, $Id : \mathcal{E}\mathcal{F}(S, i, E, I, Tran) \to (S, i, E, I, Tran)$ is universal from $\mathcal{E}$ to $(S, i, E, I, Tran)$.

Let $f = (\sigma, \lambda) : (S_1, i_1, E, I_1, Tran_1) = \mathcal{E}(P_1, j_1, l_1, L) \to (S, i, E, I, Tran)$ be a morphism of asynchronous transition systems. Remember that σ is a function from $(P_1)_0 = S_1$ to S and $\lambda : E \to E$. We define $f' : (P_1, j_1, l_1, L) \to (P, j, l, L) = \mathcal{F}(S, i, E, I, Tran)$ as follows,

- $f'_{|(P_1)_0} = \sigma$,
- $f'_{|(P_1)_1}$ is such that for all $t_{s,s'} \in (P_1)_1$, $f'_{|(P_1)_1} = \lambda(t)_{\sigma(s),\sigma(s')}$,
- for all $ab_{s,s',s'',u} \in (P_1)_2$, $f'_{|(P_1)_2}(ab_{s,s',s'',u}) = \lambda(a)\lambda(b)_{\sigma(s),\sigma(s'),\sigma(s''),\sigma(u)}$.

It is easy to see that $\mathcal{E} f' = f$, hence the universality. □

Composing $(\mathcal{E}, \mathcal{F})$ with $(T_2, \mathcal{G}_2)$ we obtain a pair of adjoint functors

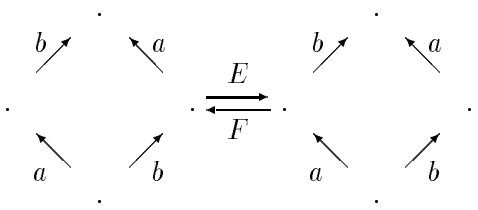
$$d\mathcal{C}p' \underset{\mathcal{G}_2 \circ \mathcal{F}}{\overset{\mathcal{E} \circ T_2}{\rightleftarrows}} ATS_E$$

corresponding to a **maximal interpretation of the independence relation** (maximal allocation strategy).

All k-mutual exclusions ($k \geq 2$) are interpreted as level of parallelism $k + 1$.

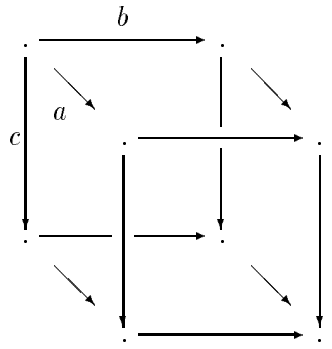
Example 11

- *supposing* $\neg(aIb)$ *we have,*



This shows that 1-mutual exclusions can be expressed under the maximal allocation interpretation of asynchronous transition systems.

- *Supposing* aIb, aIc *and* bIc *the following asynchronous transition system is mapped onto a filled-in cube (i.e. onto* $D_{[3]}$*),*



This shows that 2-mutual exclusions are identified with asynchronous execution of three actions.

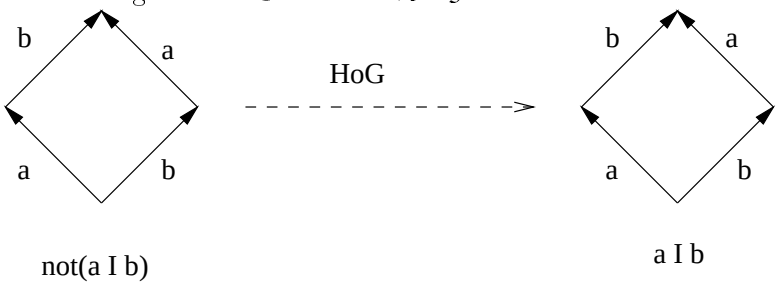
The minimal allocation strategy can be obtained very easily through the adjunction

$$ATS_E \underset{\mathcal{H}}{\overset{\mathcal{G}}{\rightleftarrows}} d\mathcal{C}p_1'$$

where,

- (P, j, l, L)=$\mathcal{G}(S, i, E, I, Tran)$ with,
  - $j = i$, $P_0 = S$,
  - $P_1 = \{t_{s,s'} / s \xrightarrow{t} s' \in Tran\}$,
  - $d_0^0(t_{s,s'}) = s$, $d_0^1(t_{s,s'}) = s'$ and $l(t_{s,s'}) = t$,
- $\mathcal{H}(P, P \xrightarrow{l} L, j) = (S, i, E, I, Tran)$ with,
  - $i = j$, $S = P_0$,
  - $s \xrightarrow{t} s' \Leftrightarrow (\exists x \in P_1, l(x) = t \wedge d_0^0(x) = s \wedge d_0^1(x) = s')$,
  - $I = E \times E$.

Figure 3.3: Co-retract $\mathcal{H} \circ \mathcal{G}$



Again, $\mathcal{G}$ and $\mathcal{H}$ act as $\mathcal{U}$ and $\mathcal{V}$ respectively on the underlying ordinary transition systems. This time, $\mathcal{G}$ forgets all the information about the independence of actions whereas $\mathcal{H}$ considers all actions to be independent.

The actions of morphisms are straightforward.

Proposition 5 *$(\mathcal{G}, \mathcal{H})$ is a pair of adjoint functors.*

PROOF. It is easy to see that $\mathcal{G} \circ \mathcal{H} = Id$. We now have to verify that for all $(P, j, l, L) \in dCp_1'$, $Id : \mathcal{GH}(P, j, l, L) \to (P, j, l, L)$ is universal from $\mathcal{G}$ to (P, j, l, L).

Let $f : (P_1, j_1, l_1, L) = \mathcal{G}(S_1, i_1, E, I_1, Tran_1) \to (P, j, l, L)$ be a morphism of HDA. We define a morphism of asynchronous transition systems $f' = (\sigma, \lambda) : (S_1, i_1, E, I_1, Tran_1) \to (S, i, E, I, Tran)$ as follows,

- $\sigma(s) = f(s)$ for $s \in S_1 = (P_1)_0$,
- $\lambda(e) = f(e)$ for all $e \in E$.

This defines a morphism of asynchronous transition systems since for all e, e', $eI_1e' \Longrightarrow f'(e)If'(e')$ because aIb is always true for any a, b.

It is then easy verification to see that $Gf' = f$, hence the universality of Id. □

Composing this with $(\mathcal{I}, T_1)$, we get the minimal allocation strategy (on one processor) of asynchronous transition systems.

Under this interpretation, all mutual exclusions and concurrent executions are identified.

3.3 Mazurkiewitz traces and HDA

We define a pair $(\mathcal{V}, \mathcal{W})$ of adjoint functors between the subcategory GTL_L of GTL (where the alphabet is fixed to L) and Cp_2 as follows,

If (M, I, L) is a generalized Mazurkiewitz trace we set $\mathcal{V}(M, I, L)$ to be the HDA $(l : P \to L, j)$ with,

- $j = \epsilon$,

- $P_0 = \{[s]_{\cong} / s \in M\}$,
- $P_1 = \{([s]_{\cong}, [sa]_{\cong}) / s \in M, a \in L\}$, $d^0_0([s]_{\cong}, [sa]_{\cong}) = [s]_{\cong}$, $d^0_1([s]_{\cong}, [sa]_{\cong}) = [sa]_{\cong}$ and $l([s]_{\cong}, [sa]_{\cong}) = a$,
- $P_2 = \{ab_{[s]_{\cong}} / s \in M, a \in L, b \in L, aI_sb\}$ and,
  - $d^0_0(ab_{[s]_{\cong}}) = ([s]_{\cong}, [sa]_{\cong})$,
  - $d^0_1(ab_{[s]_{\cong}}) = ([s]_{\cong}, [sb]_{\cong})$,
  - $d^1_0(ab_{[s]_{\cong}}) = ([sb]_{\cong}, [sba]_{\cong})$,
  - $d^1_1(ab_{[s]_{\cong}}) = ([sa]_{\cong}, [sab]_{\cong})$,
  - $l(ab_s) = a \otimes b$.

If $(l : P \to L, j)$ is a HTS then define $\mathcal{W}(l : P \to L, j) = (M, I, L)$ with,

- $M = \{(l(x_0), \ldots, l(x_n)) / (j, x_0, s_1, x_1, \ldots, x_n, s_{n+1})$ is a path in $P\}$,
- Let $(j, x_0, s_1, x_1, \ldots, x_n, s_{n+1})$ be a path of dimension 1 in P.

  $s = (l(x_0), \ldots, l(x_n)) \in M$. Then set aI_sb if and only if there is $C \in P_2$ such that $d^0_0 d^0_1(C) = s_{n+1}$ and $l(C) = a \otimes b$.

This means that the strings of the Generalized Mazurkiewitz trace are precisely traces in the HDA, and the independence relation is once again read in the 2-transitions of the HDA.

In the following, we actually restrict to $\mathcal{CC}p_2$ the subcategory of deterministic and connected HTS, i.e. the subcategory of HTS $(l : P \to L, j)$ such that for all state s of M there exists a unique path from j to s.

Proof. We first prove that $\mathcal{W} \circ \mathcal{V} = Id$. Let $(M, I, L) \in GTL_L$ and define,

$$\mathcal{V}(M, I, L) = (l : P \to L, j)$$

$$\mathcal{W}(l : P \to L, j) = (M', I', L')$$

Let $s \in M$, $s = a_0 \ldots a_n$ $(a_i \in L)$. Then we can see by a straightforward induction on n that $([\epsilon]_{\cong}, x_0, [a_0]_{\cong}, \ldots, x_n, [s]_{\cong})$ is a path of dimension one in P, where $x_i = ([a_0 \ldots a_{i-1}]_{\cong}, [a_0 \ldots a_i]_{\cong})$. This implies, by definition of $\mathcal{W}$, that $l(x_0) \ldots l(x_n)$ which is equal to $a_0 \ldots a_n$ is in M'. Therefore $M \subseteq M'$.

Conversely, let $a_0 \ldots a_n \in M'$. Then there exists $(j, x_0, s_1, x_1, s_2, \ldots, x_n, s_n)$ a 1-path in P such that $l(x_i) = a_i$. Then, $l(x_0) \ldots l(x_n) = a_0 \ldots a_n \in M$. Therefore $M = M'$.

Now, aI'_sb if and only if $s = l(x_0) \ldots l(x_n)$ and there is $C \in P_2$ such that $d^0_0 d^0_1(C) = s_{n+1}$ and $l(C) = a \otimes b$. By construction, this is equivalent to $C = ab_{[s]_{\cong}}$ and aI_sb.

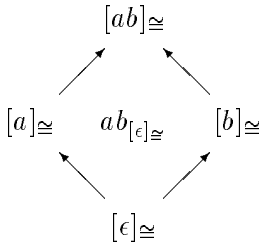
Now, we prove that when we restrict to deterministic connected HDA, there is a natural transformation $\mathcal{V} \circ \mathcal{W} \to Id$. Let $(M, I, L) = \mathcal{W}(l : P \to L, j)$ and $(l' : P' \to L, j') = \mathcal{V}(M, I, L)$. Then $j' = \epsilon = j$. Let $t \in P_0$. As P is connected, there is a unique 1-path from j to t in P_0. Let s be its trace in L. We define a

graded function $f = (f_i)_i$ from P to P' by first setting $f(t) = [s]_{\cong}$. For $x \in P_1$, define $f_1(x) = (f_0(d^0_0(x)), f_0(d^1_0(x)))$. For $A \in P_2$ with $l(A) = a \otimes b$, define $f_2(A) = ab_{[f_0(d^0_0 d^0_1(A))]_{\cong}}$. It is easy to see that f defines a morphism of HDA from P to P'. Moreover, it is natural in its argument. It defines the co-unit of the adjunction. The fact that it is universal from $\mathcal{V}$ to Id is left to the reader. □

Example 12 • *Let (M, I, L) be,*

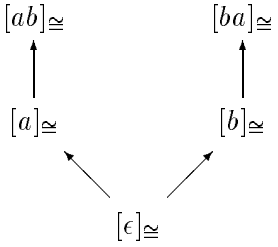
- $L = \{a, b\}$,
- $M = \{\epsilon, a, b, ab, ba\}$,
- *I is the constant function from M to $2^{L\times L}$ such that for all $t \in M$, $xI(t)y$ if and only if $x = a$, $y = b$ or $x = b$, $y = a$*

Then by the pair of adjoint functors above, we see that it corresponds to the HDA,



- *Let (M, I, L) be,*
  - $L = \{a, b\}$,
  - $M = \{\epsilon, a, b, ab, ba\}$,
  - *I is the constant function from M to $2^{L\times L}$ such that for all $t \in M$, $x, y \in L$, $xI(t)y$ is false.*

This corresponds via the pair of adjoint functors above to the HDA,



The allocation strategies deriving from this pair of adjoint functors are of the same kind as for asynchronous transition systems. Basically, we have a **maximal allocation strategy which identifies all k-mutual exclusions with level of parallelism $k+1$ when $k \geq 2$, and a minimal allocation strategy which does not express any level of parallelism strictly more than 2.**

3.4 Event structures and HDA

We use the equivalence with deterministic labeled event structure (in [SNW94]) to have interpretations of HDA in terms of a truly concurrent, linear time, behavioural model of concurrency.

It is proven in [SNW94] that one particular full subcategory of labeled event structures, the category DES of so-called deterministic labeled event structures and the category of generalized trace languages are equivalent.

This is proven with the partial morphisms only, and we review this construction in order to show that it works also for total morphisms.

We derive a generalized Mazurkiewitz trace language (M, I, L) from a labeled event structure $(E, \leq, \#, l, L)$ if we suppose it is deterministic, i.e., for any configuration c and any pair of events $e, e' \in E$, whenever $c \vdash e$, $c \vdash e'$ and $l(e) = l(e')$ then $e = e'$ as follows,

- $M = \{l^*(e_1 \ldots e_n)/\{e_1, \ldots, e_n\}$ is a securing$\}$. Notice that as $(E, \leq, \#, l, L)$ is deterministic, M is in bijection with the set of strings of events. Call this bijection Sec,
- $I_s = \{(a, b)/sab \in M, Sec(sab) = xe_0e_1,$ and $e_0 co e_1\}$

Conversely, we can define a deterministic labeled event structure $(E, \leq, \#, l, L)$ from a generalized Mazurkiewitz trace language (M, I, L).

Events in (M, I, L) are just traces identified in a suitable way, using the independence relation. Formally, let $\sim$ be the least equivalence such that,

- aI_sb implies $sa \sim sba$,
- $s \sim s'$ implies $sa \sim s'a$

then the set of events occuring in $s \in M$ is defined to be

$$Ev(s) = \{[u]_\sim / u \text{ is a non empty prefix of } s\}$$

Event $[s]_\sim$ is now before event $[s']_\sim$ ($[s]_\sim \leq [s']_\sim$) if and only if for all $u \in M$, $[s']_\sim \in Ev(u)$ implies $[s]_\sim \in Ev(u)$.

Events $[s]_\sim$ and $[s']_\sim$ are in conflict if and only if for all $u \in M$, $[s]_\sim \in Ev(u)$ implies $[s']_\sim \notin Ev(u)$.

Finally, $l([s]_\sim) = a$ if and only if $s = s'a$ for some s'.

These two transformations between Generalised Mazurkiewitz Traces and Deterministic Labelled Event Structures extend to functors which actually define an equivalence of categories [SNW94].

Example 13

- *We recall that the following Generalized Mazurkiewitz Trace Language (M, I, L) defines a mutual exclusion between letters a and b,*
  - $L = \{a, b\}$,

- $M = \{\epsilon, a, b, ab, ba\}$,
- I *is the constant function from* M *to* $2^{L \times L}$ *such that for all* $t \in M$, $x, y \in L$, $xI(t)y$ *is false.*

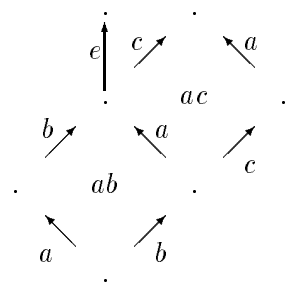
and corresponds (by the equivalence above) to the Deterministic Labelled Event Structure $(E, \leq, \#, l, L)$,

- $E = \{\epsilon, a, b, ab, ba\}$,
- $\leq$ *is the prefix ordering on strings of* a*s and* b*s,*
- $a\#b$, $ab\#ba$, $a\#ba$, $b\#ab$,
- l *and* L *are the obvious labellings.*

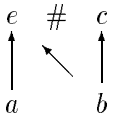
- *The following Generalized Mazurkiewitz Trace Language* (M, I, L) *defines the concurrent execution of actions* a *and* b,
  - $L = \{a, b\}$,
  - $M = \{\epsilon, a, b, ab, ba\}$,
  - I *is the constant function from* M *to* $2^{L \times L}$ *such that for all* $t \in M$, $xI(t)y$ *if and only if* $x = a$, $y = b$ *or* $x = b$, $y = a$.

  and corresponds (by the equivalence above) to the Deterministic Labelled Event Structure $(E, \leq, \#, l, L)$,
  - $E = \{[\epsilon], [a], [b]\}$,
  - $[\epsilon] \leq [a]$, $[\epsilon] \leq [b]$,
  - *there is no conflict,*
  - $L = \{a, b\}$ *and* $l([a]) = a$, $l([b]) = b$.

- *Using the maximal allocation strategy for Mazurkiewitz traces and the equivalence with event structures, we see that the semi-regular HDA (represented here via the adjunction with Top),*

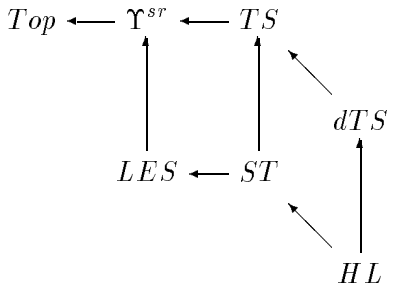


is represented by the labeled event structure,



3.5 Other models

In [SNW94], it is proved that we have the following adjunctions (the arrows go from the more concrete to the more abstract models), together with the adjunctions we have proven[4],



where LES is the category of labeled event systems (equivalent to generalized Mazurkiewitz traces and pomsets without autoconcurrency), ST is the category of synchronization trees, HL the category of Hoare languages and dTS the category of deterministic labeled transition systems.

Summary We have constructed formal correspondences (pairs of adjoint functors) in the style of [WN94] between some of the operational models of Chapter 1 and the semi-regular HDA of Chapter 2. The difference with [WN94] is that we are looking at a variety of adjunctions between models and at their meaning in terms of properties of dynamic behaviours that are forgotten.

There is in particular an isomorphism of categories between the category of labeled transition systems and a category of labeled semi-regular HDA of dimension 1. This in turn induced different ways to understand diamond shapes in transition systems. One way was to interpret them as purely non-deterministic interleavings, i.e. as an execution on one processor and another was to interpret them as purely asynchronous executions (on some number of processors). These interpretations were shown to be pairs of adjoint functors (or abstract interpretations) and to correspond to different allocation strategies (the minimal one for the former, the maximal one for the latter).

Then we showed that these adjunctions could be generalized to "decorated" transition systems like asynchronous transition systems and generalized Mazurkiewitz traces. The independence relation of ATS corresponds to a 2-transition (in fact many of them) and we have shown that different allocation strategies could be formalized. The independence relation of GTL was also shown to correspond to 2-transitions, but this time depending on a local state. We ended the chapter by using some of the results of [WN94] which gave us correspondences with event structures, Hoare languages and synchronization trees.

This gives a hint why HDA seem to be well suited for studying allocation strategies (and scheduling properties) of concurrent systems, since we have a notion of level of parallelism and its dual, a level of mutual exclusion. This will be used in Chapter 7.

[4]This is actually a commutative diagram.

Part II

Semantic Definitions

Chapter 4

Categorical properties of HDA

In this chapter, we consider constructions on the category Υ of HDA with morphisms of degree $(0,0)$, and on the full subcategory Υ_a of acyclic HDA. Some of these will bear a striking resemblance to operators of process algebra (this is much in accordance with the results by Glynn Winskel on deriving process algebras from the categorical constructions on several models of concurrency). Some others will have no known equivalent and will be discussed as new notions, except of course if we had already seen them in the previously studied categories of semi-regular, regular and partial HDA.

4.1 Limits and colimits

4.1.1 Zero object

Lemma 5 *0 is the zero object in categories Υ and Υ_a, that is, is both their initial and terminal object.*

PROOF. Obvious: there is only one morphism from 0 to any HDA M (initial object property) and only one morphism from an HDA M to 0 (final object property). □

4.1.2 Finite limits and colimits

Lemma 6 *Categories Υ, Υ_a are finitely complete.*

PROOF. We just need to prove that kernels and cartesian products exist in these categories.
Let P and Q be two HDA. Let $M = P \times Q$ as sets, and define on M:

- a structure of R-module by $\forall a \in R$, $\forall (x,y) \in M$, $a(x,y) = (ax, ay)$, and $\forall (x,y),(z,t) \in M$, $(x,y)+(z,t) = (x+y, z+t)$

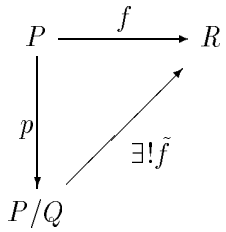
- two boundary operators by $\forall (x,y) \in M$, $\partial_j((x,y)) = (\partial_j(x), \partial_j(y))$
- two compatible gradings by $M_{i,j} = P_{i,j} \times Q_{i,j}$

It is a simple verification to see that M is a HDA and is the cartesian product of M and N in the above mentioned categories.
Now for kernels (or equalizers), let P, Q and $f, g : P \longrightarrow Q$ be respectively HDA and morphisms of HDA. Let $M = \{x / f(x) = g(x)\} = Ker\,(f - g)$. It is obviously a subHDA of P, and together with its inclusion morphism into P forms the equalizer of P, Q, f and g. □

Remarks

- Υ_F has in general only cartesian products and not equalizers.
- Υ has cartesian products and initial and final objects which coincide, thus it cannot be cartesian closed otherwise it would be a completely degenerated category (all objects would be isomorphic).

We have the notion of kernel of a morphism (equalizer of this morphism and the map 0). This enables us to study the existence of quotients. Let P, Q and R be three HDA, $Q \subseteq P$, and $f : P \to R$ a morphism of HDA such that $Ker\ f$ is a sub-HDA of Q. Then the quotient P/Q of P by Q, together with its "canonical projection" $p : P \to P/Q$, if they exist, are the unique HDA and epimorphism such that,



Lemma 7 *Categories Υ_a and Υ have quotient objects.*

Proof. We begin to prove it for acyclic automata. Quotient objects exist in the category of R-modules, so we can define, as modules, for Q sub-HDA of P,

$$(P/Q)_{i,j} = P_{i,j}/Q_{i,j}$$

They come together with projections $p_{i,j} : P_{i,j} \to P_{i,j}/Q_{i,j}$.
We can define also boundary operators ∂'_0 and ∂'_1 from $P_{i,j}/Q_{i,j}$ to $P_{i-1,j}/Q_{i-1,j}$ and from $P_{i,j}/Q_{i,j}$ to $P_{i,j-1}/Q_{i,j-1}$ respectively, by,

$$\partial'_0([x]_{Q_{i,j}}) = [\partial_0(x)]_{Q_{i-1,j}}$$

$$\partial'_0([x]_{Q_{i,j}}) = [\partial_0(x)]_{Q_{i,j-1}}$$

where $[y]_A$ denotes the class of y modulo A. It is a valid definition since if we have two representants x and x' of the same class modulo $Q_{i,j}$, $\partial_0(x) - \partial_0(x') \in$

$\partial_0(Q_{i,j}) \subseteq Q_{i-1,j}$. Same for ∂_1. It is an easy verification to show that the ∂'_0, ∂'_1 are boundary operators and then that this definition verifies the property of quotient objects with the canonical projection p being the union of the $p_{i,j}$.

For Υ, we have in particular to verify that the same construction verifies the property that no two elements of different dimension are equal. We follow the construction of [Lan93b].

Let M be the R-module $P/Q = \bigoplus_{k,l \in \mathbb{Z}} P_{k,l}/Q_{k,l}$ and H the smallest R-module of M generated by elements of the form $\sum_{k,l} p_{k,l}(x_{k,l})$ such that,

- only a finite number of $x_{k,l}$ are non null,
- $x_{k,l} \in M_{k,l}$,
- $\sum_{k,l} x_{k,l} = 0$.

It is easy to see that M/H, which is by definition the R-module underlying the HDA P/Q, is an amalgamated sum of the R-modules $P_{k,l}/Q_{k,l}$.

Now, we construct the canonical projection $p : P \to \sum_{k,l \in \mathbb{Z}} P_{k,l}/Q_{k,l}$ as follows.

Let $x = \sum_{k,l} x_{k,l} \in P$ with $x_{k,l} \in P_{k,l}$. We define $p(x) = \sum_{k,l} [p_{k,l}(x_{k,l})]$ where $[y]$ denotes the equivalence class modulo H.

Then, $\sum_{k,l} x_{k,l} = \sum_{k,l} y_{k,l}$ implies $\sum_{k,l} p_{k,l}(x_{k,l}) = \sum_{k,l} p_{k,l}(y_{k,l})$, hence p is well defined. It is obviously a surjective module homomorphism. If $x = \sum_{k,l} x_{k,l}$ then $p(x) = 0$ implies $\exists h = \bigoplus_{k,l} p_{k,l}(h_{k,l}) \in H$ such that $\bigoplus_{k,l} p_{k,l}(x_{k,l} - h_{k,l}) = 0$. Hence, $\forall k, l$, $x_{k,l} - h_{k,l} \in N_{k,l}$. But by definition of H, $\sum_{k,l} h_{k,l} = 0$, therefore $x = \sum_{k,l} (x_{k,l} - h_{k,l}) \in N$. This proves $Ker\ p = N$.

Then p induces an isomorphism $\overline{p} : P/Q \to M/H = \sum_{k,l} P_{k,l}/Q_{k,l}$. □

Υ_F does not have quotient objects in general. For instance, for $R = \mathbb{Z}$, $M_0 = (\alpha)$, $N_0 = (2\alpha)$, then $(M/N)_0 = (\alpha)_2$ which is not free.

Lemma 8 *The categories Υ, Υ_a are finitely cocomplete.*

Proof. We prove first that coproducts $\oplus$ exist in these categories. If P and Q are HDA, then define $P \oplus Q$ to be,

- as modules, $(P \oplus Q)_{i,j} = P_{i,j} \oplus Q_{i,j}$ (see [Lan93a] for the definition of the direct sum $\oplus$ on modules),
- together with boundary operators $\partial_i[P \oplus Q] = \partial_i[P] \oplus \partial_i[Q]$ $(i = 0, 1)$.

Finally, cokernels exist in the category Υ: let P, Q and $f, g : P \longrightarrow Q$ be respectively HDA and morphisms of HDA. Let $M = Q/Im\,(f-g)$ and $p : Q \to M$

the canonical surjection. They are HDA and morphism of HDA respectively, by Lemma 7. It is indeed the coequalizer. □

Therefore amalgamated sums exist in the categories Υ and Υ_a. The amalgamated sum of X and Y over Z is denoted by $X \coprod_Z Y$.

We have only finite sums in Υ_F in general.

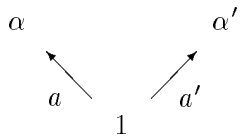
Definition and lemma 6 *Let A be a submodule of an HDA M. Then there exists a smallest sub-automaton of M containing A, denoted by $Clos(A)$. We define an operation $+$ on submodules of M, by:*

$$A + B = Clos(A) \coprod_{Clos(A) \cap Clos(B)} Clos(B)$$

Remark: *Clos* is clearly the topological operation of closure. We will see that $+$ corresponds to the geometric operation of connected sum.

We call in_1 and in_2 respectively, the canonical morphisms from A to $A + B$, and B to $A + B$.

Example 14 *Let M be automaton (1) of Example 4, and let N be the automaton M, where a is replaced by a' and α, by α'. Then $M + M'$ can be pictured as:*



Notice that cartesian product and coproduct are isomorphic constructions, so we have biproducts.

4.1.3 (†) Enriched structures

Actually, the most interesting part of the categorical structure of HDA for use of homological (or K-theory) methods is something which will not be used for giving semantics of concurrent programs.

(*) Lemma 4 Υ *is an abelian category*

PROOF. (see [ML71])

All equalizers are monics. Υ has a zero object and biproducts. Therefore, it is an additive category.

This means that $\Upsilon(A, B)$ for all A and B can be given a structure of commutative monoid[1] by setting, for $f, f' : A \to B$, $f + f' = \nabla \circ (f \oplus f') \circ \Delta$, where Δ and ∇ are the diagonal and co-diagonal morphisms respectively.

Moreover, every morphism has a kernel and a cokernel and every monic arrow is a kernel, every epi is a cokernel. This proves that Υ is an abelian category. □

4.1.4 Direct and inverse limits

We first recall some notions of algebra, that can be found for example in [Lan93a] or [Mas78]:

(*) Definition 2 *Let (I,≤) be a directed set, and C be a category. A direct system of C consists of a function which assigns to each i belonging to I, an object C_i of C, and to each pair i,j ∈ I such that $i \leq j$, a morphism $M_{ij} : C_i \to C_j$, such that the following holds:*

- *For any $i \in I$, M_{ii} is the identity map of C_i,*
- *If $i \leq j \leq k$, then $M_{ik} = M_{jk} M_{ij}$.*

For those who are familiar with the language of category theory, a direct system of C is a covariant functor from the category I (viewed as the graph of $\leq$ on I) to the category C.

(*) Definition 3 *Let C be a category, and M a direct system of C. A direct limit of M[2] consists of an object L of C and a collection of morphism $p_i : C_i \to L$ verifying the two conditions:*

(i) *For any i,j ∈ I, $i \leq j$, $p_i = p_j M_{ij}$.*

(ii) *For any object A of C and collection of morphisms $q_i : C_i \to A$ satisfying the previous property, there exists a unique morphism $h : L \to A$ such that for every $j \in I$, $q_j = h p_j$ (universal property).*

We write:

$$\varinjlim C_i = L$$

Now, we show that the notion of direct limit is functorial, that is, we can compute a notion of direct limit for maps (to be defined) between direct systems.

[1] In modern terminology we would say that Υ is a CMon-enriched category, where CMon is the category of commutative monoids.

[2] Direct limits are a particular case of colimits, sometimes called filtered colimits [ML71].

(*) Definition 4 *[Mas78] A map of a direct system C into a direct system C' consists of an order preserving map f: $I \to I'$, and for each $i \in I$, a homomorphism $F_i : C_i \to C'_{f(i)}$ subject to the following condition: if $i \leq j$, then $M'_{f(i),f(j)} \circ F_i = F_j \circ M_{i,j}$.*

The reader will certainly have recognized that F is a natural transformation (see [FS90]) of the functor M into the functor $M'f$.

Let us denote by $(L, (p_i)_{i\geq 0})$ and $(L', (p'_i)_{i\geq 0})$ the direct limits of the direct systems C and C' respectively. Now, consider for each $i \in I$ the homomorphism $p'_{f(i)} \circ F_i : C_i \to L'$. Then this collection of homomorphisms verifies condition (i) of definition 3. Hence, by condition (ii) of the same definition, there exists a unique homomorphism $F_\infty : L \to L'$ such that $\forall i, p'_{f(i)} \circ F_i = F_\infty \circ p_i$. This homomorphism F_∞ is called the direct limit of the homomorphisms F_i.
$\varinjlim$ is a covariant functor from the category of direct systems of C and maps of direct systems to C.

(*) Lemma 5 *Direct limits exist in the category of modules and linear maps.*

(*) Proof. See [Lan93a]. □

Corollary 1 *Direct limits exist in Υ, Υ_a and DG, the category of graded differential modules.*

Proof. Let $(C^i, \partial_0^i, \partial_1^i)$ be the objets of a direct system of Υ. Let $M^{i,j}$ be the morphisms from C^i to C^j of this direct system. They respect the gradings, so they induce morphisms $M_{k,l}^{i,j} : C_{k,l}^i \longrightarrow C_{k,l}^j$. Let $(L_{k,l}, p_{k,l}^i)$ be the direct limit of the direct system $D_{k,l} = (C_{k,l}^i, M_{k,l}^{i,j})_{i,j}$ of R-modules.

Consider now the map $(f, F^i)_i$ from $D_{k,l}$ to $D_{k-1,l}$:

$$f : I \longrightarrow I, \quad f = Id$$

$$F^i : C_{k,l}^i \longrightarrow C_{k-1,l}^i, \quad F^i = \partial_0$$

We have $M_{k-1,l}^{i,j} \circ F^i = F^j \circ M_{k,l}^{i,j}$ because $M^{i,j}$ is a morphism of bicomplex. Therefore, it is a map of direct systems. Let $\partial_0^\infty = \varinjlim \partial_0^i$. We can make the same construction for $F'^i = \partial_1$ which leads to an operator $\partial_1^\infty : L_{k,l} \longrightarrow L_{k,l-1}$, $\partial_1^\infty = \varinjlim \partial_0^i$. We have $\forall i$, $\partial_0^i \circ \partial_0^i = 0$. Thus, $\varinjlim (\partial_0^i \circ \partial_0^i) = \partial_0^\infty \circ \partial_0^\infty = 0$ by functoriality of $\varinjlim$. Similarly, $\partial_1^\infty \circ \partial_1^\infty = 0$ and $\partial_0 \circ \partial_1 + \partial_1 \circ \partial_0 = 0$. Moreover, we know that:

$$p_{k-1,l}^i \circ \partial_0^i = \partial_0^\infty \circ p_{k,l}^i$$

$$p_{k,l-1}^i \circ \partial_1^i = \partial_1^\infty \circ p_{k,l}^i$$

Thus, p^i is a morphism of HDA between $(C^i, \partial_0^i, \partial_1^i)$ and $(L, \partial_0^\infty, \partial_1^\infty)$. We have constructed a cone (L, p^i) in Υ. We have to prove it is universal.

Suppose we have morphisms p'^i: $C^i \longrightarrow L'$. Then there exists a unique linear map $h : L \longrightarrow L'$, such that $\forall i$, $h \circ p_i = p'_i$. We already know, by construction, that h respects the two gradings of the HDA L and L'. For $x \in C^i$, we have $h(p_i(\partial_j(x))) = h(\partial_j(p_i(x))) = p'_i(\partial_j(x)) = \partial_j(h(p_i(x)))$. Thus, for all i, h is a morphism from $Im\, p_i$ to L'. L is an amalgamated sum of the $Im\, p_i$, hence h is a morphism from L to L'. Therefore, $(L, \partial_0^\infty, \partial_1^\infty)$ is the direct limit in Υ, of $(C^i, \partial_0^i, \partial_1^i)$.

Υ_a is a full subcategory of Υ. This entails that direct limits exist in Υ_a as well.

The proof that direct limits exist in DG is similar. □

Proposition 6 *Categories Υ, Υ_a are cocomplete.*

PROOF. This is entailed by the previous result and Lemma 8. □

Inverse limits are direct limits in the opposite category. We just state:

Lemma 9 *Inverse limits exist in the categories Υ, Υ_a.*

PROOF. They exist in the category of R-modules. We conclude by using the same arguments as for the direct limits. □

Then,

Proposition 7 *Categories Υ, Υ_a are complete.*

PROOF. Follows from the finite completeness and the existence of inverse limits. □

4.2 Tensor and Hom

(*) Definition and lemma 1 *Let M and N be two HDA. Define a R-module T by:*

$$T_{p,q} = \sum_{k,l} M_{p-k,q-l} \otimes N_{k,l}$$

and two operators (j=0,1)

$$\partial_j(x \otimes y) = \partial_j(x) \otimes y + (-1)^{(dim x)} x \otimes \partial_j(y)$$

that is,

$$\partial_j^{n,m} = \sum_{p+r=n, q+s=m} (\partial_j^{p,q}[M] \otimes Id + (-1)^{p+q} Id \otimes \partial_j^{r,s}[N])$$

Then T is a HDA, called the tensor product of M and N.

The meaning of the amalgamated sum $\sum_{k,l} M_{p-k,q-l} \otimes N_{k,l}$ is intuitively clear. Nevertheless, we prefer to give a more formal definition here.

If M and N are two acyclic HDA, then the definition of the tensor product becomes the classical one (at least for complexes, see [ML63]),

$$T_{p,q} = \bigoplus_{k,l} M_{p-k,q-l} \otimes N_{k,l}$$

Suppose now that M and N are not acyclic. Let L be the bigraded module with,

$$L_{p,q} = \bigoplus_{k,l} M_{p-k,q-l} \otimes N_{k,l}$$

and H the sub-module of $\bigoplus_{p,q} L_{p,q}$ such that $H_{p,q}$ is generated by elements of the form,

$$m_{p,q} \otimes n_{r,s} - m_{p',q'} \otimes n_{r',s'}$$

with,

- $m_{p,q} \in M_{p,q}$, $m_{p',q'} \in M_{p',q'}$, $n_{r,s} \in N_{r,s}$ and $n_{r',s'} \in N_{r',s'}$,
- $m_{p,q} = m_{p',q'}$ and $n_{r,s} = n_{r',s'}$.

Now,

$$T_{p,q} = \sum_{k,l} M_{p-k,q-l} \otimes N_{k,l}$$

denotes the equivalence classes modulo H of elements of $L_{p,q}$[3]. Notice that the sign in the boundary formula is the only one compatible with this quotient operation.

PROOF. First, we have to verify: $\partial_0 \circ \partial_1 + \partial_1 \circ \partial_0 = 0$.

We compute:

$$\begin{aligned}\partial_0^{m,n} \circ \partial_1^{m,n+1} \quad = \quad & \sum_{p+r=m,q+s=n} \Big(\partial_0^{p,q}[M] \circ \partial_1^{p,q+1}[M] \otimes \mathrm{Id} + (-1)^{r+s} \partial_0^{p,q}[M] \otimes \partial_1^{r,s}[N] + \\ & (-1)^{p+q+r+s+1} \mathrm{Id} \otimes \partial_0^{p,q}[M] \circ \partial_1^{r,s+1}[N] + (-1)^{p+q} \partial_1^{p,q}[M] \otimes \partial_0^{r,s}[N] \Big)\end{aligned}$$

and

$$\begin{aligned}\partial_1^{m,n} \circ \partial_0^{m+1,n} \quad = \quad & \sum_{p+r=m,q+s=n} \Big(\partial_1^{p,q}[M] \circ \partial_0^{p+1,q}[M] \otimes \mathrm{Id} + (-1)^{r+s} \partial_1^{p,q}[M] \otimes \partial_0^{r,s}[N] + \\ & (-1)^{p+q+r+s+1} \mathrm{Id} \otimes \partial_1^{p,q}[M] \circ \partial_0^{r+1,s}[N] + (-1)^{p+q} \partial_0^{p,q}[M] \otimes \partial_1^{r,s}[N] \Big)\end{aligned}$$

[3]For more details, one can look at [Lan93b].

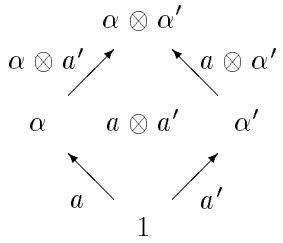
Their sum is equal to zero because of the relations of commutation (or anti-commutation) between $\partial_0[M]$ and $\partial_1[M]$, and between $\partial_0[N]$ and $\partial_1[N]$.
Secondly,

$$\begin{aligned}\partial_{\mathrm{k}} \circ \partial_{\mathrm{k}}(\mathrm{x} \otimes \mathrm{y}) &= \partial_{\mathrm{k}} \left(\partial_{\mathrm{k}}[\mathrm{M}](\mathrm{x}) \otimes \mathrm{y} + (-1)^{dim\ \mathrm{x}} \mathrm{x} \otimes \partial_{\mathrm{k}}[\mathrm{N}](\mathrm{y})\right) \\ &= (-1)^{dim\ \partial_{\mathrm{k}}[\mathrm{M}](\mathrm{x})} \partial_{\mathrm{k}}[\mathrm{M}](\mathrm{x}) \otimes \partial_{\mathrm{k}}[\mathrm{N}](\mathrm{y}) + (-1)^{dim\ \mathrm{x}} \partial_{\mathrm{k}}[\mathrm{M}](\mathrm{x}) \otimes \partial_{\mathrm{k}}[\mathrm{N}](\mathrm{y}) \\ &= 0\end{aligned}$$

□

This construction is the tensor product of the two complexes associated with M_1 and M_2 (see [Mas78]). The reader can verify that it is actually a tensor product in the category (see for instance [FS90]) of complexes with morphisms given by Definition 26.

Example 15 *The tensor product of two copies of automaton (1) of Example 4 can be pictured as:*



This corresponds to the picture we had for the tensor product of semi-regular HDA. The correspondence between the categorical structures of the different kinds of HDA will be made formal in Section 4.3.1.

Lemma 10 *Let M be an HDA. Then $(\cdot \otimes M)$, $(M \otimes \cdot)$ are endofunctors on Υ, Υ_a.*

Proof. Let $F(X) = X \otimes N$. We define the action of F on morphisms $f : X \longrightarrow Y$ by:

$$F(f) : F(X) \longrightarrow F(Y), \quad F(f)(x \otimes m) = f(x) \otimes m$$

$F(f)$ is a morphism, because,

$$\begin{aligned}\partial_i(f(x) \otimes m) &= \partial_i(f(x)) \otimes m + (-1)^{dim f(x)} f(x) \otimes \partial_i(m) \\ &= F(f)(\partial_i(x) \otimes m + (-1)^{dim x} x \otimes \partial_i(m)) \\ &= F(f)(\partial_i(x \otimes m))\end{aligned}$$

For acyclic automata, we verify that $F(Q)_{m,n}$ is a bigrading when $Q_{p,q}$ and $M_{r,s}$ are. □

(*) Definition and lemma 2 *Let P and Q be two acyclic HDA. We define $Hom(P,Q)$ as the acyclic HDA whose objects of index p,q are:*

$$Hom(P,Q)_{p,q} = \prod_{r,s\in\mathbb{Z}} Hom(P_{r,s}, Q_{p+r,q+s})$$

(where $Hom(P_{r,s}, Q_{p+r,q+s})$ is the R-module of R-linear maps from $P_{r,s}$ to $Q_{p+r,q+s}$) and whose boundary operators are:

$$\partial_0(f_{r,s}) = \partial_0[Q] \circ f_{r,s} - (-1)^{p+q} f_{r-1,s} \circ \partial_0[P]$$

$$\partial_1(f_{r,s}) = \partial_1[Q] \circ f_{r,s} - (-1)^{p+q} f_{r,s-1} \circ \partial_1[P]$$

for i=0,1, and $f_{r,s}$ being the (r,s) component of some f in $Hom(P,Q)_{p,q}$.

Proof. Easy verification. We have for instance:

$$\begin{aligned}\partial_0\partial_1 f &= \partial_0 \circ \partial_1 \circ f - (-1)^{p+q}\partial_1 \circ f \circ \partial_0 - (-1)^{p+q}\partial_0 \circ f \circ \partial_1 \\ &\quad -(-1)^{2(p+q)-1} f \circ \partial_1 \circ \partial_0\end{aligned}$$

and,

$$\partial_1\partial_0 f = \partial_1 \circ \partial_0 \circ f - (-1)^{p+q-1}\partial_0 \circ f \circ \partial_1 - (-1)^{p+q}\partial_1 \circ f \circ \partial_0 + f \circ \partial_0 \circ \partial_1$$

Thus, $\partial_0\partial_1 f + \partial_1\partial_0 f = 0$. □

Again, when P and Q are not acyclic, we would have to make more precise the meaning of $\prod_{r,s\in\mathbb{Z}} Hom(P_{r,s}, Q_{p+r,q+s})$. It is intuitively clear that we want to identify maps in it which are "equal" on "equal" elements, equality meaning here the one we may have between elements having different indexes. The formula above for $Hom(P,Q)_{r,s}$ unfortunately does not work (see [Lan93b] for a counter-example).

In fact, if P and Q are general HDA, we need to define $Hom(P,Q)_{r,s}$ to be the module of morphisms from P to Q of degree (r,s). Note that this definition coincides with the one we have given in the case of acyclic HDA. The boundary operators are defined in a similar way. Notice that the signs in these formulae are the only ones we could choose, compatible with the identification of elements of the same dimension.

Proposition 8 *Let M be an HDA. Functors $(\cdot \otimes M)$, $(M \otimes \cdot)$, $Hom(M,\cdot)$ and $(M + \cdot)$ are ω-continuous, that is, preserve direct limits.*

Proof. Let $C = (C_i, M_{i,j})_{i,j}$ be a direct system in Υ and $(D, p_i)_i = \varinjlim C$. For F a functor, $F(C) = (F(C_i), F(M_{i,j}))_{i,j}$ is a direct system in Υ and $(F(D), F(p_i))_i$ is a cone for $F(C)$. We just need to prove it is universal, and then we will conclude $F(D) = \varinjlim F(C)$. Let $(E, q_i)_i$ be a cone for $F(C)$ and

B=$\{b_1, ..., b_j, ...\}$ be a generating set for M.
We begin by considering the functor $F = (M \otimes \cdot)$. For all j, $(E, q_{i,j})_i$, where $q_{i,j}(x) = b_j \otimes x$, is a cone for C. Thus there exists $h_j : D \longrightarrow E$ such that $\forall i$, $q_{i,j} = h_j \circ p_i$. Now, let h be defined on $F(D)$ by $h(b_j \otimes x) = h_j(x)$. h verifies the identities $q_i = h \circ F(p_i)$, $\forall i$. Thus, using the same arguments as in Lemma 1 h is a morphism on $Im\ F(p_i)$, hence h is a morphism from $F(D)$ to E.
We have to prove the uniqueness of such an h. Suppose we have h and h' verifying $q_i = h \circ F(p_i) = h' \circ F(p_i)$. Then, $h(b_j \otimes \cdot) = h_j$ and $h'(b_j \otimes \cdot) = h'_j$ verify $q_{i,j} = h_j \circ p_i = h'_j \circ p_i$. Therefore $h_j = h'_j$, $\forall j$, and $h = h'$ (because they are generated by the h_j and h'_j).
For $F = \cdot \otimes M$, we reason in a similar way.
The cases $F = (M + \cdot)$ and $F = Hom(M, \cdot)$ are left to the reader.
□

4.2.1 Autonomous structure

Lemma 11 *Υ is symmetric monoidal.*

PROOF. The map,

$$A \otimes B \longrightarrow B \otimes A$$

$$a \otimes b \longrightarrow (-1)^{dim\ a\, dim\ b} b \otimes a$$

is a natural (in A and B) isomorphism. Thus Υ is symmetric monoidal. □

(*) Proposition 3

$$Hom(P \otimes Q, R) \cong Hom(P, Hom(Q, R))$$

(*) PROOF. We verify this first when all automata are acyclic.
Let $T_{p,q}$=$Hom(P \otimes Q, R)_{p,q}$ and $S_{p,q}$= $Hom(P, Hom(Q, R))_{p,q}$.
Notice that, as R-modules (and for any R-modules A, B, C):

$$Hom(A \oplus B, C) \cong Hom(A, C) \times Hom(B, C)$$

$$Hom(A \otimes B, C) \cong Hom(A, Hom(B, C))$$

$$Hom(A, B \times C) \cong Hom(A, B) \times Hom(A, C)$$

Then we have as R-modules:

$$\begin{aligned} T_{p,q} &= \prod_{r,s \in \mathbb{Z}} Hom(\bigoplus_{k,l} P_{k,l} \otimes Q_{r-k,s-l}, R_{p+r,q+s}) \\ &\cong \prod_{r,s \in \mathbb{Z}} \prod_{k,l} Hom(P_{k,l} \otimes Q_{r-k,s-l}, R_{p+r,q+s}) \\ &\cong \prod_{r,s \in \mathbb{Z}} \prod_{k,l} Hom(P_{k,l}, Hom(Q_{r-k,s-l}, R_{p+r,q+s})) \end{aligned}$$

Now, still as R-modules:

$$\begin{aligned} S_{p,q} &= \prod_{r,s\in\mathbb{Z}} Hom(P_{r,s}, Hom(Q,R)_{p+r,q+s}) \\ &\cong \prod Hom(P_{r,s}, \prod_{i,j\in\mathbb{Z}} Hom(Q_{i,j}, R_{p+r+i,q+s+j})) \\ &\cong \prod_{r,s\in\mathbb{Z}} \prod_{i,j\in\mathbb{Z}} Hom(P_{r,s}, Hom(Q_{i,j}, R_{p+r+i,q+s+j})) \\ &\cong T_{p,q} \end{aligned}$$

Now, we have to see if this isomorphism of R-modules is an isomorphism of R-bicomplexes.

$$\partial_j[T](f_{i,j,k,l}) = \partial_j[R] \circ f_{i,j,k,l} - (-1)^{p+q} f_{i,j,k,l} \circ (\partial_j[P] \otimes Id[Q])$$
$$-(-1)^{p+q+k+l} f_{i,j,k,l} \circ (Id[P] \otimes \partial_j[Q])$$

where $f_{i,j,k,l}$ acts on $P_{k,l} \otimes Q_{i-k,j-l}$, and,

$$\partial_j[S](f_{i,j,k,l}) = \partial_j[R] \circ f_{i,j,k,l} - (-1)^{p+q} f_{i,j,k,l} \circ \partial_j[Q] - (-1)^{p+q+k+l} f_{i,j,k,l} \circ \partial_j[P]$$

where $f_{i,j,k,l}$ is the (i,j)th component of the (k,l)th component of f.
In the latter term, $f_{i,j,k,l}$ is in $Hom(P, Hom(Q,R))_{p,q}$, and in the former, it is in $Hom(P \otimes Q, R)_{p,q}$, by isomorphism u given by $u(f)(x \otimes y) = (f(x))(y)$. Applying this to the former relation, we have:
$u(\partial_j[T](f_{i,j,k,l})) = \partial_j[R] \circ u(f_{i,j,k,l}) - (-1)^{p,q} u(f_{i,j,k,l}) \circ (\partial_j[P] \otimes Id[Q])$
$$-(-1)^{p+q+k+l} u(f_{i,j,k,l}) \circ (Id[P] \otimes \partial_j[Q])$$

Thus,
$\partial_j[T](f_{i,j,k,l})(x)(y) = \partial_j[R](f_{i,j,k,l})(x)(y) - (-1)^{p+q} f_{i,j,k,l}(\partial_j[P](x))(y)$
$$-(-1)^{p+q+k+l} f_{i,j,k,l}(x)(\partial_j[Q](y))$$

This equates $\partial_j[T]$ with $\partial_j[S]$.
The case of Υ is no more complex but too tedious to be detailed here. □

For f an object of dimension n of $Hom(P,Q)$, we say that:
if $n > 0$ then f *increases the degree of parallelism* (by n) and
if $n < 0$ then f *decreases the degree of parallelism* (by n).

Now, Hom is a contravariant functor in its first argument, and covariant in its second argument. Let $f : P \longrightarrow P'$ be a morphism of HDA. Then $Hom(f,Q) : Hom(P',Q) \longrightarrow Hom(P,Q)$ is the morphism such that $Hom(f,Q)(h) = h \circ f$. If $g : Q \longrightarrow Q'$ is a morphism, then $Hom(P,g) : Hom(P,Q) \longrightarrow Hom(P,Q')$ is the morphism defined by $Hom(P,g)(h) = g \circ h$.

Example 16 *Let a be an elementary object of dimension n in a HDA M. Suppose that $f_a(x) = a \otimes x$ is well defined on M. Then $dim\ f = n$ and $\partial_j(f) = f_{\partial_j(a)}$.*

(*) Lemma 6 *The map $eval : Hom(P,Q) \otimes P \longrightarrow Q$ defined by*

$$eval(\{f_{i,j}\}_{i,j}, x)^4 = f_{p,q}(x)$$

if $x \in P_{p,q}$, is a morphism (of degree 0), called the evaluation map.

PROOF. Suppose $dim\ f = n$ and $x \in P_{p,q}$. Then,

$$\partial_j(eval(\{f_{i,j}\}_{i,j} \otimes x) \quad = \quad \partial_j[Q](f_{p,q}(x))$$

and

$$\begin{aligned} eval(\partial_j(\{f_{i,j}\}_{i,j} \otimes x) &= eval(\{\partial_j(f_{i,j})\}_{i,j} \otimes x + (-1)^n \{f_{i,j}\}_{i,j} \otimes \partial_j(x)) \\ &= \partial_j[Hom(P,Q)](f_{p,q})(x)) + (-1)^n f_{p,q}(\partial_j[P](x)) \\ &= \partial_j[Q](f_{p,q}(x)) \end{aligned}$$

□

4.2.2 $*$-autonomous structures

Finite HDA

Let (1) be the HDA generated by a state 1, and with null boundary operators.

Definition 28 *Let M be a free HDA ($M \in \Upsilon_F$). Then the HDA $M^* = Hom(M,(1))$ is called the dual of M. Elements of M^* are called functionals.*

We now choose for M a basis B. The definitions involving B will be non-canonical, but in most applications, the basis comes before the module M (for instance when $M = \underline{N}$, N semi-regular HDA). For x in B, we write x^* for the functional such that $x^*(x) = 1$ and $\forall y \in B,\ y \neq x,\ x^*(y) = 0$.
This extends to any element $x = \sum_i \lambda_i x_i$, with $x_i \in B$ by letting $x^* = \sum_i \lambda_i x_i^*$.
Let now $\leq_B$ or simply $\leq$ be the reflexive and transitive relation on M generated by the relations:
for $x, y \in B\ x^*(\partial_0(y)) \neq 0 \Longrightarrow x \leq y$
for $x, y \in B\ y^*(\partial_1(x)) \neq 0 \Longrightarrow x \leq y$
We say that transition a comes before transition b if $a \leq b$. All paths p verify $i < j \Rightarrow p_i \leq p_j$. The next lemma shows that acyclic corresponds to our intuition: the flow of paths on an acyclic automaton is partially ordered.

[4] Or $eval(\{f_{i,j}\}_{i,j} \otimes x)$.

Lemma 12 *If M is acyclic then $\leq$ is a partial order.*

PROOF. We have to verify that $\leq$ is antisymmetric. Take a, b such that $a \leq b$ and $b \leq a$. Then we have five cases,

- $b^*(\partial_0(a)) \neq 0$ and $a^*(\partial_0(b)) \neq 0$. Suppose $a \in M_{p,q}$, then $\partial_0(a) \in M_{p-1,q}$. If $b \in M_{r,s}$, $b^* \in M_{-r,-s}$. Then $b^*(\partial_0(a)) \neq 0$ implies that $b^* \in M_{-(p-1),-q}$ so $r = p-1$ and $s = p$. Similarly we can prove that $\partial_0(a) \in M_{r-1,s}$. Therefore, $p = r-1$ and $q = s$ which is impossible since $r = p-1$ and $s = p$.
- $b^*(\partial_0(a)) \neq 0$ and $b^*(\partial_1(a)) \neq 0$. Similar kind of contradiction.
- $a^*(\partial_1(b)) \neq 0$ and $a^*(\partial_0(b)) \neq 0$. Contradiction again.
- $a^*(\partial_1(b)) \neq 0$ and $b^*(\partial_1(a)) \neq 0$. Contradiction.
- $a = b$. This is the only possibility.

□

More generally, we can define a bilinear product $< \cdot, \cdot >$ on M by $< x, y >= y^*(x)$. We come now to the description of the dual of an HDA.

Lemma 13 *Let M be a finite state automaton with basis B. Then M^* has basis B^*, $dim\ b^* = -dim\ b$. Moreover, if the boundary operators of M^* are denoted by ∂_0^* and ∂_1^*, then:*

$$\forall x, y \in B, < \partial_i(x), y >=< x^*, \partial_i^*(y^*) >$$

PROOF. The first part of the lemma is a well-known fact from module theory. Then, b^* is a function which only sends an element of dimension $dim\ b$, b, to an element of dimension 0, namely 1. Thus b^* is of dimension $-dim\ b$.
Now, $\partial_i^* = Hom(\partial_i, (1))$, that is, $\partial_i^*(f^*) = (-1)^{(dim f-1)} f^* \circ \partial_i$, because $\partial_i[(1)] = 0$. Thus, $< x^*, \partial_i^*(y^*) >= (\partial_i^*(y^*))^*(x^*) = (-1)^{(dim\ y-1)}(y^* \circ \partial_i)^*(x^*)$. But, $\exists(\alpha_i)_i$, $y^* \circ \partial_i = \alpha_0 x^* + \alpha_1 b_1^* + ... + \alpha_n b_n^*$. So $< \partial_i(x), y >= y^* \circ \partial_i(x) = \alpha_0$. We have also $(y^* \circ \partial_i)^* = \alpha_0 x^{**} + \alpha_1 b_1^{**} + ... + \alpha_n b_n^{**}$. Thus, $(y^* \circ \partial_i)^*(x^*) = \alpha_0$. □

The operators ∂_0^* and ∂_1^* are respectively called the *end boundary* operator and the *start boundary* operator. By analogy with [Pra92], the dual of transitions, which bear information and change time, are events, which bear time and change information. Thus, the (standard) boundary operators describe the temporal beginning and ending of events, whereas the dual boundary operators describe the information we have at the source point, and the information we have at the target point. Notice that *eval* acts on events like *sync* acts on events in CML [Rep92]: it synchronizes the event with its argument.

Example 17 *The dual of example (1) is:*

$$\begin{array}{ccc} M_{0,1} = 0 & \xrightarrow{\partial_0} & M_{-1,1} = (1^*) \\ \partial_1 \downarrow & & \partial_1 \downarrow \\ M_{0,0} = (\alpha^*) & \xrightarrow{\partial_0} & M_{-1,0} = (a^*) \end{array}$$

In this HDA, we have one 1-event namely a^*, "waiting for transition a to be fired". The "information" beginning of 1^* is a^*, the "information" end of α^* is a^*.

Lemma 14 *For M a finite state automaton, M and M^{**} are isomorphic.*

PROOF. M, M^* are isomorphic as R-modules, then M and M^{**} are also isomorphic as R-modules. Now, $\forall x, y \in B$, $< \partial_i(x), y >=< x^*, \partial_i(y^*) >=< \partial_i(x^{**}), y^{**} >$, and as B^{**} is a basis of M^{**}, these relations imply that $\partial_i(x)$ and $\partial_i(x^{**})$ decompose the same way, with the same coefficients on B (resp. B^{**}). Therefore $(\cdot)^{**}$ is an isomorphism of HDA. □

Let Υ_f be the full sub-category of Υ_F composed of finite free HDA (in the sense HDA which are modules of finite dimension). We have,

Proposition 9 *Υ_f is linear $*$-autonomous. Moreover, it is compact-closed.*

PROOF. Let $\perp$ be any HDA isomorphic to the base ring R. For instance, $\perp = (1)$ with $\partial_0(1) = \partial_1(1) = 0$. Let $(\cdot)^*$ be the contravariant functor $Hom(\cdot, \perp)$. We show that it is a dualizing functor. We already know that there exists a natural isomorphism $u : Id \cong (\cdot)^{**}$. Now, we have to verify that the following diagram commutes,

$$\begin{array}{ccc} Hom(A,B) & \xrightarrow{(\cdot)^*} & Hom(A^*, B^*) \\ & u \circ \cdot \circ u^{-1} \searrow & \downarrow (\cdot)^* \\ & & Hom(A^{**}, B^{**}) \end{array}$$

This is easy verification. It is also linear since Υ, hence Υ_f is cartesian.

Now, we want to show that $\parr = \otimes$ (compact-closedness), i.e. for all A and B in Υ_f, $A^\perp \otimes B^\perp \cong (A \otimes B)^\perp$.

We begin by showing that $A^\perp \otimes B^\perp \cong (B \otimes A)^\perp$ as HDA. The result will be entailed by the fact that $A \otimes B$ and $B \otimes A$ are isomorphic. If we look at $(A^\perp \otimes B^\perp)_{p,q}$ and $(B \otimes A)^\perp_{p,q}$, we see that they are isomorphic as modules.

The isomorphism as modules is $u(x^\perp \otimes y^\perp) = (y \otimes x)^\perp$. We have now to verify that it is a morphism in Υ_f.

$$\partial_i(u(x^\perp \otimes y^\perp)) \quad = \quad -(-1)^{dim\ x + dim\ y}(y \otimes x)^\perp \circ \partial_i$$

and,

$$\partial_i(x^\perp \otimes y^\perp) \quad = \quad -(-1)^{dim\ x} x^\perp \circ \partial_i \otimes y^\perp - (-1)^{dim\ x + dim\ y} x^\perp \otimes y^\perp \circ \partial_i$$

In order to compare these equations, we need to write $x^\perp \circ \partial_i$ as some functional. First, it is easy to show that for all HDA M bounded above and below, there exists a basis B of M stable under the application of ∂_i, i.e., $(b \in B) \Rightarrow (\partial_i(b) \in B$ or $\partial_i(b) = 0)$. This can be proved by choosing first a basis B_0 for the sub-module of objects of maximal degree of M, and then construct inductively the bases B_n of the sub-modules of objects of M of lesser degree by completing $\partial_0(B_{n-1}) \cup \partial_1(B_{n-1})$.

Then, as finite HDA are bounded above and below, we can choose bases $\mathcal{A}$ and $\mathcal{B}$ for HDA A and B verifying the previous property. Suppose that $x \in \mathcal{A}$ and $y \in \mathcal{B}$.

Now, the equation $x^\perp \circ \partial_i(a) = 1$ has as solution an affine space whose origins are the a such that $\partial_i(a) = x$. Therefore, $x^\perp \circ \partial_i$ can be decomposed on the basis $\mathcal{A}^\perp$ of $A^\perp$ as,

$$x^\perp \circ \partial_i = \sum_{a/\partial_i(a)=x} a^\perp$$

Similarly,

$$y^\perp \circ \partial_i = \sum_{b/\partial_i(b)=y} b^\perp$$

Suppose now that $a \in \mathcal{A}$, $b \in \mathcal{B}$, then the equation $(y \otimes x)^\perp \circ \partial_i(b \otimes a) = 1$ is equivalent to $\partial_i(b) \otimes a + (-1)^{dim\ b} b \otimes \partial_i(a) = y \otimes x$. All summands are members of the basis $\mathcal{B} \otimes \mathcal{A}$ of $B \otimes A$. Therefore, its solutions are generated by a and b with $\partial_i(b) = y$ and $a = x$ or $b = y$ and $\partial_i(a) = (-1)^{dim\ y} x$ Thus,

$$\partial_i(u(x^\perp \otimes y^\perp)) = -(-1)^{dim\ x + dim\ y} \left((-1)^{dim\ y} \sum_{a/\partial_i(a)=x} (y \otimes a)^\perp + \sum_{b/\partial_i(b)=y} (b \otimes x)^\perp \right)$$

and,

$$u(\partial_i(x^\perp \otimes y^\perp) = -(-1)^{dim\ x} \sum_{a/\partial_i(a)=x} (y \otimes a)^\perp - (-1)^{dim\ x + dim\ y} \sum_{b/\partial_i(b)=y} (b \otimes x)^\perp$$

are equal. This proves the proposition. □

Infinite HDA

We will not develop the case of infinite HDA here. The problem is to extend the duality from finite HDA to at least some infinite HDA. This can be done along the lines of [Bar85] by adding topologies to HDA or using Chu's construction (which has recently gained some importance in the computer scientific community, [Pra94a, Pra94b]). Then we get a $*$-autonomous category of finite and infinite HDA in which $\parr \neq \otimes$. This is left for future work.

4.3 A formal comparison of the HDA-based models

4.3.1 Semi-regular and general HDA

In this chapter and in Chapter 2 we have studied the categorical structure of Υ_{sr} and of Υ quite extensively. It is clear that many of the categorical constructors correspond in one category and in the other. We conclude that Υ is an abstraction of Υ_{sr}. Intuitively, we lose in Υ the combinatorics in the construction of HDA that we had in Υ_{sr} through, for instance, the ordering of boundaries.

First, we have to make precise the relation between the mono-index notation for transitions, and the bi-index one. Let Υ^2 be the category whose objects M are sequences $(M_{p,q})_{(p,q)\in\mathbb{Z}\times\mathbb{Z}}$ with boundary operators

$$d^0_i : M_{p,q} \to M_{p-1,q}$$

$$d^1_i : M_{p,q} \to M_{p,q-1}$$

$(0 \leq i, j \leq p+q-1)$ with $d^k_i d^l_j = d^l_{j-1} d^k_i$ (for $i < j$) and whose morphisms are $f = (f_{p,q})_{p,q}$ with $(0 \leq i \leq p+q-1)$

$$f_{p-1,q} \circ d^0_i = d^0_i \circ f_{p,q}$$

$$f_{p,q-1} \circ d^1_i = d^1_i \circ f_{p,q}$$

(the category of "bi-indexed semi-regular HDA")

Then we define a functor

$$B : \Upsilon_{sr} \to \Upsilon^2$$

as follows, where M and N are semi-regular HDA and $f : M \to N$ is a morphism of semi-regular HDA.

- $B(M)_{p,q} = M_{p+q}$,
- $B(d^k_i) = d^k_i$,
- $B(f)_{p,q} = f_{p+q}$

Functor B makes all automata acyclic.

Now we have to abstract away from the combinatorics of boundaries.

In Section 2.2.4, we had defined a functor $\mathcal{A} : \Upsilon^2 \to \Upsilon$ as follows,

$$\mathcal{A}(M) = \begin{array}{ccc} \mathcal{A}(M_{p,q}) & \xrightarrow{\partial_0} & \mathcal{A}(M_{p-1,q}) \ldots \\ \Big\downarrow{\scriptstyle \partial_1} & & \vdots \\ \mathcal{A}(M_{p,q-1}) \ \ldots & & \end{array}$$

such that $\mathcal{A}(M_{p,q})$ is the free module generated by $M_{p,q}$ and,

$$\partial_0 = \sum_{i=0}^{p+q-1} (-1)^i d_i^0$$

$$\partial_1 = \sum_{i=0}^{p+q-1} (-1)^i d_i^1$$

Then,

Lemma 15 *$\mathcal{A} \circ B : \Upsilon_{sr} \to \Upsilon$ commutes with all colimits.*

PROOF. We verify the commutation with finite colimits first, i.e. with coproducts and coequalizers,

$$\begin{aligned} \mathcal{A} \circ B(M \coprod N)_{p,q} &= R - Mod(M_{p+q} \cup N_{p+q}) \\ &= \mathcal{A} \circ B(M)_{p,q} \oplus \mathcal{A} \circ B(N)_{p,q} \end{aligned}$$

Then for coequalizers, let

$$M \underset{g}{\overset{f}{\rightrightarrows}} N \xrightarrow{h} coequ(f,g) = P$$

be the coequalizer diagram in Υ_{sr} for f, g. Then we can check that $P_n = N_n / \{f(x) = g(x) / x \in M_n\}$. Then,

$$\begin{aligned} \mathcal{A} \circ B(P)_{p,q} &= R - Mod(P_{p+q}) \\ &= R - Mod(N_{p+q}) / Im\,(\mathcal{A} \circ B(f) - \mathcal{A} \circ B(g)) \end{aligned}$$

which is seen to be equal to the coequalizer of $\mathcal{A} \circ B(f)$ and $\mathcal{A} \circ B(g)$ in Υ. The induced boundary operators are the boundary operators of the coproduct and coequalizers in Υ respectively.

Finally, we check that $\mathcal{A} \circ B$ commutes with direct limits. We do not check this directly but rather use the fact that the free R-module functor from **Set** to $R - Mod$ commutes with direct limits since it has the forgetful functor as right-adjoint. This entails that $\mathcal{A} \circ B(\varinjlim M_i) = \varinjlim \mathcal{A} \circ B(M_i)$. □

Notice that $\mathcal{A} \circ B$ does not commute with the cartesian product since in Υ we have biproducts whereas $\times \neq \oplus$ in Υ_{sr}.

By Proposition II.2.4 of [GZ67] and Lemma 15 we have the existence of $D : \Upsilon \to \Upsilon_{sr}$ right-adjoint to $\mathcal{A} \circ B$.

Note that the tensor product is also preserved via this adjunction.

4.3.2 Regular and general HDA

A very similar adjunction exists from regular to general HDA. The proofs go along the same lines as in previous section. An other way to construct the adjunction could actually use the category of combinatorial HDA of Chapter 8 mimicking the classical equivalence between complexes of modules and simplicial modules [May67], but this goes beyond the scope of this thesis (look at Chapter 10 though).

4.3.3 Semi-regular and regular HDA

Regular HDA are shown to be a straightforward abstraction of semi-regular HDA where we have the ability to speak about finite sets of transitions (and even finite linear combinations of transitions). The pair of adjoint functors relating the two models is based on the classical pair $(R-Mod, Forget)$ (see [ML71]) between **Set** and $R-Mod$.

Lemma 16 *$(R-Mod \circ B, C \circ Forget)$ is a pair of adjoint functors between Υ_{sr} and Υ_r (where C is the right-adjoint of B).*

PROOF. This is due to the fact that $(R-Mod, Forget)$ is a pair of adjoint functors [ML71] and (B, C) is a pair of adjoint functors as well (Section 4.3.1). □

We can verify that the pair of adjoint functors in Section 4.3.2 composed with the pair of adjoint functors of Section 4.3.3 is the same as the pair of adjoint functors in Section 4.3.1.

We can also generalize the $(R-Mod, Forget)$ pair of adjoint functor for having a pair of adjoint functors between $\underset{\sim}{\mathbf{Set}}$ and $R-Mod$ as follows (X and Y are sets and $f: X \to Y$ is a partial function),

- $R-Mod(X) = R-Mod(X)$,
- $R-Mod(f)(x) = \begin{cases} 0 & \text{if } f(x) \text{ is undefined} \\ f(x) \in R-Mod(Y) & \text{otherwise} \end{cases}$.

Then, it is straightforward to see that $(R-Mod \circ B, C \circ Forget)$ is a pair of adjoint functors between $CP\Upsilon$ and Υ_r.

The case of non-closed partial HDA is unclear at this point.

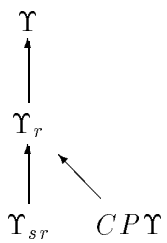
4.3.4 The lattice of HDA

We sum up the results of this chapter on formal relationships between all HDA-based models. We write $A \to B$ for categories A and B when there is a pair of adjoint functors

$$A \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} B$$

where α is left-adjoint to γ.

Then we have the following diagram,



Summary In this chapter, we have studied the categorical structure of general HDA. Υ and Υ_a were shown to be complete, co-complete autonomous categories, leading to an interpretation of intuitionistic linear logic. We proved that the category of HDA with a finite number of transitions and states formed a $*$-autonomous category, the dual of a transition being an event "waiting for the action to be over". Finally, it was formally shown that general HDA are more abstract than regular and semi-regular HDA (again in the style of [WN94]).

Chapter 5

Introduction to semantic domains of HDA

5.1 Basic principles

In this section, we wish to give a compositional (or denotational) semantics for some parallel languages, with denotations being HDA. That is, we want to describe the "higher-dimensional" traces of programs, generalizing the denotational semantics using TS, PN etc. (see Chapter 1). In some way, we would like to benefit from the advantages of both worlds of denotational and operational semantics: inductive definition and proofs of programs, and power of expression of behaviour of programs.

We first have to define what we mean by "domain" of HDA. A domain of HDA is used for giving a denotational semantics of some language $\mathcal{L}$. It should then "contain" all the execution traces of programs of $\mathcal{L}$. But to be interesting for studying the dynamics of traces, it should certainly not just be the collection of all these traces. It should be a HDA D such that all execution traces (partial or not) are subHDA of D. For instance, if D is the filled in square with edges a, b and a', b', then the possible traces of the programs written in the corresponding language are the paths described in Example 7 of Chapter 2.

Of course this is not a very interesting example. In most cases the operational semantics of $\mathcal{L}$ can be given if we have a representation as HDA of some "ground" commands (like assignments for instance), or just some atomic actions as it will be the case in the last section with a CCS-like process algebra. Then we just need to specify that if we have two traces, then we certainly have the parallel execution of these two traces. This is conveniently described as a recursive domain equation which abstract away from the actual construction of all needed transitions like,

$$D \cong (atomic\ actions) + D \otimes D$$

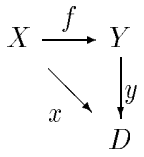
We study in Section 5.3 the mathematical meaning of such an equation, before giving an example of the definition of the semantics of a toy language. But first

of all, we relate our domains of HDA to domains used in denotational semantics [Plo84].

5.2 Domains of HDA in order-theoretic form

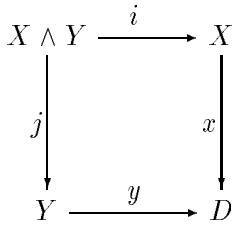
Let D be a HDA.

The category Sub of subobjects of D is a subcategory of the slice category Υ/D composed of monomorphims $x : X \hookrightarrow D$ modulo isomorphism, i.e. $x : X \hookrightarrow D$ and $y : Y \hookrightarrow D$ are identified if there exists an isomorphism $f : X \to Y$ such that the following diagram is commutative,



As Υ is complete and co-complete, Sub is actually a complete lattice with the following operations,

- Intersection $X \wedge Y$ of subobjects X and Y of D is the pullback of the corresponding morphisms,

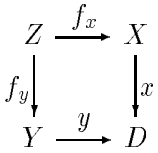


- Union $X \vee Y$ is the pushout of $X \wedge Y \xrightarrow{i} X$ and $X \wedge Y \xrightarrow{j} Y$

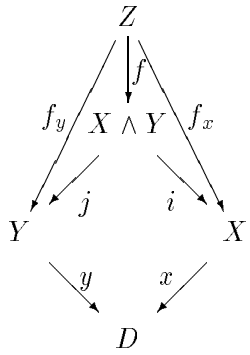
- The order to which these lattice operations correspond is $(X \xrightarrow{x} D) \leq (Y \xrightarrow{y} D)$ iff there is a monomorphism $f : X \to Y$ such that $y \circ f = x$.

Proof. The pullback of a monomorphism by any morphism is a monomorphism. The composite of two monomorphisms is a monomorphism, hence $x \circ i = y \circ j$ is a monomorphism and defines $X \wedge Y$ as a subobject of D.

Moreover, if $z : Z \to D$ is such that $z \leq x$ and $z \leq y$ then there exist f_x and f_y such that the following diagram is commutative,

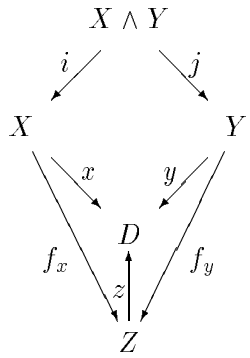


By the universal property of pullbacks, there exists $f : Z \to X \wedge Y$ such that the following diagram commutes,



This proves that $z \leq (x \wedge y)$, hence that $x \wedge y$ is indeed the greatest lower bound.

For the second part, notice the following. If $z : Z \to D$ is such that $x \leq z$ and $y \leq z$ then there exist $f_x : X \to Z$ and $f_y : Y \to Z$ such that the following diagram commutes,



Moreover, by the universal property of the pushout, there exists a unique $f : X \vee Y \to Z$ such that $f \circ in_r = f_y$ and $f \circ in_l = f_x$. There exists also a unique $u : X \vee Y \to D$ such that $u \circ in_r = y$ and $u \circ in_l = x$. u actually defines the union of x and y since as (see the diagram above) $z \circ f_x = x$ and $z \circ f_y = y$, we have $z \circ f \circ in_r = z \circ f_y = y$ and $z \circ f \circ in_l = x$, hence by unicity, $u = z \circ f$ (this holds for all z and derived f). Therefore $u \leq z$, $x \leq u$ and $y \leq u$, thus $u = x \vee y$. □

This is valid for Υ, Υ_r, $P\Upsilon$, and Υ_{sr} since all these categories have pullbacks and pushouts. In the case of Υ_{sr} we can be slightly more precise since we have seen that Υ_{sr} is an elementary topos. This implies that Sub is a Heyting algebra (Brouwerian lattice), i.e. a residuated lattice with bottom and top elements.

5.3 Recursive domain equations

(*) Lemma 7 *(see [AL91b]) Let F be a ω-continuous functor from Υ to Υ. Then,*

$$\exists D \in \Upsilon, \quad D \equiv F(D)$$

(*) PROOF. Consider the following sequence of objects and morphisms in Υ, indexed by $i \in \mathbb{N}$:

- $D_0 = F(0)$
- $D_{i+1} = F(D_i)$
- $j_0 : D_0 \longrightarrow D_1, \quad j_0 = 0$
- $j_{i+1} : D_{i+1} \longrightarrow D_{i+2}, \quad j_{i+1} = F(j_i)$

Then, for $i \leq k$, define $M_{i,k} : D_i \longrightarrow D_k$ by $M_{i,k} = j_{k-1} \circ ... \circ j_i$. $\mathcal{D} = (D_i, M_{i,j})_{i \geq 0, i \leq j}$ is a direct system in Υ. Let $(D, p_i)_{i \geq 0} = \underrightarrow{lim}\ \mathcal{D}$. Consider the direct system $\mathcal{D}' = (D_i, M_{i,j})_{i>0, i \leq j}$. $(D, p_i)_{i>0}$ is a cone for $\mathcal{D}'$. Let $(E, p'_i)_{i>0}$ be another cone for $\mathcal{D}'$, then let $p'_0 = p'_1 \circ M_{0,1}$; $(E, p'_i)_{i \geq 0}$ is a cone for $\mathcal{D}$, thus there exists $h : D \longrightarrow E$ such that $p'_i = h \circ p_i$, $i \geq 0$. Thus $(D, p_i)_{i>0}$ is also universal, and $\underrightarrow{lim}\ \mathcal{D}' = D$.
But F is a map of direct systems between $\mathcal{D}$ and $\mathcal{D}'$. F is ω-continuous, so $F(\underrightarrow{lim}\ \mathcal{D}) = \underrightarrow{lim}\ \mathcal{D}'$, that is, $F(D) \equiv D$. □

All this also gives us a means to label HDA, just knowing the labels of the "atomic actions". Let (M, l) be a labeled HDA over L. Consider now the equation $D = M + G(D)$, where G is ω-continuous in each argument. Since $(M + \cdot)$ is ω-continuous, Lemma 7 guarantees the existence of a solution to this equation. Let now D_L be the HDA verifying the equation $D_L = L + G(D_L)$. Consider the direct systems $\mathcal{D}$ and $\mathcal{L}$ induced respectively by the first and second recursive equations. We define a map u of direct systems from $\mathcal{D}$ to $\mathcal{L}$ by $u = (l + Id_{G(0)}, l + G(l + Id_{G(0)}), l + G(l + G(l + Id_{G(0)})), \ldots)$. Let $l' : D \to D_L$ be the direct limit of u and $in_1 : M \to D$, $in'_1 : L \to D_L$ be respectively the canonical monomorphism from M to D and from L to D_L. By definition of the direct limit, the diagram,

$$\begin{array}{ccc} M & \xrightarrow{in_1} & D \\ {\scriptstyle l}\downarrow & & \downarrow{\scriptstyle l'} \\ L & \xrightarrow{in'_1} & D_L \end{array}$$

commutes. Therefore, l' is a labelling of D which extends l.

Example 18 *Let D be an HDA verifying $D = M + D \otimes D$[1] where M is the HDA:*

$$M_0 = (1) \oplus (\alpha) \oplus (\alpha') \oplus (\beta) \oplus (\beta')$$
$$M_1 = (a) \oplus (a') \oplus (b) \oplus (b')$$

with $\partial_0(a) = \partial_0(a') = \partial_0(b) = \partial_0(b') = 1$ and $\partial_1(a) = \alpha$, $\partial_1(a') = \alpha'$, $\partial_1(b) = \beta$, $\partial_1(b') = \beta'$. Let L be the one defined in the previous example. M can be labeled over L by $l(a) = l(a') = \mathbf{a}$, $l(b) = l(b') = \mathbf{b}$, $l(1) = l(\alpha) = l(\alpha') = l(\beta) = l(\beta') = \mathbf{1}$. Then it extends to a labelling of D over D_L. For instance, $l(a \otimes \beta) = \mathbf{a}$, or $l(\alpha' \otimes a \otimes b) = \mathbf{a} \otimes \mathbf{b}$.

[1] The case of bifunctors covariant and ω-continuous in each argument is no more difficult than what we have just seen.

5.4 Example: A CCS-like Language

As an example, we give the semantics of a CCS-like process algebra,

Terms are built on actions which are elements of a set $\Sigma = \{a^j / j \in K\}$ and **nil**, with operators . (sequential composition), + (choice operator), | (parallel composition), $a \to \overline{a}$ (complementary action), **rec** (recursion operator) and \ (restriction operator). For a better explanation, we have divided the parallel operator in two operators: $\|$, which is parallel composition without communication, and the general one |.

Terms are then formed according to the following grammar:

t	::=	a	(single action)
	|	$\overline{a}$	(complementary action)
	|	**nil**	(idle process)
	|	$t_1 + t_2$	(choice operator)
	|	$t_1 \| \| t_2$	(parallel composition-no communication)
	|	$t_1 \mid t_2$	(parallel composition-communication)
	|	$t_1 . t_2$	(sequential composition)
	|	$t_1 \backslash c$	restriction operator
	|	**rec** $x.t(x)$	recursive agent

5.4.1 Semantics using semi-regular HDA

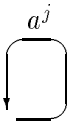
Denotational semantics

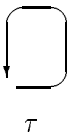
We first construct the domain we need. Here we give a denotational semantics where denotations are operational behaviours[2]. Let (a_i^j), $(\overline{a}_i^j)$, $(\tau_{i,j}^k)$ $(i, j \in \mathbb{N}, k \in K)$, (a^j), $(\overline{a}^j)$ and (τ) be the following HDA (informally or geometrically),

(a_i^j) : $1 \xrightarrow{a_i^j} \alpha_i^j$

$(\overline{a}_i^j)$: $1 \xrightarrow{\overline{a}_i^j} (\overline{\alpha}_i^j)$

$(\tau_{i,j}^k)$: $1 \xrightarrow{\tau_{i,j}^k} \alpha_i^k \otimes \overline{\alpha}_j^k$

(a^j) : 

$(\overline{a}^j)$: 

(τ) : 

We suppose 1 to be a neutral element for the tensor product (this is formally realized by some suitable quotient construction).

[2] We could have given one in a more classical form like input-ouput relations or history of communications.

Let P and L be the domains given by the recursive equations,

$$P \cong (a_i^j)_{i,j} + (\overline{a}_i^j)_{i,j} + (\tau_{i,j}^k)_{i,j,k} + P \otimes P$$

$$L \cong (a^j) + (\overline{a}^j) + (\tau) + L \otimes L$$

Let $l : P \to L$ be the morphism of HDA defined by,

- $\forall (i,j) \in \mathbb{N} \times K,\ l(a_i^j) = a^j$
- $\forall (i,j) \in \mathbb{N} \times K,\ l(\overline{a}_i^j) = \overline{a}^j$
- $\forall (i,j,k) \in \mathbb{N} \times \mathbb{N} \times K,\ l(\tau_{i,j}^k) = \tau$
- $\forall (x,y) \in P \times P,\ l(x \otimes y) = l(x) \otimes l(y)$

We introduce a new operator $\otimes_c$ for dealing with synchronization. We define, for P and Q HDA, $P \otimes_c Q$ by,

$$u \in (P \otimes_c Q)_n \Leftrightarrow \left\{ \begin{array}{l} u = x \otimes y, \quad \left\{ \begin{array}{c} x \in P_m, y \in Q_{n-m}, \\ l(x) \text{ and } l(y) \text{ contain} \\ \text{no complementary actions} \end{array} \right. \\ \\ u = x \otimes \left(\underset{(k,i)\in T_1,(k,j)\in T_2}{\otimes} \tau_{i,j}^k \right) \otimes y, \quad \left\{ \begin{array}{c} x \otimes \left(\underset{(j,i)\in T_1}{\otimes} a_i^j \right) \in P_m \\ \left(\underset{(j,i)\in T_2}{\otimes} \overline{a}_i^j \right) \otimes y \in Q_{n+card(T_1)-m} \\ card(T_1) = card(T_2) \end{array} \right. \end{array} \right.$$

Notice that this view and the introduction of the $\tau_{i,j}^k$ in the domain D is much alike the synchronization algebras used in [Win88].

The domain of HDA in which we give the semantics of the language is $D = l : P \to L$. We can actually give it in $D = P$, and recover the full definition by applying the labelling l.

We use the notation (x) to denote the subHDA of D "generated" by $x \in D$. This means it is the smallest HDA contained in D and containing x. Then,

- $[\![\mathbf{nil}]\!] = (1)$
- $[\![a^j]\!] = (a_i^j)$ for some fresh i
- $[\![\overline{a}^k]\!] = (\overline{a}_j^k)$ for some fresh j
- $[\![p + q]\!] = [\![p]\!] \coprod [\![q]\!]$ ($\coprod$ is the coproduct in Υ_{sr}/D, it corresponds to an amalgamated sum in Υ_{sr})
- $[\![p.q]\!] = [\![p]\!] \coprod f \otimes [\![q]\!]$ where f is the subHDA of final states of $[\![p]\!]$ defined as $f = \{s \in [\![p]\!]_0 / \forall t \in [\![p]\!]_1, d_0^0(t) \neq s\}$
- $[\![p \| q]\!] = [\![p]\!] \otimes [\![q]\!]$

- $[\![p \mid q]\!] = [\![p]\!] \otimes_c [\![q]\!]$
- $[\![p\backslash a]\!] = [\![p]\!]\backslash\{x \otimes a_i^j \otimes y, x \otimes \overline{a}_i^j \otimes y\}$
- $[\![\mathbf{rec}\ x.p[x]]\!] = \overrightarrow{lim}\ [\![p^i[\mathbf{nil}]]\!]$ where the direct limit is taken on the full subcategory of Υ_{sr}/D whose objects are the $[\![p^i[\mathbf{nil}]]\!]$

Operational semantics

Its operational semantics is then (by results of Section 2.2.1),

$$\frac{}{\mathbf{nil} \models 1 \xrightarrow{1} 1}$$

$$\frac{}{a^j \models 1 \xrightarrow{a^j} \alpha_i^j} \qquad \frac{}{\overline{a}^j \models 1 \xrightarrow{\overline{a}^j} \overline{\alpha}_i^j}$$

$$\frac{Q \models s \xrightarrow{a} t}{Q + Q' \models s \xrightarrow{a} t}$$

$$\frac{Q' \models s' \xrightarrow{a'} t'}{Q + Q' \models s' \xrightarrow{a'} t'}$$

$$\frac{Q \models s \xrightarrow{a} t}{Q.Q' \models s \xrightarrow{a} t}$$

$$\frac{Q' \models s' \xrightarrow{a'} t' \qquad f \in \{s \in Q_0 / \neg\exists t \in Q_1, d_0^0(t) = s\}}{Q.Q' \models f \otimes s' \xrightarrow{a'} f \otimes t'}$$

$$\frac{Q \models s \xrightarrow{a} t \qquad Q' \models s' \xrightarrow{a'} t'}{Q \| Q' \models s \otimes s' \xrightarrow{a \otimes a'} t \otimes t'}$$

$$\frac{Q \models s \xrightarrow{a} t \qquad Q' \models s' \xrightarrow{a'} t' \quad a \neq \overline{a'}}{Q \mid Q' \models s \otimes s' \xrightarrow{a \otimes a'} t \otimes t'}$$

$$\frac{Q \models s \xrightarrow{x \otimes a \otimes y} t \qquad Q' \models s' \xrightarrow{z \otimes \overline{a} \otimes u} t'}{Q \mid Q' \models s \otimes s' \xrightarrow{x \otimes z \otimes \tau \otimes y \otimes u} t \otimes t'}$$

$$\frac{Q[\mathbf{rec}\ x.Q[x]] \models s \xrightarrow{u} t}{\mathbf{rec}\ x.Q[x] \models s \xrightarrow{u} t}$$

The last two rules have not been shown in Section 2.2.1. They are easy to derive. The last one expresses that $[\![\mathbf{rec}\ x.Q[x]]\!]$ forms a co-cone for the diagram $([\![Q^i[\mathbf{nil}]]\!])_i$.

Last but no least, we have a correctness result,

Proposition 10 *The truncation at dimension one $T_1([\![p]\!])$ is bisimulation equivalent to the interleaving semantics of the CCS term p (as defined in [Mil89]).*

SKETCH OF PROOF. The rules when restricted to 1-transitions become,

(1)
$$\frac{}{a^j \models 1 \xrightarrow{a^j} \alpha_i^j} \qquad \frac{}{\overline{a}^j \models 1 \xrightarrow{\overline{a}^j} \overline{\alpha}_i^j}$$

(2)
$$\frac{Q \models s \xrightarrow{a} t}{Q + Q' \models s \xrightarrow{a} t}$$

(3)
$$\frac{Q' \models s' \xrightarrow{a'} t'}{Q + Q' \models s' \xrightarrow{a'} t'}$$

(4)
$$\frac{Q \models s \xrightarrow{a} t}{Q.Q' \models s \xrightarrow{a} t}$$

(5)
$$\frac{Q' \models s' \xrightarrow{a'} t' \qquad f \in \{s \in Q_0 / \neg \exists t \in Q_1, d_0^0(t) = s\}}{Q.Q' \models f \otimes s' \xrightarrow{a'} f \otimes t'}$$

(6)
$$\frac{Q \models s \xrightarrow{a} t \qquad Q' \models s' \xrightarrow{a'} t'}{\begin{array}{l} Q\|Q' \models s \otimes s' \xrightarrow{a \otimes s'} t \otimes s' \\ Q\|Q' \models s \otimes s' \xrightarrow{s \otimes a'} s \otimes t' \\ Q\|Q' \models s \otimes t' \xrightarrow{a \otimes t'} t \otimes t' \\ Q\|Q' \models t \otimes s' \xrightarrow{t \otimes a'} t \otimes t' \end{array}}$$

(7)
$$\frac{Q \models s \xrightarrow{a} t \qquad Q' \models s' \xrightarrow{a'} t' \quad a \neq \overline{a'}}{\begin{array}{l} Q\|Q' \models s \otimes s' \xrightarrow{a \otimes s'} t \otimes s' \\ Q\|Q' \models s \otimes s' \xrightarrow{s \otimes a'} s \otimes t' \\ Q\|Q' \models s \otimes t' \xrightarrow{a \otimes t'} t \otimes t' \\ Q\|Q' \models t \otimes s' \xrightarrow{t \otimes a'} t \otimes t' \end{array}}$$

(8)
$$\frac{Q \models s \xrightarrow{a} t \qquad Q' \models s' \xrightarrow{\overline{a}} t'}{Q \mid Q' \models s \otimes s' \xrightarrow{\tau} t \otimes t'}$$

(9)

$$\frac{Q[\mathbf{rec}\ x.Q[x]] \models s \xrightarrow{u} t}{\mathbf{rec}\ x.Q[x] \models s \xrightarrow{u} t}$$

Rules (3) and (9) are just the same as the ones for + and **rec** in CCS. Rules (6) and (7) are similar since | and || have the same effect if applied to actions which cannot synchronize (they are not complementary). Therefore, we can restrict to terms of CCS with no **rec** nor +.

The idea for constructing a bisimulation R between the transition system $\mathcal{T}$ defining CCS and the one above $\mathcal{U}$ is that the states of $\mathcal{T}$ are the subterms yet to be executed and in $\mathcal{U}$ they are the concatenations of final states of actions already executed.

Let t be a CCS term. We define R_t as follows,

- $1 R_t t$,
- if $t \xrightarrow{a^j} t'$ and $u R_t t$ then $(u \otimes \alpha_i^j) R_t t'$ (α_i^j is the next α^j that has not been used up to this point).

We prove that R_t is the required bisimulation by induction on t. We verify just a few cases,

- $t = a^j$ and $t = \overline{a}^j$ are immediate.
- $t = a^j.q$. By hypothesis we have a bisimulation R_q between the HDA representing q and the transition system defining q. Now $1 R_t t$, $\alpha_i^j R_t q$ and $(\alpha_i^j \otimes u) R_t v$ if and only if $u R_q v$ by definition of the family of relations R_*. This obviously defines a bisimulation for t.
- $t = p \mid q$. Easy: rules (6) and (7) precisely define the interleaving of actions.

□

5.4.2 Semantics using general HDA

By the general categorical results of Chapter 4 we know that we can write in the domain of general HDA (applying functor $\mathcal{A}$ of Chapter 2),

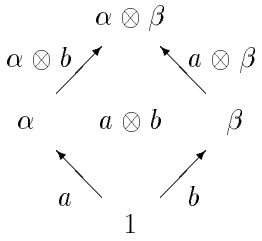
$$D \cong (a_i^j)_{i,j} + (\overline{a}_i^j)_{i,j} + (\tau_{i,j}^k)_{i,j,k} + D \otimes D$$

- $[\![\mathbf{nil}]\!] = (1)$
- $[\![a^j]\!] = (a_i^j)$
- $[\![\overline{a}^j]\!] = (\overline{a}_i^j)$

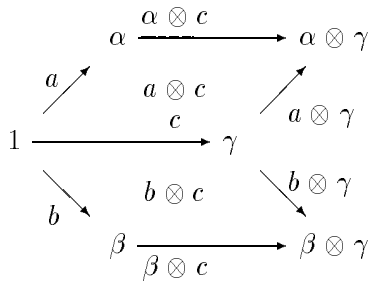
- $[\![p+q]\!] = [\![p]\!] + [\![q]\!]$
- $[\![p.q]\!] = [\![p]\!] + H_0([\![p]\!], \partial_0) \otimes [\![q]\!]$
- $[\![p \| q]\!] = [\![p]\!] \otimes [\![q]\!]$
- $[\![p \mid q]\!] = [\![p]\!] \otimes_c [\![q]\!]$ (where $\otimes_c$ is an abuse of notation for the image of $\otimes_c$ of Υ_{sr} under functor $\mathcal{A}$)
- $[\![p \backslash a^j]\!] = [\![p]\!] / \{x \otimes a_i^j \otimes y, z \otimes \overline{a}_i^j \otimes t\}$
- $[\![recx.p[x]]\!] = \overrightarrow{lim}\ [\![p^n[\mathbf{nil}]]\!]$ where the diagram is composed of all morphisms[3] of labeled HDA from $[\![p^n]\!]$ to $[\![p^{n+1}]\!]$ (for all n).

Example 19 (1) *We use the inductive construction above to translate the CCS-term $(a|b)$ (which is equal to $(a\|b)$). We have:*

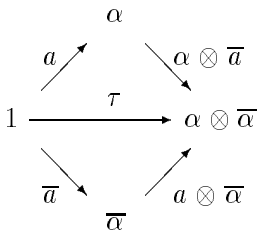
to represent $[\![a]\!]$ and $[\![b]\!]$ respectively. We now form the tensor product:



(2) *We now compute $[\![(a+b)\|c]\!]$.*

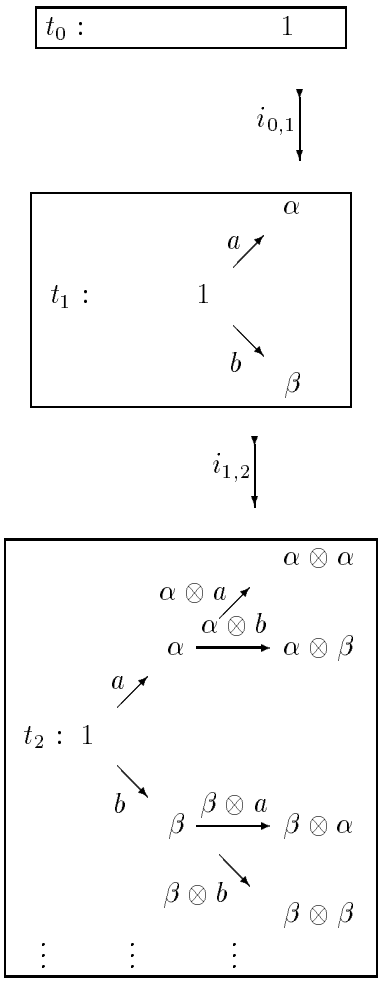


(3) *Let q=$(a \mid \overline{a})$. Then $[\![q]\!]$ is:*



[3] This could actually be refined by taking the canonical morphisms for the categorical constructions, and some well-chosen ones for the tensor product and $\otimes_c$.

(4) *We study the translation of the term* $t = recx.(a.x + b.x)$. *The development of* t *is* ($t_0 = \mathbf{nil}, t_1 = a.\mathbf{nil} + b.\mathbf{nil}, t_2 = a.(a.\mathbf{nil} + b.\mathbf{nil}) + b.(a.\mathbf{nil} + b.\mathbf{nil}), \ldots$). *Thus we have:*



The semantic value of t *is then the direct limit of the diagram above.*

We will see in Part III that if the operational semantics is more intuitive, the denotational-like gives the characterization of the geometry of the transition system.

5.5 Semantics of concurrent languages

5.5.1 Introduction

We extend here our framework in order to give semantics to concurrent languages which handle values. This is done by mimicking the ordinary denotational semantics approach [Ten91], using environments to hold values of variables. We also give the corresponding SOS approach.

First, we have to define domains for values, and define real states.

Let V be a set (of values). We write $\overline{V}$ for the HDA whose states are generated by V, and whose boundary operators are null. We have already seen an example of this construction, for $V = \{1\}$; $\overline{V}$ was written (1). This construction will be applied for sets of values like $\mathbb{N}$, or Bool=$\{tt, ff\}$. Notice that it is again a functorial construction: if f is a function between two sets V and V', then $\overline{f}$ (sometimes abbreviated by f), the linear extension of f, is a morphism of degree 0 between $\overline{V}$ and $\overline{V'}$.

The same construction can be carried out for any relation on sets of values. Consider for instance a relation $R(x, y)$ on $V \times V$. Construct a "relation" $\overline{R}$ on $\overline{V} \otimes \overline{V}$, with value in $\overline{Bool}$ by $\overline{R}(\overline{x}, \overline{y}) =_{def} \overline{R}(x \otimes y) = tt \Leftrightarrow R(x, y)$ and $\overline{R}(\overline{x}, \overline{y}) =_{def} \overline{R}(x \otimes y) = (ff) \Leftrightarrow \neg R(x, y)$.

Now, suppose we have elementary functions on these sets of values. We want to represent the application of such a function f to a value by a 1-transition between the input value to the output value. The way to do this, is to construct (when it exists in the considered domains) the $(\partial_1 - \partial_0)$-chain homotopy (see [ML63]) linking the input to the output state, that is the transition between Id and $\overline{f}$, denoted by $[Id, f]$ or $\check{f}$ ("name of f").
Suppose we have a domain D of HDA (elements of which are its sub-automata) containing 1-transitions s and t with $\partial_0(s) = 1$, $\partial_1(s) = 0$, and $\partial_0(t) = 0$, $\partial_1(t) = 1$. Let f and g be two linear maps, and define:

$$[f, g] = s \otimes f + t \otimes g$$

Then,

$$\partial_0 \circ [f, g] = f - [\partial_0 \circ f, \partial_0 \circ g] \quad \partial_1 \circ [f, g] = g - [\partial_1 \circ f, \partial_1 \circ g]$$

and when f and g are morphisms (of any degree),

$$\partial_0 \circ [f, g] + [f, g] \circ \partial_0 = f \quad \partial_1 \circ [f, g] + [f, g] \circ \partial_1 = g$$

Thus,

$$(\partial_1 - \partial_0) \circ [f, g] + [f, g] \circ (\partial_1 - \partial_0) = g - f$$

and $[f, g]$ is an $(\partial_1 - \partial_0)$-chain homotopy between f and g (we will see the meaning of these homotopies in Chapter 7). To come back to our function f, $\check{f} = [Id, f]$, then: $\partial_0(\check{f}) = Id$ and $\partial_1(\check{f}) = \overline{f}$. We have also, $\forall x \in \overline{V}$, $\partial_0(\check{f}(x)) = x$ and $\partial_1(\check{f}(x)) = \overline{f}(x)$. Hence we call $\check{f}$ "name of f", because it is the label of all applications of f.

In general, we have to use fresh copies of these t and v to build homotopies. These copies will be denoted by t_i and v_i, where i is an index (generally in $\mathbb{N}$). For f a linear function on a HDA V, we define an extension of f on tensor products of t_i, v_i and elements of V by:

$$f(t_i) = t_i \quad f(v_i) = v_i$$

$$f(x \otimes y) = f(x) \otimes f(y)$$

If f is a morphism (of degree 0) on V, then this extension defines a morphism as well. With these conventions, we have the following laws of calculus:

$$f[g,h] = [fg, fh] \quad [g,h]f = [gf, hf]$$

$$f\check{g} = [f, fg] \quad \check{g}f = [f, gf]$$

$$[f,g][k,l] = [[fk, fl], [gk, gl]]$$

The last equation shows that homotopies compose to give homotopies of higher dimension. We define also for a linear function f (not necessarily a morphism), another linear function, $\hat{f}$, called the sequentialization of f, by:

$$\hat{f}(g) = g + f \circ H_0(g, \partial_0)$$

We will see its use later on.

5.5.2 An imperative language with shared memory

Let $\mathcal{L}$ be the language (first-order imperative language - shared memory) whose syntax is defined as follows.
Let Var be a set of variable names (x, y, z...). We consider a set of values $v \in Val$, containing integers n, booleans tt and ff. We write X, Y, Z for objects which are values or variables. f is any function on Val.
The language is formed out of values v, tests t, and expressions e:

$$\begin{array}{rcl} v & ::= & x \\ & | & n \\ & | & tt \\ & | & ff \\ t & ::= & R(X,Y) \end{array} \qquad \begin{array}{rcl} e & ::= & \mathbf{nil} \\ & | & x := v \\ & | & x := h(x,v) \\ & | & e; e' \\ & | & e \mid e' \\ & | & (t \to e) \square (t' \to e') \\ & | & \mathbf{rec}\ x.q(x) \end{array}$$

where $x := h(x,v)$ is a function like $(x := v)$, $(x\times = v)$, or $(x+ = v)$ etc. That is, proceeds to an "atomic" operation on a variable. q is any syntactic expression of $\mathcal{L}$ with one hole (a context).

This language may be seen as a some subset of Concurrent Pascal, with no procedures or compound data types.

We give the semantics, considering the following domains:

- C is the HDA with C_0 generated by Var and $\partial_0 = \partial_1 = 0$,
- V is the HDA with V_0 generated by Val and $\partial_0 = \partial_1 = 0$.

We would like to have environments (i.e. assignments of values to variables) as states of our automata: the domain D to be defined should include $Env = Hom(C \oplus V, C \oplus V)$[4].

Let f be an element of Env. The elements $c \in C$ such that $f(c) = c$ are non-assigned variables. If f and g are two elements of Env, then $g \circ f$ is the assignment by f then by g (it might assign some values to some variables untouched by f, but does not change any of the assignments made by f).

We want also to have all homotopies (all transitions of any dimension) between states. This requires for D to have all tensor products between the t_i, v_i and elements of D. This leads to defining D by the equation:

$$D \equiv (t_{c,i})_{c,i} \oplus (v_{c,i})_{c,i} \oplus Hom(C \oplus V, C \oplus V) \oplus C \oplus V + (D \otimes D)$$

where c is an index lying in the set $\{x := n, x := h(x,v)/x \in Var,\ v \in Val\}$. The domain for the labelling is defined by:

$$D_L \equiv (t_c)_c \oplus (v_c)_c \oplus Hom(C \oplus V, C \oplus V) \oplus C \oplus V + (D_L \otimes D_L)$$

The labelling l is induced by $l(t_{c,i}) = t_c$, $l(v_{c,i}) = v_c$. Define now the function $[u \Leftarrow v]$ (an elementary substitution) on $C \oplus V$ by:

$$[u \Leftarrow v](u) = v, \quad [u \Leftarrow v](w) = w$$

for all $w \neq u$. Therefore, $[u \Leftarrow v] = v \otimes u^* + \sum_{w \neq u} w \otimes w^*$ It can be extended to a morphism on D as described in the previous section.

The functions h considered in $\mathcal{L}$ induce morphisms of the form h_x from $\rho \in Env$ to Env:

$$h_x(\rho, v)(x) = h(\rho(x), v) \quad h_x(\rho, v)(y) = \rho(y)$$

for all $y \neq x$. Their action is to apply the arithmetic function which h describes to the only x part of the substitution ρ. For instance, $(x := .)$ induces the morphism $[x \Leftarrow .]$. In the case of $x+ = v$, that is $h(x,v) = x + v$, we have for example, $h_x([x \Leftarrow u][y \Leftarrow w], v) = [x \Leftarrow u + v][y \Leftarrow x]$.

Then we have a semantic function $[\![\cdot]\!]$: $\mathcal{L} \to Env \to Sub(D)$ (where $Sub(D)$ is the set of all subHDA of D) given by:
for values,

$$[\![x]\!]\rho = \rho(x) \tag{5.1}$$

$$[\![n]\!]\rho = (n) \tag{5.2}$$

$$[\![tt]\!]\rho = (tt) \tag{5.3}$$

$$[\![ff]\!]\rho = (ff) \tag{5.4}$$

for tests,

$$[\![R(X,Y)]\!]\rho = \overline{R}([\![X]\!]\rho, [\![Y]\!]\rho) \tag{5.5}$$

[4] Env may also be called domain of substitutions, or store. Valid substitutions are always identity on values.

for processes,

$$[\![\mathbf{nil}]\!]\rho = \rho \tag{5.6}$$

$$[\![x := v]\!]\rho = [\rho, \rho \circ [x \Leftarrow [\![v]\!]\rho]] = \rho \circ [x \Leftarrow\check{} [\![v]\!]\rho] \tag{5.7}$$

$$[\![x := h(x,v)]\!]\rho = [\rho, h_x(\rho, [\![v]\!]\rho)] = h_x(.,\check{[\![v]\!]}\rho)\rho \tag{5.8}$$

$$[\![e; e']\!]\rho = \hat{[\![e']\!]}([\![e]\!]\rho + \rho[\![e]\!]) + \rho\hat{[\![e']\!]}[\![e]\!] \tag{5.9}$$

$$[\![e \mid e']\!]\rho = [\![e]\!]([\![e']\!]\rho + \rho[\![e']\!]) + [\![e']\!]([\![e]\!]\rho + \rho[\![e]\!]) + \rho([\![e]\!][\![e']\!] + [\![e']\!][\![e]\!]) \tag{5.10}$$

$$[\![(t \to e)\Box(t' \to e')]\!]\rho = tt^*([\![t]\!]\rho).[\![e]\!]\rho + tt^*([\![t']\!]\rho).[\![e']\!]\rho \tag{5.11}$$

$$[\![\mathbf{rec}\ x.q(x)]\!]\rho = \varinjlim [\![q^n(\mathbf{nil})]\!]\rho \tag{5.12}$$

In all these equations, the labelling is implicit: when an homotopy is used for the semantics of $x := v$ or $x := h(x,v)$, it is formed of some fresh $t_{x:=v,i}$ and $v_{x:=v,i}$ or $t_{x:=h(x,v),i}$ and $v_{x:=h(x,v),i}$ respectively.

Equations 5.1, 5.2, 5.3, 5.4 and 5.5 are obvious. In environment ρ, the semantics of variable x is $\rho(x)$. The constants are interpreted independently of the environment. This is much alike the usual equations in ordinary denotational semantics for imperative languages [GS90].

Equation 5.6 reads "**nil** does not act on the environment".

Equations 5.7 and 5.8, written in two forms, build an homotopy between the environment and the transformed one (notice that substitutions compose the other way round). When ρ is a state, this is just a transition from the input of h to the output of h.

Equation 5.9 applies the sequentialization of $[\![e']\!]$ to $[\![e]\!]$, that is, applies e' to the final states of e, and takes the union with the translation of e.

Equation 5.10 looks like interleaving, but is not. $[\![e']\!]([\![e]\!]\rho)$ is isomorphic to $([\![e']\!]) \otimes ([\![e]\!]\rho)$, thus is a good candidate as a parallel composition (see Chapter 4). But $[\![e]\!]$ and $[\![e']\!]$ may not commute if some of their actions are not independent, therefore we need the term $[\![e]\!]([\![e']\!]\rho)$. We may need as well all other permutations between ρ, $[\![e]\!]$ and $[\![e']\!]$. There can be non-independence if there is simultaneous use of some shared item.

Equation 5.11 takes the (disjoint or not) union of the two alternatives of the guarded statement.

Finally, Equation 5.12 takes the unfolding of a recursive agent as its semantics. The unfolding is represented by the direct limit of the diagram whose objects are the successive steps of unfolding $[\![q^n(\mathbf{nil})]\!]\rho$, and whose morphisms are the obvious ones (all morphisms between $[\![q^{n-1}(\mathbf{nil})]\!]\rho$ and $[\![q^n(\mathbf{nil})]\!]\rho$).

Example 20 • *We consider the term x+=1 in the context* $\rho = [x \Leftarrow 1]$*:*

$$[\![x+ = 1]\!]\rho = [\rho, h_x(\rho, 1)]$$

$$= [[x \Leftarrow 1], [x \Leftarrow 2]]$$

Therefore,

$$\boxed{[\![x+ = 1]\!]\rho \ = {}_{[x \Leftarrow 1]} \xrightarrow{x+ = 1} {}_{[x \Leftarrow 2]}}$$

- *Now, consider the term (x:=1)|(x+=1) in the context* $\rho = [x \Leftarrow 0] \circ [y \Leftarrow 42]$*:*

  $[\![(x := 1) \mid (x+ = 1)]\!]\rho = [\![(x := 1)]\!]([\![(x+ = 1)]\!]\rho) + [\![(x+ = 1)]\!]([\![(x := 1)]\!]\rho) + \ldots$

  the dots being terms which are seen to be equal to the previous two ones. But,

  $$[\![(x := 1)]\!]\rho = \rho \circ [x \check{\Leftarrow} 1]$$
  $$= [\rho, [x \Leftarrow 1] \circ [y \Leftarrow 42]]$$

  Then,

  $$[\![(x+ = 1)]\!]([\![(x := 1)]\!]\rho) = [([\![(x := 1)]\!]\rho), h_x(([\![(x := 1)]\!]\rho, 1)]$$

  $= [[[x \Leftarrow 0][y \Leftarrow 42], [x \Leftarrow 1][y \Leftarrow 42]], [[x \Leftarrow 1][y \Leftarrow 42], [x \Leftarrow 2][y \Leftarrow 42]]]$

  which is geometrically realized by a square whose four vertices (only three of them are disjoint) are $[x \Leftarrow 0][y \Leftarrow 42]$*,* $[x \Leftarrow 1][y \Leftarrow 42]]$*,* $[x \Leftarrow 1][y \Leftarrow 42]]$*, and* $[x \Leftarrow 2][y \Leftarrow 42]$*.*

- *Let us compute the semantics of (x:=1);(x+=3) in the context* $\rho = [x \Leftarrow 0]$

  $$[\![(x := 1); (x+ = 3)]\!]\rho = [\![(x \hat{+}= 3)]\!]([\![x := 1]\!]\rho)$$
  $$= [\![(x \hat{+}= 3)]\!]([[x \Leftarrow 0], [x \Leftarrow 1]])$$
  $$= [\![(x+ = 3)]\!]([x \Leftarrow 1]) + [[x \Leftarrow 0], [x \Leftarrow 1]]$$
  $$= [[x \Leftarrow 1], [x \Leftarrow 4]] + [[x \Leftarrow 0], [x \Leftarrow 1]]$$

  $[\![(x := 1); (x+ = 3)]\!]\rho = {}_{[x \Leftarrow 0]} \xrightarrow{x = 1} {}_{[x \Leftarrow 1]} \xrightarrow{x+ = 3} {}_{[x \Leftarrow 4]}$

- *Finally, let e=((x=1?→y:=2)□(x=y?→x:=0)). Then in context* $\rho = [x \Leftarrow 1][y \Leftarrow 1]$ *we have:*

  $$[\![e]\!] = tt^*([\![x = 1?]\!]\rho).[\![y := 2]\!]\rho + tt^*([\![x = y?]\!]\rho).[\![x := 0]\!]\rho$$

  But,

  $$[\![x = 1?]\!]\rho = tt$$
  $$[\![x = y?]\!]\rho = tt$$

  thus,

  $$[\![e]\!] = [\![y := 2]\!]\rho + [\![x := 0]\!]\rho$$
  $$= [[x \Leftarrow 1][y \Leftarrow 1], [x \Leftarrow 1][y \Leftarrow 2]] + [[x \Leftarrow 1][y \Leftarrow 1], [x \Leftarrow 0][y \Leftarrow 1]]$$

  Thus,

  $[\![e]\!]\rho = {}_{[x \Leftarrow 1][y \Leftarrow 1]}$ with $y := 2$ to ${}_{[x \Leftarrow 1][y \Leftarrow 2]}$ and $x := 0$ to ${}_{[x \Leftarrow 0][y \Leftarrow 1]}$

  This is a one-dimensional branching at state $[x \Leftarrow 1][y \Leftarrow 1]$*. It describes an internal non-deterministic choice.*

An alternative semantics à la SOS

States ρ describe the store as a substitution between values and variables. Then we have by eliminating the context ρ in the denotational semantics,

$$\frac{}{\mathbf{nil} \models \rho \xrightarrow{\mathbf{nil}} \rho}$$

$$\frac{}{x := v \models \rho \xrightarrow{x := \rho(v)} \rho \circ [x \Leftarrow \rho(v)]}$$

$$\frac{}{x := h(x,v) \models \rho \xrightarrow{x := h(x,\rho(v))} \rho \circ [x \Leftarrow h(x,\rho(v))]}$$

$$\frac{e \models \rho \xrightarrow{a} \sigma \qquad \rho(t) = tt}{(t \to e)\Box(t' \to e') \models \rho \xrightarrow{a} \sigma}$$

$$\frac{e' \models \rho' \xrightarrow{a'} \sigma' \qquad \rho(t') = tt}{(t \to e)\Box(t' \to e') \models \rho' \xrightarrow{a'} \sigma'}$$

$$\frac{e \models \rho \xrightarrow{a} \sigma \qquad e' \models \rho' \xrightarrow{a'} \sigma'}{e;e' \models \rho \xrightarrow{a} \sigma}$$

$$\frac{e \models \rho \xrightarrow{a} \sigma \qquad e' \models \rho' \xrightarrow{a'} \sigma' \quad f \in H_0(e, \partial_0)}{e;e' \models \rho' \circ f \xrightarrow{a' \circ f} \sigma' \circ f}$$

$$\frac{e \models \rho \xrightarrow{a} \sigma \qquad e' \models \rho' \xrightarrow{a'} \sigma'}{e \mid e' \models \rho \circ \rho' \xrightarrow{a \circ a'} \sigma \circ \sigma'}$$

$$\frac{e \models \rho \xrightarrow{a} \sigma \qquad e' \models \rho' \xrightarrow{a'} \sigma'}{e \mid e' \models \rho' \circ \rho \xrightarrow{a' \circ a} \sigma' \circ \sigma}$$

$$\frac{q[\mathbf{rec}\ x.q[x]] \models \rho \xrightarrow{a} \sigma}{\mathbf{rec}\ x.q[x] \models \rho \xrightarrow{a} \sigma}$$

5.5.3 A variant with a Fork operator

We can replace the parallel composition operator by a fork operator (as in [Hav94] or in [Rep92]).

Let $\mathcal{L}_f$ be the language whose syntax is as follows.
Let Var be a set of variable names $(x, y, z...)$. We consider a set of values $v \in Val$, containing integers n, booleans tt and ff. We write X, Y, Z for objects which are values or variables. f is any function on Val.
The language is formed out of values v, tests t, and expressions e:

$$\begin{array}{rcl} v & ::= & x \\ & | & n \\ & | & tt \\ t & ::= & R(X,Y) \\ e & ::= & \mathbf{nil} \\ & | & x := v \\ & | & e; e' \\ & | & fork(e) \\ & | & t \rightarrow e \Box t' \rightarrow e' \\ & | & \mathbf{rec}\ x.q(x) \end{array}$$

where q is any syntactic expression of $\mathcal{L}_f$ with one hole (a context).
Now, we give the semantics using the same domain D as before.

The semantic function is now slightly more complex:

$$[\![.]\!] : \mathcal{L}_f \longrightarrow Hom(Env, Hom(Hom(D,D),D))$$

The new argument is a context $\kappa \in Hom(D,D)$ (similar to continuations used in denotational semantics [Sch86]). Env is the domain of environments, $Env = ZHom(C \oplus V, C \oplus V) = Hom(C \oplus V, C \oplus V)$. We propose to give a semantics using the "parallel" continuation as follows,

for values,

$$[\![x]\!]\rho\kappa = \kappa(\rho(x))$$

$$[\![n]\!]\rho\kappa = \kappa(\rho(n))$$

$$[\![tt]\!]\rho\kappa = \kappa(\rho(tt))$$

$$[\![ff]\!]\rho\kappa = \kappa(\rho(ff))$$

for tests,

$$[\![R(X,Y)]\!]\rho\kappa = \kappa(\overline{R}([\![X]\!]\rho\, Id, [\![Y]\!]\rho\, Id))$$

for processes,

$$[\![\mathbf{nil}]\!]\rho\kappa = \kappa(\rho)$$

$$[\![x := v]\!]\rho\kappa = \kappa(\rho \circ [x \Leftarrow ([\![\check{v}]\!]\rho\ Id)])$$

$$[\![e; e']\!]\rho\kappa = [\![e]\!]\rho\ \kappa \circ [\![\hat{e'}]\!]$$

$$[\![fork(e)]\!]\rho\kappa = [\![e]\!](\kappa\rho)\ Id + \kappa([\![e]\!]\rho)$$

$$[\![t \to e \Box t' \to e']\!]\rho\kappa = tt^*([\![t]\!]\rho\ Id) \otimes [\![e]\!]\rho\kappa + tt^*([\![t']\!]\rho\ Id) \otimes [\![e']\!]\rho\kappa$$

$$[\![\mathbf{rec}\ x.q(x)]\!]\rho\kappa = \kappa(\varinjlim [\![q^n]\!]\rho\ Id)$$

Summary We have shown that using the categorical properties of Chapter 4 we could give denotational semantics to some concurrent languages, like CCS or some toy imperative languages. We have shown that HDA could be thought of as domains (as in domain theory) of possible executions and that there were interesting constructions from denotational semantics which apply to the HDA framework, like recursive domain equations or continuations (in order to model forking processes). Considerations about the construction of domains for concurrent imperative languages lead to a very geometric construction involving "homotopies" that deform some part of a domain to another one in a constructive way.

Some work remains to be done on recursive domain equations involving contravariant functors (in particular "reflexive" domains). Languages like CML could then be given semantics using HDA in a categorical way. A possibility seems to be to add some topology that constrains the morphisms between HDA (by considering continuous morphisms only). This has not been formalized yet in a suitable way.

Part III

Geometric Properties

Chapter 6

Basic geometric properties

Given the algebraic formulation of HDA, we are now ready to study the very basic geometric properties of the transition system it represents in purely algebraic terms, in the style of ordinary homological algebra. This gives definitions as well as means of computations (some of them are given at the end of this chapter). Here we focus on initial and final states, branchings and mergings, deadlocks and refinement. At the end of this chapter we will see applications of the classification of these local geometric shapes.

6.1 Homology

(*) Definition 5 *For $(Q,\partial_0,\partial_1)$ a general HDA, we define two sequences of homology (see for instance [ML63]) modules:*

- $H_i(Q,\partial_0) = \frac{Ker\ \partial_0^i}{Im\ \partial_0^i}$
- $H_i(Q,\partial_1) = \frac{Ker\ \partial_1^i}{Im\ \partial_1^i}$

where $\partial_j^i = \sum_{p+q=i} \partial_j^{p,q}$. An element of Ker ∂_j^i (i.e. the kernel of the function ∂_j^i, the module formed of the x such that $\partial_j^i(x) = 0$) is an i-cycle*, and an element of Im ∂_j^{i+1} (i.e. the image of ∂_{i+1}^j, i.e. $\partial_j^{i+1}(Q_{i+1})$) is an* i-boundary*. An element of $H_i(Q,\partial_0)$ is called a* branching *of dimension i. An element of $H_i(Q,\partial_1)$ is called a* merging *of dimension i. We write $H_*(T,\partial_j)$ for $\bigoplus_{k\geq 0} H_k(T,\partial_j)$.*

Example 21

- *For HDA (1), Example 4, Chapter 2, we have:*
  $H_0(M,\partial_0) = (\alpha)$, $H_0(M,\partial_1) = (1)$, and the other homology groups are null.
- *For HDA (2), all the homology groups are null.*
- *For HDA (3), we have:*
  $H_0(M,\partial_0) = (\gamma)$, $H_0(M,\partial_1) = (1)$, and the other homology groups are null.

- *For HDA (4), we have:*
  $H_0(M,\partial_0) = (\alpha)$, $H_0(M,\partial_1) = (1)$, $H_1(M,\partial_0) = (b-a)$, $H_1(M,\partial_1) = (b'-a')$, *and the other homology groups are null. The branching* $(b-a)$ *of dimension one expresses the fact that in (4) there is a non-deterministic choice between actions* a *and* b*. The confluence* $(b'-a')$ *shows that after the actions* b' *and* a'*, the system goes to a same (idle) state.*

(*) Lemma 8 *Let* $f : (K,\partial) \to (K',\partial')$ *be a morphism of complex. Then* f *induces a module homomorphism* $f_* : H_*(K,\partial) \to H_*(K',\partial')$.

(*) Proof. We just have to verify that (for all k):

$$f(Ker\ \partial^k) \subseteq Ker\ \partial'^k$$

and

$$f(Im\ \partial^k) \subseteq Im\ \partial'^k$$

Let $x \in Ker\ \partial^k$, then $\partial(x) = 0$, so is $f(\partial(x)) = \partial(f(x))$, thus $f(x)$ belongs to $Ker\ \partial$. But f is graded, so $f(x) \in Ker\ \partial^k$.
Now, take x in $Im\ \partial^k$. Then, $x = \partial(y)$, with $y \in K_k$. $f(x) = f(\partial(y)) = \partial(f(y))$. f is graded, so $f(y) \in K'_k$, and then $f(x) \in Im\ \partial^k$.□

The previous lemma states that local geometric invariants like the homology groups are invariants of simulations.

6.1.1 Initial and final states

In semi-regular, regular HDA and partial HDA, a final state is a state from which no path can begin. Similarly, an initial state is a state to which no path can lead to. Applying the $\mathcal{A}$ functor we see that,

- the module of states of an HDA $\mathcal{A}(M)$ is $\mathcal{A}(M)_0 = Ker\ \partial_{0|\mathcal{A}(M)_0}$,
- a path can begin from a state s if and only if there exists a 1-transition a with $s = \partial_0(a)$ (so $s \in Im\ \partial_0$),
- a path can lead to a state s if and only if there exists a 1-transition a with $\partial_1(a) = s$ (so $s \in Im\partial_1$).

Therefore, it is easy to see that the module generated by the initial states (respectively final states) of some semi-regular, regular or partial HDA M is $H_0(\mathcal{A}(M),\partial_1)$ (respectively $H_0(\mathcal{A}(M),\partial_0)$). This is then the most consistent definition for general HDA,

Definition 29 *We call final (or accepting) state of an HDA* M *any state of* $H_0(M,\partial_0)$.

Example 22 *Consider the (standard) automaton* $(A, \Sigma, \delta, I, F)$ *with* $A = \{u, v, w\}$, $\Sigma = \{a, b\}$, $I = \{u\}$, $F = \{v\}$ *and* $\delta = \{(u, a, v), (u, b, w)\}$. *Then the associated HDA is* M, *with:*

$$M_0 = (u) \oplus (v) \oplus (w)$$

$$M_1 = (a) \oplus (b)$$

and, $\partial_0(a) = u = \partial_0(b)$, $\partial_1(a) = v$, $\partial_1(b) = 0$ *Obviously,* $H_0(M, \partial_0) = (v) = R - Mod(F)$, $H_1(M, \partial_0) = (a) \oplus (b)$, $H_0(M, \partial_1) = (u)$ *and* $H_1(M, \partial_1) = (b)$.

Definition 30 *We call initial state of an HDA* M *any state of* $H_0(M, \partial_1)$.

Example 23

- *Consider the automaton of the previous example. We have seen that* $H_0(M, \partial_1) = (u)$: u *is an initial state of* M.
- *Let* M *and* N *be the following HDA (*M *and* N *are composed of one 1-transition* a, *respectively* b*),*

$$\begin{array}{ccc} M_{1,0} = (a) & \longrightarrow & M_{0,0} = (\alpha) \\ \downarrow & & \\ M_{1,-1} = (\beta) & & \end{array}$$

$$\begin{array}{ccc} N_{1,0} = (b) & \longrightarrow & N_{0,0} = (\gamma) \\ \downarrow & & \\ N_{1,-1} = (\delta) & & \end{array}$$

Then $M \otimes N^*$ *is,*

$$\begin{array}{ccccc} & & (\alpha \otimes \gamma^*) & \longrightarrow & (\alpha \otimes \gamma^*) \\ & & \downarrow & & \downarrow \\ (a \otimes \delta^*) & \longrightarrow & (a \otimes b^*) \oplus (\alpha \otimes \delta^*) \oplus (\beta \otimes \gamma^*) & \longrightarrow & (\alpha \otimes b^*) \\ \downarrow & & \downarrow & & \\ (\beta \otimes \delta^*) & \longrightarrow & (\beta \otimes b^*) & & \end{array}$$

Its states are $a \otimes b^*$, $\alpha \otimes \gamma^*$, $\alpha \otimes \delta^*$, $\beta \otimes \delta^*$ *and* $\beta \otimes \gamma^*$. *But,*

- $\partial_1(\alpha \otimes \gamma^*) = \alpha \otimes b^*$, *hence according to the interpretation of boundary operators we have in dual HDA, the information beginning of* $\alpha \otimes \gamma^*$ *is* $\alpha \otimes b^*$, *i.e. the event "waiting for transition* b *to be fired in context* α*",*
- $\partial_1(\alpha \otimes \delta^*) = 0$,
- $\partial_1(\beta \otimes \gamma^*) = \beta \otimes b^*$,
- $\partial_1(\beta \otimes \delta^*) = 0$,
- $\partial_1(a \otimes b^*) = \beta \otimes b^*$.

Therefore, according to Definition 30 the module of initial states for $M \otimes N^*$ is equal to $(\alpha \otimes \delta^*) \oplus (\beta \otimes \delta^*)$. Intuitively, the definition is coherent with the interpretation we have of events. $\alpha \otimes \delta^*$ and $\beta \otimes \delta^*$ are valid initial states since these are not blocked any longer by waiting for action b, whereas nothing can be fired from $\alpha \otimes \gamma^*$ nor from $\beta \otimes \gamma^*$ since the external b action has not yet been fired.

Now we come to deadlocks and their dual, the "initial" deadlocks.

6.1.2 Deadlocks and initial deadlocks

We have seen that we needed the partial HDA to handle deadlocks. In partial HDA, the obvious definition of a deadlock, is a n-transition ($n > 1$) from which no action can be fired. As $n > 1$, this means that all boundaries, i.e. all ways to complete normally the n-transition are missing. Therefore a deadlock of dimension n in $\mathcal{A}(M)$ where M is a closed partial HDA is a n-transition t such that $\partial_1(t) = 0$. All n-deadlocks boundaries of a $(n+1)$-transition should be considered equivalent as they essentially describe which actions will not terminate if asynchronously executed. Since we want to distinguish n-deadlocks (which involve only one elementary n-transition) from confluences (which involve more than one elementary n-transition), this leads to the following definition for general HDA,

Definition 31 *A n-transition leading to (or is) a deadlock in a HDA M is any elementary n-transition of $H_n(M, \partial_1)$. The word deadlock is given to the 1-transitions which deadlock M (for n greater than one, we say n-deadlock).*

Example 24

- *The typical 1-deadlock is,*

$$1 \xrightarrow{\ t\ } 0$$

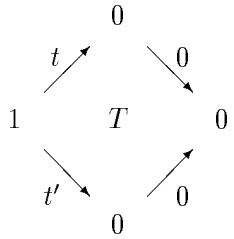
We can interpret its unique maximal path as,

(1) *the processor is in an idle state,*

(t) *the processor is firing transition t,*

(0) *the processor never terminates the execution of t (no final state can be observed).*

- *In Example 22, b is an action leading to a deadlock.*
- *The typical 2-deadlock is,*

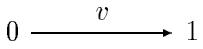


It is easy to see (examining the differents paths) that it describes a program which deadlocks two processors.

Dually, we can define "initial" deadlocks as actions that can never be fired, but would terminate if fired. We have already seen them in the construction of homotopies (Section 5.5).

Definition 32 *A n-transition leading to (or is) an initial deadlock in an HDA M is an elementary n-transition of $H_n(M, \partial_0)$. Again, the word initial deadlock is given to 1-transitions.*

Example 25 *The typical initial 1-deadlock is the dual of the 1-deadlock (t) (under the reversing of time),*

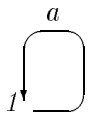


6.1.3 Divergence

Definition 33 *A HDA M diverges if and only if it does not have any final state, i.e. if and only if $H_0(M, \partial_0) = 0$*

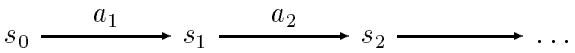
Example 26

- *Let M be the HDA with $M_0 = (1)$, $M_1 = (a)$ and $\partial_0(a) = \partial_1(a) = 1$, represented by,*



  Then $H_*(M, \partial_0) = 0$, and M diverges.

- *The following HDA diverges as well,*



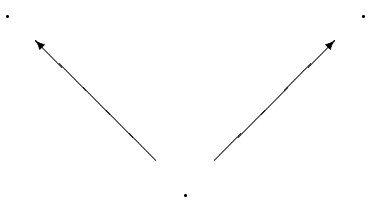
Dually, an HDA M co-diverges, if it has no beginning.

6.1.4 Branchings and mergings

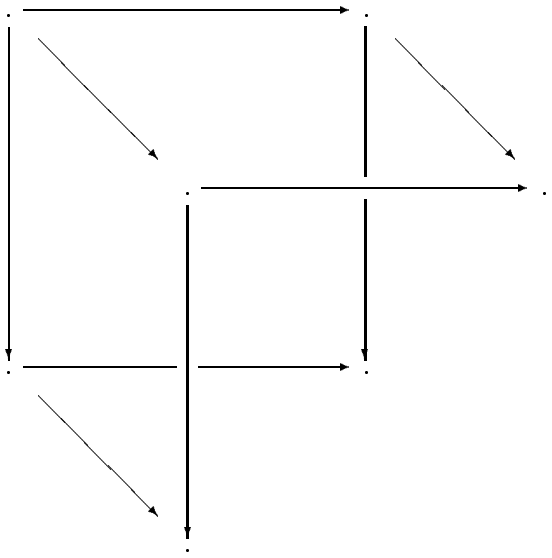
We have already defined branchings and mergings in all dimensions. Branchings provide a measure of non-determinism in all dimensions. Mergings reduce the non-determinism of an automaton. For instance,

Example 27

- *A typical branching of dimension one is,*

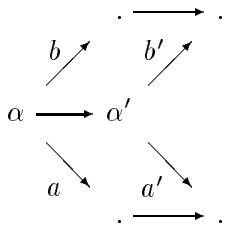


- *A typical branching of dimension two is,*



where the three faces are filled in.

Be careful though and notice that branchings should be called "necessary branchings" since for instance,



has only one 1-branching, namely $(a' - b')$. $(a - b)$ is certainly not one, and this can be interpreted (in a modal way) as saying that in state α' a choice is necessary between a' and b' whereas in α, no choice is necessary.

Definition 34 *A HDA M is finitely branching in dimension n if and only if $H_n(M, \partial_0)$ is a free module of finite dimension. Similarly, we say that M is finitely merging in dimension n if and only if $H_n(M, \partial_1)$ is a free module of finite dimension. By M is finitely branching (resp. merging), with no precision of dimension, we mean M is finitely branching (resp. merging) in all dimensions.*

Notice that if M is finitely branching then it has finitely many final states, and if M is finitely merging, then it has finitely many initial states.

Example 28 *In all examples we had up to now, automata were finitely branching and merging.*

An application of the classification of branchings will be given at the end of this chapter for determining necessary conditions for some (branching-time) semantic equivalences. The mathematically inclined reader will have guessed that typical non-existence (and obstruction) theorems from algebraic topology will be given computer-scientific meaning.

6.2 Refinement of actions

The notion of subdivision of cubicalation is directly related to refinement of actions by completely deterministic processes (like division of action: $a \longrightarrow a_1; a_2$).

Define an operator G, the "subdivision" operator by

$$G(I) \equiv (\{0\} \times I) \coprod \{1\} \times I))/\{(0,1) = (1,0)\}$$

$$G(x) = \begin{cases} (0,x) & 0 \leq x \leq 1/2 \\ (1, x-1/2) & 1/2 \leq x \leq 1 \end{cases}$$

and for all n, $G(\square_n) = G(I)^n$.

G splits I into two equal parts, then $I \times I \equiv \square_2$ into four equal parts etc. $\square_n$ into 2^n equal parts.

$G(\cdot \longrightarrow \cdot)$ = $\cdot \longrightarrow \cdot \longrightarrow \cdot$

More precisely,

Lemma 17 *$G : \square_n \to G(\square_n)$ is an isomorphism in Top and $G(\square_n)$ is an amalgamated sum of 2^n copies of $\square_n$.*

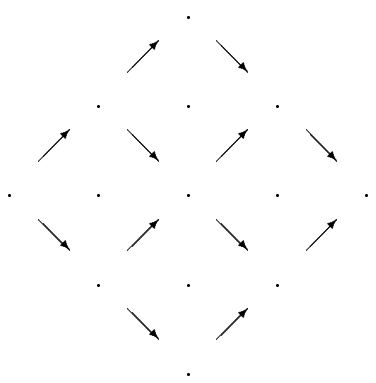
PROOF. Straightforward. □

A morphism $f : \square_n \to X$ in *Top* therefore induces 2^n morphisms $f^1, ..., f^{2^n}$ each one from a copy of $\square_n$ in $G(\square_n)$.

Consider now for a semi-regular automaton M the sub-HDA $\eta(M)(M)$ of $S(|M|)$. Its elementary objects are morphisms $f_{n,k} : \square_n \to |M|$. Then,

Definition 35 *A (one-step) refinement of a unlabeled regular HDA M is the automaton $N = Re(M)$ generated by the objects $f^1_{n,k}, ..., f^{2^n}_{n,k}$ for all n and k.*

Example 29 *A one-step refinement of automaton (3) of Example 1 is:*



It is quite easy to prove that $H_*(Re(M), \partial_j) \equiv H_*(M, \partial_j)$. Therefore, refinement does not change the essential topological properties of HDA, and thus is a valid technique for abstracting from unnecessary details when analysing programs (see Section 6.5 where invariants of bisimulation equivalence are given in terms of the H_*). This is nevertheless not the most general kind of refinement techniques one can consider (see [Ren93]).

For labeled regular automata $p : E \to B$, the (one-step) refinement process must be applied at the same time to E and B:

Definition 36 *A (one-step) refinement of a labeled regular automaton $p : E \to B$ is the labeled regular automaton $q : N(E) \to N(B)$ with q induced by $S(\mid p \mid)$ on $G(\square_n)$.*

6.3 Homology functors

Now that we have defined some interesting properties in terms of homology, we need to be able to compute these groups in an effective way. Most results here come from [ML63].

6.3.1 Homology of limits and colimits

(*) Lemma 9 *the homology of T, coproduct of Q and Q' is:*

$$\forall i, j, H_i(T, \partial_j) = H_i(Q, \partial_j) \oplus H_i(Q', \partial_j)$$

PROOF. Straightforward. □

(*) Theorem 2 *The homology functor commutes with the direct limit functor:*

$$H_*(\varinjlim C_i, \varinjlim \partial_i) = \varinjlim H_*(C_i, \partial_i)$$

PROOF. See for example [Mas78]. □

6.3.2 Homology of tensor products and Hom

Now the homology groups of the tensor product are given by the Künneth formula[1].

[1] Simplified here to the case where R does not contain any proper torsion subgroup (like $\mathbb{Z}$ or $\mathbb{Z}_2$). for a more general formulation, see for instance [ML63].

(*) Lemma 10 *For Q and Q' HDA such that for all i, $H_i(Q,\partial)$ and Q_i are projective modules, we have:*

$$H_k(T,\partial) = \bigoplus_{i=0\ldots k} (H_{k-i}(Q,\partial) \otimes H_i(Q',\partial))$$

PROOF. See for example [Mas78] or [ML63]. □

For a HDA M, we write Z_jM for the module of cycles (for ∂_j) of M, and B_jM for the module of boundaries (for ∂_j) of M.

Lemma 18 *$Z_0Hom(P,Q)_n \cap Z_1Hom(P,Q)_n$, for P and Q HDA, is the module generated by the morphisms (of HDA) of degree n between P and Q.*

PROOF. The relation $f \in Z_jHom(P,Q)_n$ reads:

$$\partial_i \circ f - (-1)^n f \circ \partial_i = 0$$

Therefore f is a morphism of degree n. □

Define $H^n(K;L) = H_{-n}(Hom(K,L))$. When $L = (1)$, $H^n(K;L)$ is called the nth cohomology group of K.

For the next result, we define the notion of *exact sequence*.

(*) Definition 6 *(see [Lan93a]) A sequence of homomorphisms having more than one term like:*

$$G_1 \xrightarrow{f_1} G_2 \xrightarrow{f_2} G_3 \qquad \ldots \xrightarrow{f_{n-1}} G_n$$

is called exact if and only if

$$\forall i = 1, ..., n-2, \quad Im\ f_i = Ker\ f_{i+1}$$

A short exact sequence is an exact sequence with n=5, $G_1 = G_5 = 0$.

(*) Lemma 11 *(Homotopy classification theorem)*
For K and L complexes of abelian groups with each K_n free as an abelian group, there is for each n, a short exact sequence

$$\begin{array}{ccc} & & 0 \\ & & \downarrow \\ \prod_{p=-\infty,\ldots,\infty} Ext(H_p(K), H_{p+n+1}(L)) & \xrightarrow{\beta} & H_n(Hom(K,L)) \\ & & \downarrow \alpha \\ & & \prod_{p=-\infty,\ldots,\infty} Hom(H_p(K), H_{p+n}(L)) \\ & & \downarrow \\ & & 0 \end{array}$$

This sequence splits (though unnaturally in K).

PROOF. see [ML63]. □

In particular, if all $H_p(K)$ are projective (or the $H_{p+n+1}(L)$, injective) then

$$Ext(H_p(K), H_{p+n+1}(L)) = 0$$

and as the previous sequence splits,

$$H_n(Hom(K,L)) \equiv \prod_{p=-\infty,\ldots,\infty} Hom(H_p(K), H_{p+n}(L))$$

When K is a HDA with all $H_p(K)$ projective, then $H_n(K^*) = H^{-n}(K) = (H_n(K))^*$.

6.3.3 Exact sequences

The aim of this subsection is to show the use of the Mayer-Vietoris exact sequence to compute the homology of a complex (Q, ∂), given the homology of two subcomplexes (Q^1, ∂) and (Q^2, ∂) whose union is (Q, ∂), and the homology of their intersection $(Q^1 \cap Q^2, \partial)$. This will be extensively used in next section.

(*) Definition and lemma 3 *Consider the following short exact sequence of chain complexes:*

$$0 \longrightarrow L' \xrightarrow{\alpha} L \xrightarrow{\beta} L'' \longrightarrow 0$$

then there is a morphism $[\partial]$ of degree -1, called a connecting homomorphism, such that if (x'') is a sub-R-module of $H_*(L'', \partial)$, then $([\partial](x'')) = (\alpha^{-1} \circ \partial \circ \beta^{-1}(x''))$ is a sub-R-module of $H_*(L', \partial)$. Then, the following sequence is also exact:

$$\begin{array}{ccc} H(L') & \xrightarrow{\alpha_*} & H(L) \\ & {\scriptstyle [\partial]} \nwarrow \quad \swarrow {\scriptstyle \beta_*} & \\ & H(L'') & \end{array}$$

PROOF. See [Lan93a]. □

(*) Proposition 4 *The following sequence, called the Mayer-Vietoris sequence, is exact:*

$$\ldots \longrightarrow H_n(Q^1) \oplus H_n(Q^2) \xrightarrow{i_*} H_n(Q) \xrightarrow{[\partial]} H_{n-1}(Q^1 \cap Q^2) \xrightarrow{j_*} H_{n-1}(Q^1) \oplus H_{n-1}(Q^2) \xrightarrow{i_*} \ldots$$

where, $i_*(x^1 \oplus x^2) = i^1_*(x^1) + i^2_*(x^2)$, if $i^1 : Q^1 \to Q$ and $i^2 : Q^2 \to Q$ are the inclusion morphisms from Q^1 to Q and from Q^2 to Q respectively, $j_*(x) = j^1_*(x) \oplus j^2_*(x)$, if $j^1 : Q^1 \cap Q^2 \to Q^1$ and $j^2 : Q^1 \cap Q^2 \to Q^2$ are the inclusion morphisms from $Q^1 \cap Q^2$ to Q^1 and from $Q^1 \cap Q^2$ to Q^2 respectively, and $[\partial]$ is the connecting homomorphism of Definition and Lemma 3.

PROOF. This result is due to the application of Definition and Lemma 3 to the following exact sequence of R-modules (see for instance [Lan93a]):

$$0 \longrightarrow Q^1 \cap Q^2 \longrightarrow Q^1 \oplus Q^2 \longrightarrow Q^1 + Q^2 \longrightarrow 0$$

□

Remark We find again Lemma 9 using Proposition 4. $Q^1 \cap Q^2$ is now empty, thus its homology is 0. Therefore we have short exact sequences (for all n):

$$0 \longrightarrow H_n(Q^1) \oplus H_n(Q^2) \longrightarrow H_n(Q) \longrightarrow 0$$

We conclude that $H_n(Q)$ is isomorphic to $H_n(Q^1) \oplus H_n(Q^2)$.

6.4 Branchings and mergings of CCS

In this section, we use the general HDA semantics of Chapter 5 and results of this chapter to compute the branchings and mergings of our CCS-like process algebra. This computation will be used in next section to (semi-) decide some semantic equivalences and to prove global results about computability of some functions modulo some semantic equivalences.

Lemma 19 *(Branchings and mergings for CCS) From now on we suppose that for all HDA we consider, their underlying modules are projective and their homology modules are projective. This is trivially verified when the base ring R is a division ring (R=$\mathbb{Z}_2$ or $\mathbf{Q}$)*

(i)

$$H_i([\![a]\!], \partial_j) = \begin{cases} (1) & (i,j) = (0,1) \\ (\alpha_k) & (i,j) = (0,0) \\ 0 & i > 0 \end{cases}$$

(ii)

$$H_i([\![\overline{a}]\!], \partial_j) = \begin{cases} (1) & (i,j) = (0,1) \\ (\overline{\alpha}_k) & (i,j) = (0,0) \\ 0 & i > 0 \end{cases}$$

(iii)

$$H_i([\![\mathbf{nil}]\!],\partial_j) = \begin{cases} (1) & (i,j)=(0,0)\wedge(i,j)=(0,1) \\ 0 & i>0 \end{cases}$$

(iv)

- $H_0([\![p+q]\!],\partial_0) = H_0([\![p]\!],\partial_0)\oplus H_0([\![q]\!],\partial_0)$ *if* $\exists(X,X')\in[\![p]\!]\times[\![q]\!]$, $\partial_0(X)=\partial_0(X')=(1)$,
- $H_0([\![p+q]\!],\partial_0) = (H_0([\![p]\!],\partial_0)\oplus H_0([\![q]\!],\partial_0))/(1)$ *if* $\nexists(X,X')\in[\![p]\!]\times[\![q]\!]$, $\partial_0(X)=\partial_0(X')=1$,
- $H_1([\![p+q]\!],\partial_0) = H_1([\![p]\!],\partial_0)\oplus H_1([\![q]\!],\partial_0)\oplus R-Mod\{X-X'/(X,X')\in[\![p]\!]\times[\![q]\!],\partial_0(X)=\partial_0(X')=1\}$,
- $H_i([\![p+q]\!],\partial_0) = H_i([\![p]\!],\partial_0)\oplus H_i([\![q]\!],\partial_0)$ *for all* $i\geq 2$,
- $H_0([\![p+q]\!],\partial_1) = (1)$,
- $H_i([\![p+q]\!],\partial_1) = H_i([\![p]\!],\partial_1)\oplus H_i([\![q]\!],\partial_1)$ *for all* $i\geq 1$.

(v)

$$H_i([\![p.q]\!],\partial_j) = \begin{cases} H_0([\![p]\!],\partial_1)=(1) & (i,j)=(0,1) \\ H_0([\![p]\!],\partial_0)\otimes H_0([\![q]\!],\partial_0) & (i,j)=(0,0) \\ H_i([\![p]\!],\partial_j)\oplus H_0([\![p]\!],\partial_0)\otimes H_i([\![q]\!],\partial_j) & i\geq 1 \end{cases}$$

(vi)

$$H_i([\![p\|q]\!],\partial_j) = H_i([\![p]\!],\partial_j)\otimes H_i([\![q]\!],\partial_j)$$

(vii)

$$H_i([\![p\backslash a]\!],\partial_j) = F_{A_1}\circ\ldots\circ F_{A_n}(H_*([\![p]\!],\partial_j))_i$$

where,

- $i\leq j\Rightarrow dimA_i\leq dimA_j$
- $\{u\in[\![p]\!]/\exists x,y\in D, u=x\otimes a_i\otimes y\vee u=x\otimes\overline{a}_i\otimes y\} = R-Mod(A_1,\ldots A_n)$
- $$\begin{cases} i\neq dimA_k, & F_{A_k}(H_*(Q,\partial))_i = H_i(Q,\partial) \\ i=dim\ A_k & F_{A_k}(H_*(Q,\partial))_i = \begin{cases} H_i(Q,\partial)\ \textit{if}\ H_i(Q,\partial)\subseteq\sum\limits_{u\neq A_k}(u) \\ \textit{otherwise,} \\ H_i(Q,\partial)/\{v/v\in H_i(Q,\partial), v\notin\sum\limits_{u\neq A_k}(u)\} \end{cases} \end{cases}$$

(viii)

$$H_i([\![\mathbf{rec}\ x.q[x]]\!],\partial_j) = \varinjlim H_i([\![q^n[\mathbf{nil}]]\!],\partial_j)$$

where the diagram on which we take the limit is the image of the diagram defining **rec** $x.q[x]$ *by the functor* H_i.

PROOF.

(i),(ii),(iii) Direct computation.

(iv) Consider first a HDA Q and two subHDA Q^1 and Q^2 such that,

- $Q^1 \cup Q^2 = Q$
- $Q^1 \cap Q^2 = (\alpha) = H_0(Q, \partial_1) = H_0(Q^1, \partial_1) = H_0(Q^2, \partial_1)$

Then,

- $H_0(Q, \partial_0) = H_0(Q^1, \partial_0) \oplus H_0(Q^2, \partial_0)$ if $\exists (X, X') \in Q^1 \times Q^2, \partial_0(X) = \partial_0(X') = \alpha$
- $H_0(Q, \partial_0) = (H_0(Q^1, \partial_0) \oplus H_0(Q^2, \partial_0))/(\alpha)$ if $/\exists (X, X') \in Q^1 \times Q^2, \partial_0(X) = \partial_0(X') = \alpha$
- $H_1(Q, \partial_0) = H_1(Q^1, \partial_0) \oplus H_1(Q^2, \partial_0) \oplus R - Mod\{X - X'/(X, X') \in Q^1 \times Q^2, \partial_0(X) = \partial_0(X') = \alpha\}$
- $\forall k \geq 2, H_k(Q, \partial_0) = H_k(Q^1, \partial_0) \oplus H_k(Q^2, \partial_0)$

To prove this, we write the Mayer-Vietoris sequence for the complexes with boundary operator ∂_0 :

$$\cdots \longrightarrow H_n(Q^1) \oplus H_n(Q^2) \xrightarrow{i_*} H_n(Q) \xrightarrow{[\partial]} H_{n-1}(Q^1 \cap Q^2) \xrightarrow{j_*} H_{n-1}(Q^1) \oplus H_{n-1}(Q^2) \xrightarrow{i_*} \cdots$$

But, $Q^1_k \cap Q^2_k$ is null for $k \geq 1$, so we have short exact sequences:

$$\forall m \geq 2, \quad 0 \longrightarrow H_m(Q^1) \oplus H_m(Q^2) \xrightarrow{i_*} H_m(Q) \longrightarrow 0$$

Therefore, $H_m(Q, \partial_0)$ is isomorphic to $H_m(Q^1) \oplus H_m(Q^2)$, with isomorphism i_*. Now, we have a long exact sequence,

$$0 \longrightarrow H_1(Q^1, \partial_0) \oplus H_1(Q^2, \partial_0) \xrightarrow{i_*} H_1(Q, \partial_0) \xrightarrow{[\partial_0]} H_0(Q^1 \cap Q^2, \partial_0) \xrightarrow{j_*} H_0(Q^1, \partial_0) \oplus H_0(Q^2, \partial_0) \longrightarrow H_0(Q, \partial_0) \longrightarrow 0$$

$H_0(Q^1 \cap Q^2, \partial_0) = Q^1 \cap Q^2 = (\alpha)$, so $Ker\, j_* = (\alpha)$ or $Ker\, j_* = 0$. $Im\, [\partial_0] = Ker\, j_*$ implies that the first case is only possible if $\exists (X, X') \in Q^1 \times Q^2, \partial_0(X) = \partial_0(X') = \alpha$.

(a) Suppose $\exists (X, X') \in Q^1 \times Q^2, \partial_0(X) = \partial_0(X') = \alpha$, then $j_* = 0$. Then we have the short exact sequence

$$0 \longrightarrow H_1(Q^1, \partial_0) \oplus H_1(Q^2, \partial_0) \xrightarrow{i_*} H_1(Q, \partial_0) \xrightarrow{[\partial_0]} (\alpha) \longrightarrow 0$$

From this we deduce, $H_1(Q,\partial_0)$ is $[\partial_0]^{-1}(\alpha)\oplus H_1(Q^1,\partial_0)\oplus H_1(Q^2,\partial_0)$. We conclude by noticing that $[\partial_0]^{-1}(\alpha) = R\text{-}Mod\{X-X'/(X,X')\in Q^1\times Q^2, \partial_0(X)=\partial_0(X')=\alpha\}$. We have also a short exact sequence

$$0 \longrightarrow H_0(Q^1,\partial_0)\oplus H_0(Q^2,\partial_0) \longrightarrow H_0(Q,\partial_0) \longrightarrow 0$$

Therefore $H_0(Q,\partial_0) = H_0(Q^1,\partial_0)\oplus H_0(Q^2,\partial_0)$.

(b) Assume the contrary. Then, $Ker\, j_* = 0 = Im\,[\partial]_{0|H_1(Q,\partial_0)}$ and $H_1(Q,\partial_0) = H_1(Q^1,\partial_0)\oplus H_1(Q^2,\partial_0)$. This is what we wanted to prove under the hypothesis $\nexists (X,X')\in Q^1\times Q^2, \partial_0(X)=\partial_0(X')=\alpha$. We have also a short exact sequence,

$$0 \longrightarrow (\alpha) \longrightarrow H_0(Q^1,\partial_0)\oplus H_0(Q^2,\partial_0) \longrightarrow H_0(Q,\partial_0) \longrightarrow 0$$

Therefore, $H_0(Q,\partial_0) = (H_0(Q^1,\partial_0)\oplus H_0(Q^2,\partial_0))/(\alpha)$

Then, notice that for all CCS-terms p, q, by induction, $H_0([\![p]\!],\partial_1) = (1)$ and $\nexists (X,X')\in [\![p]\!]\times[\![q]\!]$, $\partial_1(X)=\partial_1(X')\neq 0$. Applying the previous result to the complexes for ∂_0 and for ∂_1 gives the result.

(v) We have $[\![p.q]\!] = [\![p]\!] + H_0([\![p]\!],\partial_0)\otimes[\![q]\!]$. As we always use fresh copies of atomic actions, $[\![p]\!]\cap[\![q]\!] = (1)$. Therefore, $[\![p]\!]\cap H_0([\![p]\!],\partial_0)\otimes[\![q]\!] = H_0([\![p]\!],\partial_0)$, and the Mayer-Vietoris exact sequence gives,

$$(1):\ \forall n\geq 2 \qquad 0 \longrightarrow H_n([\![p]\!],\partial_i)\oplus H_n(H_0([\![p]\!],\partial_0)\otimes[\![q]\!],\partial_i)$$

$$\searrow$$

$$H_n([\![p.q]\!],\partial_i) \longrightarrow 0$$

$$(2):\quad 0 \longrightarrow H_1([\![p]\!],\partial_i)\oplus H_1(H_0([\![p]\!],\partial_0)\otimes[\![q]\!],\partial_i) \longrightarrow H_1([\![p.q]\!],\partial_i)$$

$$\searrow^{[\partial_i]}$$

$$H_0([\![p]\!],\partial_0) \xrightarrow{j_*} H_0([\![p]\!],\partial_i)\oplus H_0(H_0([\![p]\!],\partial_0)\otimes[\![q]\!],\partial_i)$$

$$\searrow$$

$$H_0([\![p.q]\!],\partial_i) \longrightarrow 0$$

By the Künneth formula, $H_n(H_0([\![p]\!],\partial_0)\otimes[\![q]\!],\partial_i) = H_0([\![p]\!],\partial_0)\otimes H_n([\![q]\!],\partial_i)$ (for all $n\geq 0$). Therefore, by (1),

$$\forall n\geq 2,\quad H_n([\![p.q]\!],\partial_i) = H_n([\![p]\!],\partial_i)\oplus H_0([\![p]\!],\partial_0)\otimes H_n([\![q]\!],\partial_i)$$

Then, examining (2), we have two cases,

- $\partial_i = \partial_0$: j_* is the identity on $H_0([\![p]\!],\partial_0)$. Therefore $Ker\, j_* = 0 = Im\,[\partial_0]$ so $H_1([\![p.q]\!],\partial_0) = H_1([\![p]\!],\partial_0)\oplus H_0([\![p]\!],\partial_0)\otimes H_1([\![q]\!],\partial_0)$. Then quotienting the second part of the exact sequence (2) by $H_0([\![p]\!],\partial_0)$ leads to $H_0([\![p.q]\!],\partial_0) = H_0([\![p]\!],\partial_0)\otimes H_0([\![q]\!],\partial_0)$.
- $\partial_i = \partial_1$: j_* is an isomorphism between $H_0([\![p]\!],\partial_0)$ and $H_0([\![p]\!],\partial_0)\otimes H_0([\![q]\!],\partial_1)$. Therefore, we have the same result as previously for $H_1([\![p.q]\!],\partial_1)$. Then we quotient the second half of the exact sequence (2) by $H_0([\![p]\!],\partial_0)\otimes H_0([\![q]\!],\partial_1)$ to get $H_0([\![p.q]\!],\partial_1) = H_0([\![p]\!],\partial_1) = (1)$.

(vi) We have $[\![p\|q]\!] = [\![p]\!] \otimes [\![q]\!]$. Therefore, (vi) is a direct consequence of the Künneth formula.

(vii) We first compute the homology of (Q^2, ∂) given the ones of (Q^1, ∂), (Q, ∂), and $(Q^1 \cap Q^2, \partial)$. The first case which is of interest for us is the one when we deal with bicomplexes, $\partial = \partial_0$, and we have a projection[2] p whose kernel is a sub-R-module of the set of 1-transitions Q_1, $Q^1 = Ker\ p \oplus \partial(Im\ p) \oplus \partial_1(Im\ p)$, $Q^2 = Q/Ker\ p$. The following result corresponds to the simpler case $Ker\ p = (a)$, a is not in the boundary of any 2-transition. We prove,

- $\forall k \geq 2, H_k(Q^2, \partial) = H_k(Q, \partial)$
- for $k = 0, 1$, we have two cases:
  - **(1)** $\nexists X \neq a, \partial(X) = \partial(a)$, then
    - $H_1(Q^2, \partial) = H_1(Q, \partial)$
    - $H_0(Q^2, \partial) = H_0(Q, \partial) \oplus (\partial(a))$
  - **(2)** $\exists X \neq a, \partial(X) = \partial(a)$, then
    - $H_1(Q^2, \partial) = H_1(Q, \partial)/R\text{-}Mod\{a - X/\partial(X) = \partial(a), X \neq a\}$
    - $H_0(Q^2, \partial) = H_0(Q, \partial)$

We have the Mayer-Vietoris sequence,

$$\cdots \longrightarrow H_n(Q^1) \oplus H_n(Q^2) \xrightarrow{i_*} H_n(Q) \xrightarrow{[\partial]} H_{n-1}(Q^1 \cap Q^2) \xrightarrow{j_*} H_{n-1}(Q^1) \oplus H_{n-1}(Q^2) \xrightarrow{i_*} \cdots$$

But, $Q^1 \cap Q^2 = \partial(Ker\ p) \oplus \partial_1(Ker\ p) = \{\partial(a), \partial_1(a)\}$ is composed only of two states. Therefore $\forall n \geq 2, H_{n-1}(Q^1 \cap Q^2) = 0$. This implies

$$\forall k \geq 2, H_k(Q, \partial) = H_k(Q^1, \partial) \oplus H_k(Q^2, \partial) = H_k(Q^2, \partial)$$

Now, noticing that $H_0(Q^1, \partial) = (\partial_1(a))$, $H_0(Q^1 \cap Q^2, \partial) = (\partial(a)) \oplus (\partial_1(a))$ and $H_1(Q^1, \partial) = 0$, we have a long exact sequence,

$$0 \longrightarrow H_1(Q^2, \partial) \xrightarrow{i^2_*} H_1(Q, \partial) \xrightarrow{[\partial]} (\partial(a)) \oplus (\partial_1(a)) \xrightarrow{j_*} (\partial_1(a)) \oplus H_0(Q^2, \partial) \xrightarrow{i_*} H_0(Q, \partial) \longrightarrow 0$$

We have $j_*(\partial(a)) = j^1_*(\partial(a)) \oplus j^2_*(\partial(a)) = 0 \oplus j^2_*(\partial(a))$.

But $Ker\ j_* = Im\ [\partial]$, so if $\exists X \neq a, \partial(X) = \partial(a)$ (case (2)), then $X - a \in H_1(Q, \partial)$ because a is not in the boundary of any 2-transition, and $[\partial](X - a) = \partial(a) \in Ker\ j_*$, thus $j^2_*(\partial(a)) = 0$ and $j_*(\partial(a)) = 0$.

[2] We recall that a projection is an idempotent endomorphism on a R-module V, uniquely characterized by its kernel (or its image because $V = Im\ p \oplus Ker\ p$).

Otherwise (case (1)), $\nexists X \neq a, \partial(X) = \partial(a)$, and $j^2_*(\partial(a)) = \partial(a) = j_*(\partial(a))$.
We have also $j_*(\partial_1(a)) = j^1_*(\partial_1(a)) \oplus j^2_*(\partial_1(a)) = \partial_1(a) \oplus x$, x verifying $i_*(\partial_1(a) \oplus x) = \partial_1(a) + x = 0$, so $x = -\partial_1(a)$.
Therefore, in case (1), $Ker\ j_* = 0$, and in case (2), $Ker\ j_* = (\partial(a))$. Factoring[3], we get a short exact sequence for each case:

(1)

$$\begin{array}{ccccccc} 0 & \longrightarrow & (\partial(a)) \oplus (\partial_1(a)) & & & & \\ & & \searrow^{j_*} & & & & \\ (\partial_1(a)) \oplus H_0(Q^2,\partial) & & \xrightarrow{i_*} & & H_0(Q,\partial) & \longrightarrow & 0 \end{array}$$

(2)

$$\begin{array}{ccccccc} 0 & \longrightarrow & (\partial_1(a)) & & & & \\ & & \searrow^{j_*} & & & & \\ (\partial_1(a)) \oplus H_0(Q^2,\partial) & & \xrightarrow{i_*} & & H_0(Q,\partial) & \longrightarrow & 0 \end{array}$$

We deduce that in case (1), $H_0(Q^2,\partial) = H_0(Q,\partial) \oplus (\partial(a))$, and in case (2), $H_0(Q^2,\partial) = H_0(Q,\partial)$.
Considering the left-hand side of the long exact sequence of the beginning, we have short exact sequences for each case:

(1)

$$0 \longrightarrow H_1(Q^2,\partial) \xrightarrow{i^2_*} H_1(Q,\partial) \xrightarrow{[\partial]} 0$$

(2)

$$0 \longrightarrow H_1(Q^2,\partial) \xrightarrow{i^2_*} H_1(Q,\partial) \xrightarrow{[\partial]} (\partial(a)) \longrightarrow 0$$

Thus, in case (2), $H_1(Q^2,\partial) = H_1(Q,\partial)/[\partial]^{-1}(\partial(a)) = H_1(Q,\partial)/R - Mod\{a - X / \partial(X) = \partial(a), X \neq a\}$, and in case (1), $H_1(Q^2,\partial) = H_1(Q,\partial)$.

More generally, let Q be a bicomplex. Let ∂ be one of its boundary operators. We consider the projection p from Q to Q, with $Ker\ p = (A)$, A being a n-transition ($n \geq 2$) of Q which is not is the boundary of any $(n+1)$-state. Let Q^1 be the smallest subHDA of Q containing $Ker\ p$, $Q^2 = Q/Ker\ p$. Then

$$\begin{array}{l} H_0(Q^1,\partial) = (\delta) \\ \forall i \geq 1, \quad H_i(Q^1,\partial) = 0 \\ H_0(Q^1 \cap Q^2,\partial) = (\delta) \\ H_{n-1}(Q^1 \cap Q^2,\partial) = (x) \\ \forall i \geq 1,\ i \neq n-1, \quad H_i(Q^1 \cap Q^2,\partial) = 0 \end{array}$$

[3] i.e. considering the quotient map induced by $j_*\ j^*$ from $(\partial(a) \oplus \partial_1(a))/(Ker\ j_*)$ to $(\partial_1(a)) \oplus H_0(Q,\partial)$.

And then,

$$\forall k, \quad k \geq n+1 \quad \vee \quad 0 \leq k \leq n-2 \quad H_k(Q^2) = H_k(Q)$$
$$Ker\ j_*^{n-1} = 0 \Longrightarrow (H_{n-1}(Q^2) = H_{n-1}(Q) \oplus (x) \quad \wedge \quad H_n(Q^2) = H_n(Q))$$

otherwise,

$$Ker\ j_*^{n-1} = (x) \Longrightarrow \Big(H_{n-1}(Q^2) = H_{n-1}(Q) \quad \wedge \quad H_n(Q^2) = H_n(Q)/[\partial]^{-1}(x)\Big)$$

(the condition $Ker\ j_*^{n-1} = (x)$ means that there exists a branching of dimension n in Q at x). This is due to the fact that the $H_k(Q, \partial)$ are the solutions of the following equations (Mayer-Vietoris):

$$(ES_1) : 0 \longrightarrow H_1(Q^2) \xrightarrow{i_*^1} H_1(Q) \xrightarrow{[\partial]^1} (\delta) \xrightarrow{j_*^0} (\delta) \oplus H_0(Q^2) \xrightarrow{i_*^0} H_0(Q) \longrightarrow 0$$

$$(ES_k) : 0 \longrightarrow H_k(Q^2) \xrightarrow{i_*^k} H_k(Q) \longrightarrow 0 \quad \forall k, \quad k \geq n+1 \quad \vee \quad 2 \leq k \leq n-2$$

$$(ES_n) : 0 \longrightarrow H_n(Q^2) \xrightarrow{i_*^n} H_n(Q) \xrightarrow{[\partial]^n} (x) \xrightarrow{j_*^{n-1}} H_{n-1}(Q^2) \xrightarrow{i_*^{n-1}} H_{n-1}(Q) \longrightarrow 0$$

By (ES_k) we immediately conclude

$$\forall k, \quad k \geq n+1 \quad \vee \quad 2 \leq k \leq n-2 \quad H_k(Q^2) = H_k(Q)$$

If we examine (ES_1), we have to notice that $j_*^0(\delta)$ cannot be equal to zero, thus (ES_1) splits into two short exact sequences, giving the result for $k = 0$ and $k = 1$.

Then, for (ES_n), we have a discussion on j_*^{n-1}. We have two cases, in which the sequence splits into two short exact sequences. In the first case: $Ker\ j_*^{n-1} = 0$, and we have:

$$0 \longrightarrow (x) \xrightarrow{j_*^{n-1}} H_{n-1}(Q^2) \xrightarrow{i_*^{n-1}} H_{n-1}(Q) \longrightarrow 0$$
$$0 \longrightarrow H_n(Q^2) \xrightarrow{i_*^n} H_n(Q) \longrightarrow 0$$

And in the second case, $Ker\ j_*^{n-1} = (x)$, and we have:

$$0 \longrightarrow H_{n-1}(Q^2) \xrightarrow{i_*^{n-1}} H_{n-1}(Q) \longrightarrow 0$$
$$0 \longrightarrow H_n(Q^2) \xrightarrow{i_*^n} H_n(Q) \xrightarrow{[\partial]^n} (x) \longrightarrow 0$$

Combining all these results proves (vii).

(viii) This is a direct consequence of Lemma 2.

□

6.5 Application: Semantic Equivalences

In this section, we make the first move from local geometric properties like branchings and mergings to geometric properties of paths of HDA. We will show that the local properties provide valuable information about these global geometric properties like branching-time semantic equivalences. The methodology we are adopting is to define properties combinatorially (using semi-regular, partial or regular HDA), because they are closer to the computer scientific intuition and then use general HDA and local invariants to characterize them.

To get usual definitions of semantic equivalences to work (as those in [vG90]) which are made using ordinary transition systems we give a notation for "transitions" read in paths,

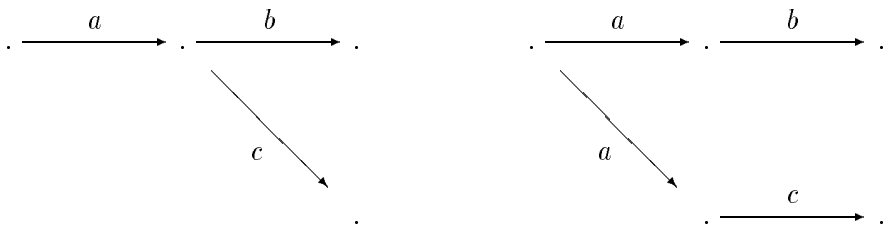
Definition 37 *Let $\Psi(Q)$, for $l : Q \to L$ a labeled HDA, be the set of paths (partial or total) of Q. Let $\to$ be the following subset of $\Psi(Q) \times L \times \Psi(Q)$: $p \xrightarrow{\sigma} p'$ if and only if $p = (p_1, ..., p_n)$, $p' = (p_1, ..., p_n, q)$ with $l(q) = \sigma$. As usual, we define $\xrightarrow{\sigma,*}$ to be the transitive closure of the relation $\xrightarrow{\sigma}$ on $\Psi(Q)$, and $\xrightarrow{*}$ to be the transitive closure of the relation $\xrightarrow{L}$ defined by $p \xrightarrow{L} p' \Leftrightarrow \exists \sigma$, $p \xrightarrow{\sigma} p'$.*

Using these two definitions of paths, we can generalize the notions of trace equivalence, failure equivalence, ready set equivalence etc. Except for the first one, all these semantic equivalences are "branching-time" equivalences. We refer to [vG90] for details about the standard definitions of semantic equivalences for parallel programs.

6.5.1 Linear-time semantic equivalences

Definition 38 *(Trace semantics)*
For M a labeled HDA over L, with labelling l, let $\mathcal{P}(M)$ be the set of all semi-partial paths of M. Then, two labeled HDA (over L) (P,l), (Q,k) are trace equivalent if and only if $l(\mathcal{P}(P))=k(\mathcal{P}(Q))$.

As we are to do in the following, this definition can specialize to one in which we only consider "homogeneous" paths, i.e. $l : M \to L$ and $k : P \to L$ are H-trace equivalent if and only if $l(P_n(M)) = k(P_n(P))$ for all $n \geq 1$. Both definitions quite obviously generalize ordinary trace equivalence. For instance, the two following automata are trace and H-trace equivalent,



Trace equivalence does not respect the branchings nor the mergings. It is a linear-time semantic equivalence.

6.5.2 Branching-time equivalences

Failure equivalence

Definition 39 *Let σ be a semi-partial path of a labeled HDA (M, l) over L, and X be a set of states and transitions of L. Then $(l(\sigma), X)$ is a failure pair for M if and only if σ cannot be extended by firing transitions whose labels are not in X. Two labeled automata P and Q are failure equivalent if and only if their sets of failure pairs are equal.*

Consider the failure pairs $(l(\sigma), X)$ with X, set of 1-transitions of L. Take the smallest such set X. If it is void, then this means that σ_n[4] is a final state of M. Reciprocally, if we have a failure pair of the form $(l(\sigma), \emptyset)$ then σ_n is a final state. This proves that $l_*(H_0(M, \partial_0))$ is preserved by failure equivalence.

Now, suppose it is not void, and contains two distinct 1-transitions a and b. Then by hypothesis, there exists x and y with

$$\partial_0(x) = \partial_0(y) = \sigma_n$$

$$p(x) = a \quad p(y) = b$$

Thus, $x - y$ is a 1-cycle for ∂_0.

We take for granted now that (M, l) is a labeled automaton such that all states are reachable.

Suppose that M is a standard automaton, i.e. has only states and 1-transitions. It follows that $x - y$ is a generator of $H_1(M, \partial_0)$ and $a - b$ is a generator of $l_*(H_1(M, \partial_0))$. Reciprocally, if we have a generator u of $l_*(H_1(M, \partial_0))$, then u can be taken as $l(x) - l(y)$, with x and y elements of the chosen basis of M, and $\partial_0(x) = \partial_0(y)$. Call this state α, and let σ be a semi-partial path of length n such that $\sigma_n = \alpha$. It is easy to see that there exists a failure pair (σ, X) with $\{x, y\} \subseteq X$. This implies that $l_*(H_1(M, \partial_0))$ is preserved under failure equivalence[5].

Readiness equivalence

Definition 40 *(Readiness equivalence)*
$(l(\sigma), X)$ is a ready pair for an HDA M if and only if, σ is a semi-partial path of length n, X is a set of states and transitions of L such that σ can only be extended by elements of X. P and Q are ready equivalent if they have the same sets of ready pairs.

As for failure equivalence, we see that $l_*(H_0(M, \partial_0))$ is preserved, and for standard automata, $l_*(H_1(M, \partial_0))$ is also preserved.

[4] n is the length of σ.

[5] This is the basic requirement for being called branching-time semantic equivalence

Bisimulation equivalence

First Approach - inhomogeneous paths Here we follow the lines of [GJ92].

Definition 41 *S is a bisimulation between (Q,l) and (Q',l') if:*

- *S is a relation between the states, events and transitions of Q and Q'*
- *initial states are related to initial states*
- *$(s,s') \in S \Rightarrow$ ($\forall$ q a path for Q such that $\exists i$, $q_i = s$, $\exists q'$ a path for Q' such that $\exists j$, $q'_j = s'$ and $(q_{i+1}, q'_{j+1}) \in S$, $l(q_{i+1}) = l(q'_{j+1})$)*
- *$(s,s') \in S \Rightarrow$ ($\forall q'$ a path for Q' such that $\exists j$, $q'_j = s'$, $\exists q$ a path for Q such that $\exists i$, $q_i = s$ and $(q_{i+1}, q'_{j+1}) \in S$, $l(q_{i+1}) = l(q'_{j+1})$)*

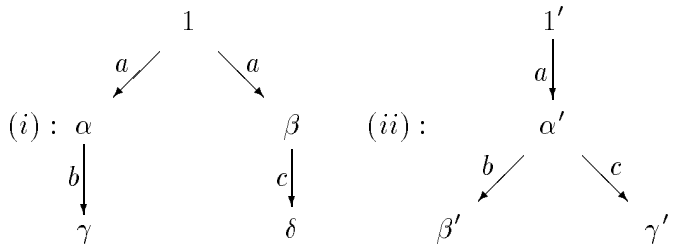
(Q,l) and (Q',l') are bisimulation equivalent if and only if there exists a bisimulation between them.

This notion of bisimulation equivalence "naturally" generalizes the usual notion (as found in [Mil89]) of observational equivalence, or bisimulation equivalence on one dimensional automata. Two HDA are bisimulation equivalent if and only if each time one can fire a transition (of any dimension) then the other can fire the same. This bisimulation implies that not only are we looking at the time choices are made for firing ordinary transitions, but also we are looking at the allocation of the different actions through time. This is what we are going to prove in the rest of this section.

One can verify that bisimulation equivalence implies readiness equivalence, which in turn implies failure equivalence and then trace equivalence. Notice also that for standard automata, these notions coincide with the usual ones. We have just added the opportunity to observe simultaneous actions we could not before, because we could not express them in the semantics. This means that we can observe schedules on any number of processes.

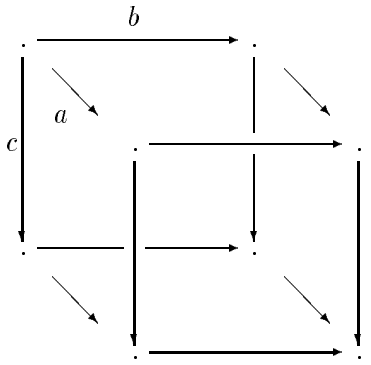
In our setting, the description of bisimulation equivalence is more complex than in the sequential case. Nevertheless, we can show that it is still a branching (in our sense) time equivalence, that is, it locally preserves some geometric shapes.

Example 30 • *The labeled HDA represented as,*

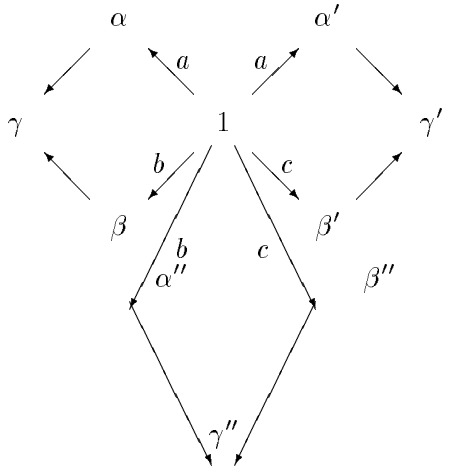


are not bisimulation equivalent since in (i) the choice of a copy of a implies that we have already chosen if we will do b or c whereas in (ii) this choice is made after firing a. Notice that (i) (see the semantics of CCS, Chapter 5) corresponds to the CCS term $a.b + a.c$ whereas (ii) is $a.(b + c)$. This is the classical example in CCS $a.(b + c) \neq a.b + a.c$.

- *The labeled HDA (i) represented as,*



and the one (ii) represented as,



where all the squares are filled in are not bisimulation equivalent since in (ii), choosing a transition a imposes which transition we can fire concurrently in the future whereas in (i), this choice is not yet done. Looking at the semantics of CCS of Chapter 5, the reader should be able to convince himself that (ii) corresponds to the term $a\|b + b\|c + c\|a$ and that (i) corresponds to no term (this will be proved later on).

We begin by looking at bisimulation equivalence for semi-regular HDA.

Definition 42 *Let M be a semi-regular HDA and x a state of M. The local skeleton of dimension n at x is $V_n(x)$ subHDA of M generated by,*

$$W_n(x) = \{y \in M / dim y = k \leq n \wedge d^0_0 d^0_1 \ldots d^0_{k-1}(y) = x\}$$

Let V_x be the amalgamated sum of all the $V_n(x)$, $n \in \mathbb{N}$. The first n (if it exists) such that $V_n(x) = V_{n+1}(x)$ is called the local dimension of M at x. By convention, if no such n exists, we say that the local dimension is infinite.

Now we state the result which proves that bisimulation equivalence preserves local branchings:

Lemma 20 *(Local test)*
Let (P,l) and (Q,k) be two labeled semi-regular path-connected HDA over L. If they are bisimilar then $\forall x$, $\exists y$, and $\forall y$, $\exists x$, with semi-partial paths p_x (resp. p_y) ending at x (resp. y) such that $l(p_x) = k(p_y)$ in both cases and $H_*(l(V_x(P)), \partial_0)) = H_*(k(V_y(Q)), \partial_0))$.

PROOF. Let S be a bisimulation between P and Q. Let x be an element of P. P is path-connected, so there exists a path σ connecting an initial state i of P to x. By induction on the length of σ, we show that there exists τ, a path connecting an initial state j of Q to a state y, with $\sigma S \tau$. Now, we show that V_x and V_y considered as labeled HDA are bisimulation equivalent.
Now, we prove that $l(V_x(P)) = l(V_y(Q))$. It suffices to show that for all n, $l(W_n(x)) = l(W_n(y))$, the result being entailed by considering the smallest HDA generated by both terms of the equality. Let $t \in W_n(x)$ be a transition of dimension $k \leq n$. By definition of $W_n(x)$, there exists a path (by "maximal allocation" of processors) from x to t, e.g. $p = (x = d^0_{k-1} \ldots d^0_0(t), d^0_{k-2} \ldots d^0_0(t), \ldots, d^0_0(t), t)$. p can be decomposed, using the transition relation $\rightarrow$ as,

$$p^0 = (x) \xrightarrow{\mathrm{l(d^0_{k-2} \ldots d^0_0(t))}} p^1 = (x, d^0_{k-2} \ldots d^0_0(t)) \longrightarrow \ldots p^{k-1} = (x, \ldots, d^0_0(t)) \xrightarrow{\mathrm{l(t)}} p^k = p$$

As $\sigma S \tau$, and as σ, τ end at x and y respectively, we can associate (by induction on j) to the p^j paths q^j beginning at y with,

$$p^j S q^j$$

$$q^0 = (y) \xrightarrow{\mathrm{l(d^0_{k-2} \ldots d^0_0(t))}} q^1 = (x, d^0_{k-2} \ldots d^0_0(t)) \longrightarrow \ldots q^{k-1} = (x, \ldots, d^0_0(t)) \xrightarrow{\mathrm{l(t)}} q^k$$

Therefore, $l(t) \in l(Q)$. Moreover, the construction of the path q^k above (by "maximal allocation") proves that there exists a transition of dimension k of $W_n(y)$ which has label $l(t)$. Exchanging the roles of P and Q gives $l(W_n(x)) = k(W_n(y))$.
Now, $\forall n$, $\forall k < n$, $H_k(l(V_n(x)), \partial_0) = H_k(l(V_x), \partial_0) = H_k(k(V_n(y)), \partial_0) = H_k(k(V_y), \partial_0)$. □

This test shows that under the observation of labels, bisimulation equivalent automata are locally isomorphic. As we deal with local shapes, we try to classify them using homology. The way we extract these local shapes implies

that only the homology with respect to ∂_0 (branchings) is relevant: for all k and x, $H_k(l(V_x) = V_{l(x)}[l(V_x)], \partial_0) \subseteq H_k(l(P), \partial_0)$. Thus, $H_k(l(.), \partial_0)$ (for all k) is invariant under bisimulation equivalence. This is then a test for bisimulation for general HDA.

Now, notice that this test is still not powerful enough for our purpose. For instance, the test of the previous proposition distinguishes the first two HDA of Example 30 but not the last two ones. We need to generalize what we have done.

First, we extend the concept of local skeleton. Let X be a semi-regular HDA and a a m-transition of X (and not only a state as we had before). Then the local skeleton of dimension n at a is $V_n(a)$ subHDA of X generated by,

$$W_n(a) = \{y \in X / dim\ y \leq n \wedge d^0_{k-1} \ldots d^0_m(y) = a\}$$

V_a is then the amalgamated sum of all the $V_n(a)$.

Then,

Lemma 21 *(Generalized local test)*
Let (P, l) and (Q, k) be two labeled semi-regular path-connected HDA over L. If they are bisimilar then $\forall x$ transition, $\exists y$ transition, and $\forall y$, $\exists x$, with semi-partial paths p_x (resp. p_y) ending at x (resp. y) such that $l(p_x) = k(p_y)$ in both cases and $H_*(l(V_x(P)), \partial_0)) = H_*(k(V_y(Q)), \partial_0))$.

PROOF. Similar proof as for the local test. □

Now, this is enough for proving that in the second example of Example 30, the two HDA are not bisimulation equivalent. How can we decide this generalized local test, knowing the physical branchings, the labelling and the shape of the domain?

Unfortunately, the more powerful test (as well as easier to decide) which would be $l(H_*(M, \partial_0)) = k(H_*(M, \partial_0))$ is not true in general when P and Q are bisimilar. Consider for instance $P = (a \otimes a') + (a' \otimes a'') + (a'' \otimes a)$ and $Q = (a \otimes a')$ where a, a' and a'' are three 1-transitions. Suppose $k = l$, $l(a) = l(a') = l(a'')$. P and Q are bisimulation equivalent, but $l(H_2(P, \partial_0)) = (a \otimes a)$ and $k(H_2(Q, \partial_0)) = 0$.

We have to make a few assumptions on the labelling to relate these two "local" tests. An important class of labellings is the one seen in Chapter 2. Let $(a_i), (b_i), \ldots (i \in \mathbb{N})$ be 1-transitions and D be the domain given by the recursive domain equation,

$$D \cong \sum_i ((a_i) + (b_i) + \ldots) + D \otimes D$$

Then define a morphism from D to D by $l(a_i) = a_0$, $l(b_i) = b_0$, ..., and $l(x \otimes y) = l(x) \otimes l(y)$. We set $L = l(D)/\{s - 1/s \in D_{p,-p}, p \in \mathbb{Z}\}$. Let l be induced morphism from D to L. Then for such a labelling,

Claim 1 *Let $k \in \mathbb{N}$, X, Y subHDA of D and x, y states of X, Y respectively. Then,*

$$\forall (u,v) \in V_x(X) \times V_y(Y),$$

$$H_k(l(V_u(X)),\partial_0) = H_k(l(V_v(Y)),\partial_0) \Rightarrow \begin{cases} l^*(H_k(V_x(X),\partial_0))/\{a_0^{\otimes^k}, b_0^{\otimes^k}, \ldots\} \\ = \\ l^*(H_k(V_y(Y),\partial_0))/\{a_0^{\otimes^k}, b_0^{\otimes^k}, \ldots\} \end{cases}$$

Therefore $l(H_k(.,\partial_0))/\{a_0^{\otimes^k}, b_0^{\otimes^k}, \ldots\}$ is invariant under bisimulation equivalence.

Notice that, as we have higher-order automata (the $Hom(P,Q)$ in Chapter 4), this property allows also for an immediate generalization, which was far from obvious for bisimulation and higher-order bisimulation.

As an application of Lemma 20, we show that some behaviours modulo bisimulation equivalence are not implementable using CCS.

Lemma 22 *There exists an element in the semantic domain D we have used previously for giving semantics to CCS terms, which is not bisimulation equivalent to any term of CCS.*

PROOF. ($R = \mathbb{Z}_2$)
Consider the HDA t generated by the three 2-transitions $a \otimes b$, $a \otimes c$ and $c \otimes b$. The reader can verify that $t = (1) \oplus (\partial_1(a)) \oplus (\partial_1(b)) \oplus (\partial_1(c)) \oplus (\partial_1(a) \otimes \partial_1(b)) \oplus (\partial_1(a) \otimes \partial_1(c)) \oplus (\partial_1(c) \otimes \partial_1(b)) \oplus (a) \oplus (a \otimes \partial_1(b)) \oplus (a \otimes \partial_1(c)) \oplus (b) \oplus (\partial_1(a) \otimes b) \oplus (\partial_1(c) \otimes b) \oplus (c) \oplus (c \otimes \partial_1(a)) \oplus (c \otimes \partial_1(b)) \oplus (a \otimes b) \oplus (a \otimes c) \oplus (c \otimes b)$.

We see that $a \otimes b + a \otimes c + c \otimes b$ is in $H_2(t,\partial_0)$, because $\partial_0(a \otimes b) = a + b$, $\partial_0(a \otimes c) = a + c$, and $\partial_0(c \otimes b) = c + b$, so a, b, and c are counted twice in $\partial_0(a \otimes b + a \otimes c + c \otimes b)$, and also there is no 3-transitions from which $a \otimes b + a \otimes c + c \otimes b$ could be the boundary of.

Now, let z be any CCS-term. We show that no element of $l_*(H_2([\![z]\!],\partial_0))$ can be of the form $a \otimes b + a \otimes c + b \otimes c$, then t is not bisimulation equivalent to z by Lemma 20. Assume we have a CCS-term z bisimulation equivalent to t.

First of all, we can restrict ourselves to considering terms z only built with actions a, b, and c, and operators $+$, $.$, and $\|$. Moreover, no two operators $|$ can be nested (because we would have 3-transitions, and t does not contain any). Let a_i, b_i, and c_i for a certain number of i's, be physical copies of actions a, b, and c respectively, appearing in $[\![z]\!]$ (coded during the translation of the term z by forming the tensor product of a, b, and c with states)

Let x be an element of $l_*(H_2([\![z]\!],\partial_0))$. x corresponds to l_* of some sub-vector-space of $H_2([\![z]\!],\partial_0)$. But by the remarks we have made and the Künneth formula (appearing during the computation of the homology of $|$-terms), x is of the form $\sum_{u,v} l_*(u \otimes v)$, where $u \in H_1(Q,\partial_0)$ and $v \in H_1(Q',\partial_0)$, for some automata Q, and Q' built with a, b, c, $+$, etc.

A straightforward application of Section 5.4.2 shows that necessarily, u and v are of the form $a_i + a_j$, $a_i + b_j$, $a_i + c_j$, $b_i + b_j$, $b_i + c_j$ or $c_i + c_j$. Therefore, x is of the form $a \otimes a$, $a \otimes a + a \otimes b$, $a \otimes a + a \otimes c$, $a \otimes b$, $a \otimes b + a \otimes c$, $a \otimes c$, $a \otimes a + a \otimes b + b \otimes b$, $a \otimes a + a \otimes c + a \otimes b + b \otimes c$, $a \otimes b + b \otimes b$, $a \otimes b + a \otimes c + b \otimes b + b \otimes c$, $a \otimes c + b \otimes c$, and all other terms obtained by a suitable permutation on the symbols a, b, and c. We see that we cannot obtain $a \otimes b + a \otimes c + b \otimes c$ from these terms. This is a contradiction. □

The term we have exhibited has an interest of its own. It is the dynamic allocation on two processors of three processes (or the typical branching of dimension 2 of Section 6.1.4).

Second Approach - homogeneous paths Now the observable paths for bisimulation are restricted to homogeneous paths in some $P_n(M)$, the R-module of n-paths of M. We authorize observation of finite sets of paths at the same time (use of the formal sum of paths, i.e. the addition in the modules of n-paths).

Definition 43 *Let $l : M \to L$ and $k : N \to L$ be two labeled HDA. Then a H_n-bisimulation ($n \geq 0$) R between M and N is a relation between n-transitions of M and n-transitions of N such that,*

(i) aRa' *and* bRb' *implies* $(a + b)R(a' + b')$,

(ii) aRa' *and* $\lambda \in R$ *implies* $\lambda aR\lambda a'$,

(iii) $0\ R\ 0$,

(iv) initial states of M are related to initial states of N,

(v) aRa' *and* $\exists p \in P_n(M)$ *with* $\partial_0(p_i) = a$ *(for some i), then there exists* $q \in P_n(N)$ *with* $\partial_0(q_j) = a'$ *(for some j),* $k(q_j) = l(p_i)$ *and* $\partial_1(p_i)R\partial_1(q_j)$,

(vi) aRa' *and* $\exists q \in P_n(N)$ *with* $\partial_0(q_j) = a'$ *(for some j), then there exists* $p \in P_n(M)$ *with* $\partial_0(p_i) = a$ *(for some i),* $l(p_i) = k(q_j)$ *and* $\partial_1(p_i)R\partial_1(q_j)$.

Let S_n be the following "shift operator". For M a HDA, $S_n(M)_{p,q} = M_{p+n,q}$ and $\partial_0[S_n(M)] = \partial_0$, $\partial_1[S_n(M)] = \partial_1$. S_n is easily seen to define an endofunctor in Υ. A similar definition could be given for semi-regular, partial and regular HDA.

The nice thing about H_n-bisimulation is,

Lemma 23 *The set of H_n-bisimulation between M and N is in one-to-one correspondance with the set of H_1-bisimulation between $S_{n-1}(M)$ and $S_{n-1}(N)$.*

PROOF. Easy verification since $P_n(M) = P_1(S^{n-1}(M))$. □

As usual we say that M and N are H_n-bisimulation equivalent if and only if there exists a H_n-bisimulation between them. The previous lemma shows that M and N are H_n-bisimulation equivalent if and only if $S_{n-1}(M)$ and $S_{n-1}(N)$ are H_1-bisimulation equivalent. We are now concentrating on H_1-bisimulation equivalence.

Lemma 24 *If M and N are H_1-bisimulation equivalent then*

$$l(Ker\ \partial_0^1[M]) = k(Ker\ \partial_0^1[N])$$

SKETCH OF PROOF. Let $c \in Ker\ \partial_0^1[M]$. Then $c \in P_1(M)$. 0 is related by the bisimulation R to 0 by (iii) of Definition 43. Therefore there exists $c' \in N_1$ such that $l(c) = k(c')$. □

This implies that if looking at H-bisimulation equivalence (i.e. the smallest equivalence subsuming all the H_n, $n \geq 1$) between schedulers of M (see Chapter 7) and schedulers of N then it implies that $l_*(H_*(M, \partial_0)) = k_*(H_*(N, \partial_0))$, that is the preservation of the labels of the branchings.

Another way (closer to the ordinary testing methodology, [DNH83]) to obtain the result is to remark that conditions (i), (ii) and (iii) make the bisimulation a submodule B of $M \times N$. Now, conditions (v) and (vi) show that as soon $(\partial_0(t), \partial_0(t')) \in B$, where $t \in M_1$, $t' \in N_1$ and $l(t) = k(t')$, then $(\partial_1(t), \partial_1(t')) \in B$. If we notice that B is a submodule of the pushout (synchronized product),

$$\begin{array}{ccc} M \times_L N & \xrightarrow{p_1} & M \\ \downarrow{\scriptstyle p_2} & & \downarrow{\scriptstyle l} \\ N & \xrightarrow{k} & L \end{array}$$

and that in $M \times_L N$ this precisely means that $\partial_0(t, t')$ and $\partial_1(t, t')$ are $(\partial_0 - \partial_1)$-connected, we get the following result (generalized easily to any dimension).

Lemma 25 *Suppose M and N are two connected HDA. Then M and N are H_n-bisimulation equivalent if and only if $p_1(C_0) = M_0$ and $p_2(C) = N_0$ where C is the connected component C of (I, I') in $M \times_L N$.*

C represents the maximal bisimulation. The lemma can be interpreted (computer-scientifically) as saying that when synchronizing M and N (over their labels) all states of M and N should be reached from the initial state (I, I').

Finally, we can notice that M and N are H-bisimulation equivalent implies M and N are bisimulation equivalent. Since H-bisimulation is easier to prove or to negate, the previous proposition lets us think that H-bisimulation should better be used in practical semantic definitions.

Summary We showed that many dynamic properties of interest in HDA were in fact local geometric properties, that could be computed or characterized using homology theory. Among them were initial and final states, deadlocks and initial deadlocks, divergence, branchings and mergings. We have used classical results from homology theory to compute these geometric properties inductively on the syntax of CCS. We ended up by showing that some "branching-time" semantic equivalences were preserving branchings indeed. We introduced two kinds of "higher-dimensional" bisimulations for HDA and showed that they were preserving some local geometric shapes as well, hence giving a semi-decision procedure for proving that two HDA are not bisimulation equivalent. As an application, we proved that no CCS term could be bisimulation equivalent to the dynamic allocation on two processors of three processes.

Chapter 7

Serialization and schedulers

7.1 Introduction and motivation

7.1.1 Scheduling problems in computer science

The use of schedulers is somewhat pervasive to many branches of computer science. We mention below a few application areas, the properties that schedulers are to verify and give some references to the theoretical work done in these different fields.

Safety and efficiency of the implementation of concurrent languages

A real parallel machine has but a limited number of resources. It has limited memory, limited number of processing units and many constraints on the way it can use them. The idealistic view of true concurrency semantics, assuming an infinite number of processors for instance, is therefore misleading when it comes to runtime behaviour of programs. It may happen that to badly schedule spawning operations may deadlock (just delay in practice) a process that would need to synchronize with another (not yet executed) process. It may happen as well that some shared resources of the machine have to be used in **mutual exclusion**.

The safety (respectively efficiency) properties that schedulers must verify are mainly choosing behaviours that will not lead to deadlocks (respectively not delay too much the execution of some process) and implementing mutual exclusion of some resources. This last property could well be implemented by standard techniques (Peterson's algorithm for shared-variables or hardware test-and-set like operations) independently of programs but this would be at the expense of efficiency.

Let us take an example extracted from [HMC94] and [PF94]. Many modern CPUs like SPARCs or MIPS pipeline instructions. Of course, their functional units, registers or bus are all used in mutual exclusion. Unfortunately the pipelined instructions overlap in time as they use more than one clock cycle and some of them cannot be executed (unless "**structural hazards**" occur) within

Figure 7.1: MIPS R4000 floating point unit.

instructions/cycle	0	1	2	3	4	5
add.s	U	S+A	A+R	R+S	$\emptyset$	$\emptyset$
add.s		U	S+A	A+R	R+S	$\emptyset$

where U is unpack, S is shift, A is adder and R is round.

a certain number of cycles after some others (see Figure 7.1). We do not want to use the pipeline in mutual exclusion since we would have to empty it after every instruction. The problem addressed in [PF94] is to verify that schedulers for a single process ensure that structural hazards will not occur (safety). In a concurrent framework, if there are more processes than processors, we can address the new problem of finding a way to interleave actions from different processes executed on the same processor, that verify the constraints while using the pipeline at the best of its capabilities (see Example 31).

A similar example at a more macroscopic level is given by an I/O buffer shared by two or more processes. Some processors (like INTEL's Pentium) are even more complex to deal with since some resources may be used by at most two processes in parallel but not three[1].

This chapter is about the first mathematical definitions of schedulers within the HDA framework. We will see in Part IV how to get the best scheduler (or an approximation of it) using abstract interpretation.

Example 31 *Suppose that we want to execute two instructions add.s one after the other on the MIPS R4000 floating point unit*[2]*. Then at cycle 2 the adder A has to be used by both instructions (coming from the same thread). The same holds at cycle 3 for the round unit R. We say in that case that there is an hazard on A at cycle 2 and an hazard on R at cycle 3. A good scheduler should have prevented us from this situation by interleaving the two threads after the first add.s and continue with non-conflicting instructions of the second thread for the pipeline to be emptied a bit before executing the second add.s.*

Another example of scheduling properties can be found in the **parallelization** litterature. Given a sequential program p, can we decide which parts of it can be executed in parallel? This is dual to the problems we have described above. Parallelization is about relaxing the constraints on scheduling that a sequential program put arbitrarily. Most approaches up to now are based on program transformation [LZ93, PP92, SMC91]. We propose here a theoretical framework that enables us to derive a scheduler choosing dynamically the intructions to spawn. The actual algorithms derived from this framework will be developed in Part IV.

[1] It has two integer arithmetic units.

[2] Taken from [PF94].

Protocols in distributed/concurrent systems

In order to have well behaved distributed systems, one very often has to make local processors agree on some criterion, like elect a leading one or organize the flow of information to guarantee the coherence of the global state of the system (by local rules only) like the consensus, set or renaming **agreement tasks** [Her94]. This is done by defining protocols. An example of such a situation is given by a parallel machine whose different units communicate by asynchronous messages along channels which have a given topology (let us say a ring topology for instance). Now, a protocol for guaranteeing a global knowledge of some fact must serialize all message passing primitives according to the communication topology (in our example of the ring topology, messages are to be waited for from say the left neighbour before passing them to the right neighbour). In [Her94] some of these problems are addressed in a static manner (the topology is fixed once and for all). We propose here to use the dynamic semantics to deal with changing topologies as well[3].

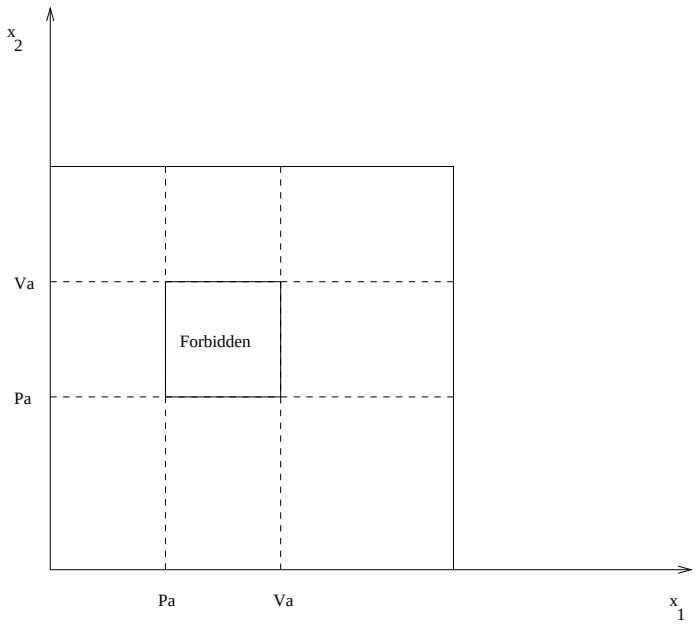
Another example can be found in concurrent database systems [Ull82]. A transaction in a database system is defined to be any query to the database, like reading or writing entries. The database itself is shared by many processes which are sequences of transactions. To ensure the consistency of the database the processes have to lock some entries and then unlock them after some of their transactions have been executed. Protocols define the way processes lock and unlock items. A good instance of this is the **two-phase protocol**. Given processes P_i accessing items in sets A_{P_i} the two-phase protocol consists in locking (giving exclusive access to the locking process) all items in P_i, before all transactions in P_i and then unlocking (releasing the unique access grants) all items in A_{P_i}. There again, the protocol is a constraint on scheduling. The notion of consistency of the database or soundness of the protocol is known as **serializability**. This means that all schedulers constrained by the protocol must be equivalent in some sense (at least give the same result). We give a general definition of serializability, carrying on the work presented in [Gou93], and give a practical test for protocols based on the semantics of the processes and not only on static or syntactic ground. Very recently, Jeremy Gunawardena [Gun94] has given a very clear explanation about why serializability has something to do with **homotopy**. We first recall what the problem is in concurrent databases and give a formalization of these notions using **higher-dimensional automata**.

7.1.2 A geometric approach

Interestingly enough, a graph-based criterion is known for serializability [Ull82]. In more general protocols for "decision problems" recent results [Her94] use combinatorial algebraic topology on static representations of protocols. We will show here that we can use more general tools from algebraic topology directly on

[3] In many languages, like CM-Fortran with the CMMD library, or CML [Rep92], channels are defined during the execution of the program. They are not physical but they are logical channels.

Figure 7.2: A process graph for two transactions accessing the same shared item.



the dynamic semantics of the systems studied to extract the information about serializability and about schedulers. We will develop in particular a **homotopy theory** of oriented paths (next section). Let us explain the intuition about it.

The two-phase protocol

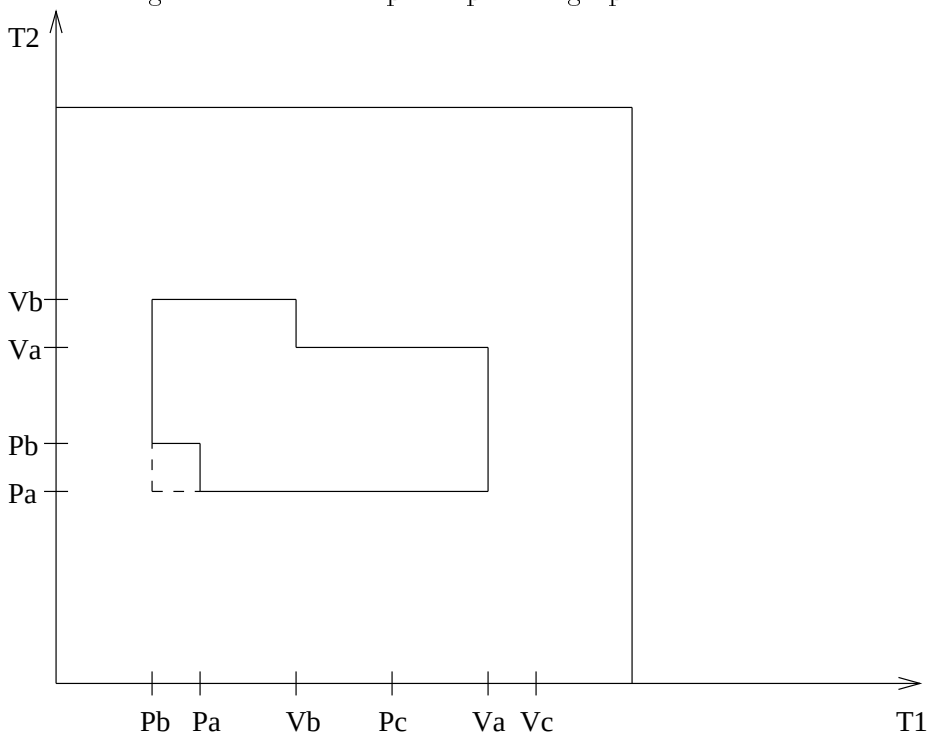
A concurrent database is composed of a set of shared objects, or *items*, and a set of processes accessing these items $T_1, \ldots, T_n$, or *transactions*. The transactions can be executed in parallel, and one can think of a good example (of economic interest too!) as being a reservation system of an airline. The items are seats in the planes and the transactions are individual queries from customers, made in parallel since there may be many different selling points. The basic property we want to insure is that no seat is sold twice (at the same time) to different customers. This is rather basic since we do not even ask for a priority rule like "first arrived, first served". In the shared memory paradigm the well-known method for attacking this is to put locks [Dij68] on shared variables. In Dijkstra's formalism, for an item a, Pa is the action of locking a and Va is the action of relinquishing the lock on a. As long as we are only interested in the policy of acquiring items and not in their actual values, we can abstract the transactions in such a way that they are written as strings of Px, Vx, x ranging over the shared objects. This is a more important abstraction than one may notice at first. We will come back to that when formalizing these notions.

As an example, consider

$$T_1 = PaVa$$

$$T_2 = PaVa$$

Figure 7.3: An example of process graph.



There is an old way to represent these transactions due to Dijkstra again (see [Dij68], look also at [Hoa85]), known as *process graphs*. We will see that it has much to do with the HDA approach. The idea is to associate to each transaction a "local time" which geometrically is one coordinate in an euclidean space. Supposing that all processes can individually terminate, we may normalize this local time for it to range over $[0,1]$. A purely asynchronous execution of n transactions is now any path from $(0,\ldots,0)$ to $(1,\ldots,1)$ in the n-cube $[0,1]^n$ where the local times, i.e. the coordinates always increase. But the executions are constrained by the fact that shared objetcs are accessed in mutual exclusion. In Figure 7.2 we have pictured the central square in $[0,1]$ which is forbidden: a valid path of execution cannot enter it since it is precisely the region in which both transactions access the same object a. The more complex example borrowed from [Gun94]:

$$T_1 = PbPaVbPcVaVc$$

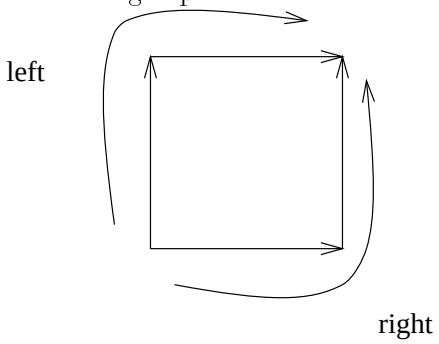
$$T_2 = PaPbVaVb$$

is pictured in Figure 7.3. Using these geometric representations, we have two main questions,

(i) Can the system of transactions deadlock?

(ii) Is the system of transactions correct in some sense?

Figure 7.4: Left and right paths in a mutual exclusion.

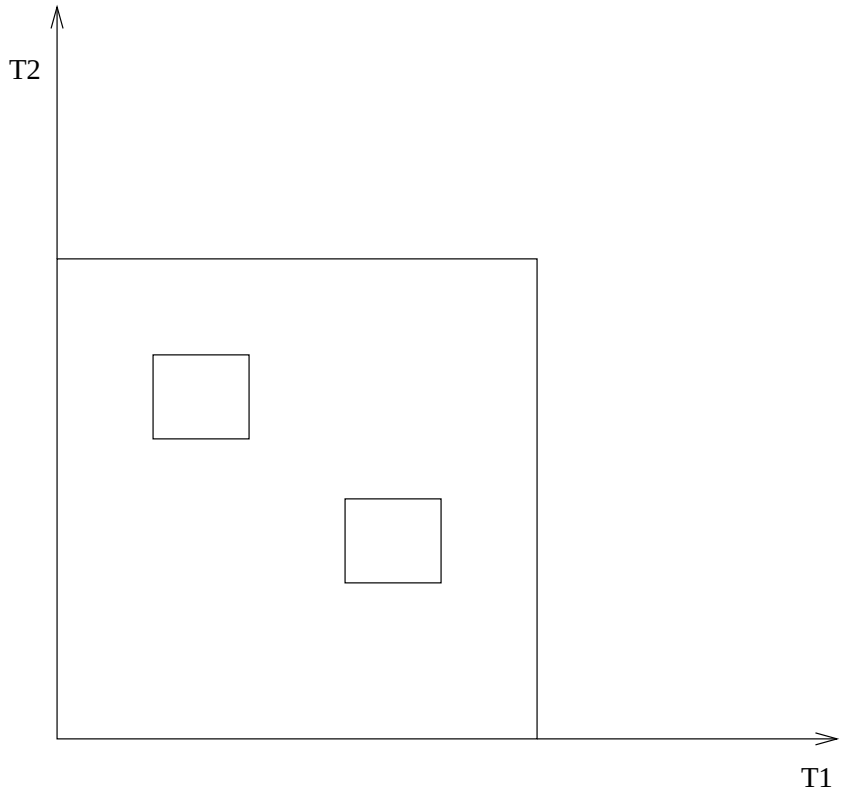


As for question (i), the answer is geometrically clear (see [CRJ87]). The only way a path coming from $(0,\ldots,0)$ may be stopped before reaching $(1,\ldots,1)$ is by "meeting" a corner like the dashed one ($PaPb$, $PbPa$) in Figure 7.3. As soon that a path goes into the small dashed rectangle, it cannot reach $(1,\ldots,1)$. Formally, this question relates to a connectedness result. We will look at that in Section 7.2.

Question (ii) is less immediate since we first have to define what the correctness condition is. In the airline reservation system example we have only demanded that no seat be sold twice. This means that some parts of the transactions may be done in parallel, but that the execution must be sufficiently constrained so that the resulting reservations are the same as some sequential treatment of the queries of the customers. In database theory this correctness criterion is known as "serializability". It has a basic "geometric" formalization in [Ull82] in the form of a topological condition on the "graph of transactions". We show now, following J. Gunawardena [Gun94], that it is even more directly of a geometric nature, and that the serialization property can be read on the process graph.

This may seem strange since the correctness criterion seems essentially given as a condition on states of the system. Do not forget though this assumption on the representation of transactions as not depending on the actual values of items. Surely, some arithmetical operation involved in the booking process may commute with other operations for some values of the items, but we have to think that to be true for all values is odd (strange programming at least). The condition now is then only on paths of executions. If you look at the forbidden square, or mutual exclusion in 7.2 reproduced in 7.4, the values of the items at its top right depend on the way we have reached this point. Going on the left of the hole may give different results from going on the right of the hole: just think at the following example. The initial value of a is 0. The arithmetic operations involved for the processes when the lock on a has been acquired is $a := a+1$ for T_1 and $a := 2{*}a$ for T_2. Going on the left means doing $a := 2{*}a$ before $a := a+1$, result is $a = 1$. Going on the right means doing $a := a + 1$ before $a := 2 * a$, result is $a = 2$. Instead of looking at the holes, look at the filled parts of the drawing 7.2. It becomes obvious now that all paths below (or at the right hand side of) the hole are serializable to the right boundary of the square, and that all paths above (or at the left hand side of) the hole are serializable to the left

Figure 7.5: A process graph with two mutual exclusions.



boundary of the square. Holes appear to be the elements to discover. They are the obstructions to the "continuous" deformation of paths (homotopy), which is the "infinitesimal" serialization equivalence. Here, the system is serializable since any path can be deformed onto one of the interleavings $T_1;T_2$ or $T_2;T_1$, i.e. any path gives the same result as a serial execution of the transactions.

Let us examine another process graph we may have (see Figure 7.5). Here, the paths "in between" the two holes cannot be deformed onto one of the interleavings, hence they are not serializable.

The aim of protocols for concurrent databases is to provide us with a uniform[4] way of insuring the consistency of the database. A good example is the two-phase protocol. Every transactions must acquire all locks of all items they will compute on (first phase), compute, and at the end they must release all their locks. For instance $T_1 = PaPbPcVaVbVc$ verifies the two-phase protocol (is "two-phase locked") whereas $T_2 = PaVaPbVb$ does not. It can be proven by combinatorial means that it makes all systems of transactions serializable (or in short, it is serializable). However it does not prevent deadlocks. Geometrically, the proof that it is serializable has been given in [Gun94], and is much more illuminating than the combinatorial one of, say, [Ull82]. Basically, it is proven that the n-cubes forbidden by the two-phase protocol form a unique hole in the "centre" of $[0,1]^n$. It is then easy, using a "radial" homotopy to deform all paths

[4] I.e. independently of what process we want to program.

of execution onto one of the interleavings, and then prove the serializability.

This proof is not completely satisfactory though. First, we use continuous methods. They are elegant but induce a few complications, like knowing that the paths correspond to real ones. Secondly, we use a standard theory of homotopy which authorizes reversal of time. Here, we really need a homotopy theory for "oriented" paths in which the allowed deformations are only transverse to the flow of time.

In the following we develop such a theory, in a discrete framework using Higher-Dimensional Automata. In Part V, we will look at the continuous couterpart. The theory will generalize also to higher-dimensional mutual exclusion problems.

The simple theory we are going to develop applies for semi-regular HDA only. Then, we will introduce a more complex theory, generalization of this to "combinatorial" HDA (Chapter 10) and to general HDA.

Protocols for distributed systems

Here, we want to deal with general problems that one can have in programming distributed systems. The case of concurrent databases can be considered as a first example. More generally, we are concerned with the following type of problems,

- given a number of hypotheses on the distributed system, like a topology of the communication network, a specification of the way messages are sent and received (asynchronously, synchronously, with bounded buffers, with no loss etc.), or in case of a shared-memory system, a number of assumptions like sequential/concurrent read/write etc.

- given a specification of what we want to program (as a set of distributed processes) on that system in the form of conditions on the input values accepted by this set of processes and conditions on the output values that this set of processes should compute,

- given a number of requirements on the execution of this program, like being as most efficient as it can be, or (it may be seen as a limit case of the previous requirement) being robust enough to compute a good part of the output specification even if some processors fail,

- the questions are: "Does such an algorithm exist on such machines?" and if the answer is positive, "Can we derive it from the specification of the problem ?".

All this is formalized under the name of *decision tasks*. Let us first give a few examples, following the presentation of [HS93] and [Her94].

The *consensus task* (abstraction of the commitment problem in concurrent database theory where transactions have to agree on a common value or abort) is a decision task in which N asynchronous processes begin with arbitrary input

values in some set S and must agree at the end on some common value taken from S.

The *renaming task* is another decision task in which N asynchronous processes begin with disjoint values in a set of "names" S and must end with new names (i.e. disjoint values) taken from a much smaller subset S' of S.

Finally the *k-set agreement task* asks for arbitrary input values (in some set S) but no more than k output values (in S as well). This can be seen as a partial consensus among the processes.

Now, an algorithm may be constrained in the following way. Call an execution of a program on a distributed system t-faulty if at most t processes in the program fail. Then an algorithm is t-resilient if it solves a decision task in every t-faulty execution. An algorithm is wait-free if it is $(n-1)$-resilient, where n is the number of processes.

It is proved for instance in [HS93] that in a shared-memory model with single reader/sin–gle writer registers providing atomic read and write operations, k-set agreement requires at least $\lfloor f/k \rfloor + 1$ rounds where f is the number of processes that can fail. This is done in a very nice geometric framework, and general tests are given for solving t-resilient problems. Not only impossibility results can be given but also constructive means for finding algorithms derive from this work (see for instance [HS94]).

We will see how it relates to the HDA approach in Section 8.2.3 and in Section 10.5.

Scheduling problems on modern architectures

Modern machine and processor architecture combine many elements that, if well used, greatly enhance the performance of the system, but if not, slow down the computation a lot. Vectorial units or pipelines (see Example 7.1) are an example. The reason is that some of these elements have good performance (like pipelines) if and only if we can assume a very precise ordering on instructions executed at run-time, whereas others can run almost arbitrary sequences of operations. Knowing this, we may reorder the actions to be executed on the latter elements so that we can use the former elements at the best of their capabilities.

This is the view taken for instance in [AF92] where some elements have strong consistency requirements (serializability) whereas some others have weak consistency requirements only. As we can see, this elaborates on the case of concurrent databases in the sense that we need a real classification of all possible orderings of actions (i.e. of all schedulers) and not only a proof that all schedulers are "equivalent" to one of the interleavings of the transactions.

Our approach

Let us discuss now what we should consider to be a scheduler of a program or a set of processes in the HDA framework.

Figure 7.6: A HDA (i) and its set of paths (ii).

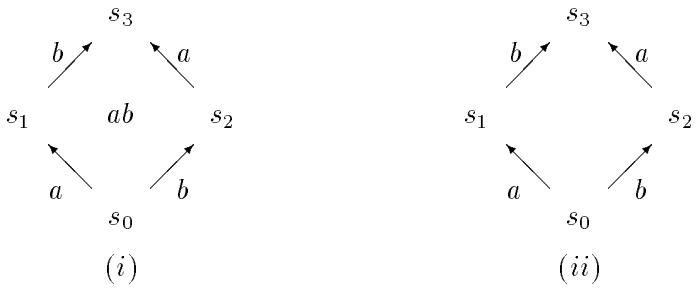
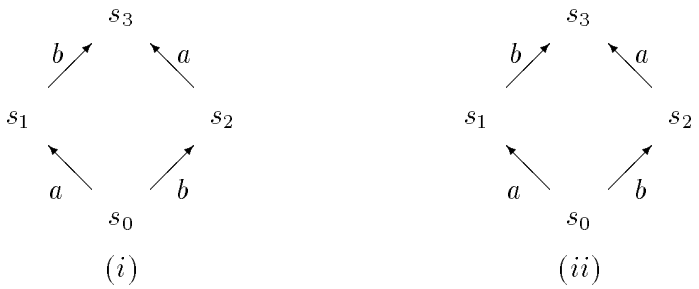


Figure 7.7: Another HDA (i) and its set of paths (ii)



Suppose we have a real machine with only a finite number n of processors on which we want to implement a semantics given by HDA. What should we consider as a **valid implementation**?

We first look at an instructive example for $n = 1$. Suppose the semantics of a program P is given by the truly concurrent execution of a and b pictured as the 2-transition in (i) of Figure 7.6. Then the valid execution paths are given by (ii) of the same figure. A scheduler can choose statically to do a then b or b then a. a then b is one scheduler and b then a is another. They are essentially the same (this will be defined formally as an equivalence relation between schedules) since a and b are non-interfering. In a geometrical manner, they are equivalent since one can continuously deform one path onto the other through the 2-transition ab (homotopy). In more well-known terms (Mazurkiewitz trace theory) one can understand ab as a commutation relation between a and b that is, ab is serializable to a then b and serializable to b then a [Ull82].

If P were the mutual exclusion between a and b ((i) of Figure 7.7) then do we have also two equivalent schedulers on a one-processor machine? The answer is no: choosing *a-priori* to fire a before b is radically different from choosing *a-priori* to fire b before a. Suppose for instance that a is the action on a process 1 of accessing a shared resource R and b is the action on a process 2 of accessing R as well. Then we should think of the two processes to be in competition for R and the scheduler does not have to make one wait for the other to access it first if the other was ready to: it is a matter of inefficiency and it transforms the

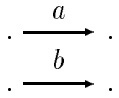
properties of the program (livelocks etc.). Moreover, if we are at an abstract level of the semantics, (where we have folded together some of the states for instance) we cannot be sure that the results of the two paths will be the same (look again at example $(a := a + 1) \mid (a := 2 * a)$). We knew that when we had the 2-transition in Figure 7.6 because it indicated a non-interfering behaviour, but here we just do not know. This is the abstract point of view implicitly used for studying protocols and concurrent databases (because they should not depend on the particular values of the items). There must then be one and only one scheduler whose trace is represented as (ii) in Figure 7.7. s_1 is an internal choice the parameters of which the scheduler cannot influence. There, the "hole" between ab and ba prevents us from deforming one onto the other. We now formalize this in more abstract geometrical terms.

7.2 The group of connected components

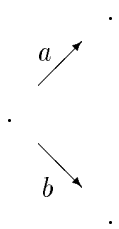
First of all, what is the amount of external non-determinism in a regular HDA ? This is the very first question we have to solve because we can only speak about serializability of computations once a branch of (external) choice has been chosen.

The geometric representations of non-determism are of two kinds.

The first one is an external choice between a and b:



The second one is an internal choice between the same actions:



We see that the degree of external non-determinism is related to the number of connected components of the HDA. We therefore come to defining this notion in a formal way.

Let s and s' be two states of M, i.e. $s, s' \in M_0$. We say that s and s' are connected and that s comes before s' ($s \leq s'$) if and only if there exists a 1-path p of finite length such that $\partial_0(p_1) = s$ and $\partial_1(p_k) = s'$.

Recall that a path p of dimension one (or 1-path), of length k and from p, q in an automaton M is a sequence $p = (p_i)_{1 \leq i \leq k}$ of elements of M such that

- $p_i \in M_{p+i-1,q-i+1}$,
- $d^1_0(p_i) = d^0_0(p_{i+1})$.

Lemma 26 *If two states s and s' are such that $s \leq s'$ then*

$$[s] = [s'] \text{ in } H_0(Tot(M))$$

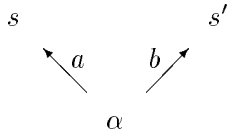
The reciprocal is false in general.

PROOF. Notice that $H_0(Tot(M)) = Tot(M)_0/Im(\partial_1 - \partial_0)$ so we only need to prove: s and s' are in the same connected component implies $\exists z \in Tot(M)_1$ with $s - s' = (\partial_1 - \partial_0)(z)$,

We have $p = (p_i)_{i=1,\ldots,k}$ such that $\partial_0(p_1) = s$ and $\partial_1(p_k) = s'$. Therefore if we set $z = p_1 + p_2 + \ldots + p_k$ then,

$$\begin{aligned}(\partial_1 - \partial_0)(z) &= -s + d^1_0(p_1) - d^0_0(p_2) + d^1_0(p_2) - \ldots + d^1_0(p_{k-1}) - d^0_0(p_k) + s' \\ &= -s + s'\end{aligned}$$

A counterexample for the reciprocal is given by the branching,



$[s] = [s']$ in $H_0(Tot(M))$ but there is certainly no path from s to s'. □

The equivalence relation induced by the preorder $\leq$ on states is exactly the relation "being in the same connected component". It is entirely classified by $H_0(Tot(M))$. We call $\tilde{\Pi}_0(M) = H_0(Tot(M))$ the reduced group (or module) of connected components of M.

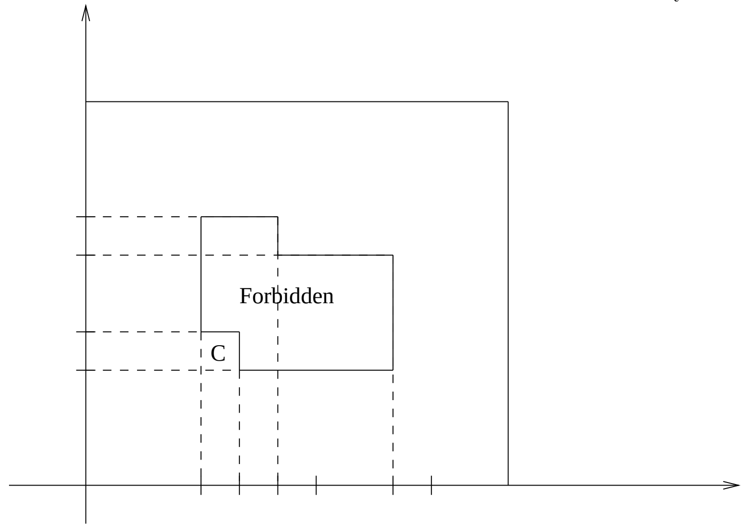
In fact, this is not quite enough for our purpose. What we really need to formalize is the fact that a point can be reached from another point by an *increasing path*, that is by a path in the automaton. In Figure 7.8, no path from the points in region C can reach the final state (the upper right corner of the square), so to some extent, this shape should not be considered connected in the "oriented" setting. We can see that the new notion of connectivity is going to classify the deadlocks of the system. Nevertheless, this new connectivity is obviously not an equivalence relation, so there is no way to define a set of "oriented" connected components that would partition an automaton.

Instead of that we will say that a HDA X is connected in the oriented sense with respect to the pair of states (α, β) if and only if all states of X are reachable from α by a path of X and β is reachable from any state of X by a path of X.

7.3 Towards formal definitions

Let M be a regular automaton. **All automata will be acyclic HDA** unless stated otherwise.

Figure 7.8: A "non-connected" automaton in the oriented theory



Let $p = (p_i)_{i=1,\ldots,k}$ be a 1-path. It can be pictured as, for paths of dimension one,

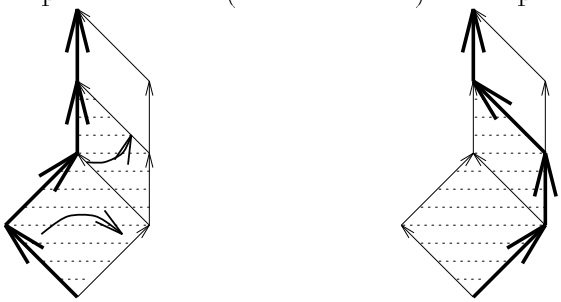
$$
\begin{array}{ccccccc}
 & & & & p_1 \in M_{1,0} & \xrightarrow{d_0^0} & \alpha \in I \\
 & & & & & & \downarrow d_0^1 \\
 & & p_1 \in M_{2,-1} & & \xrightarrow{d_0^0} & & = \\
 & & \vdots & & & & \\
p_k \in M_{k,-k+1} & \xrightarrow{d_0^0} & = & & & & \\
\downarrow d_0^1 & & & & & & \\
\epsilon & & & & & &
\end{array}
$$

where α and ϵ are respectively its initial and final states.

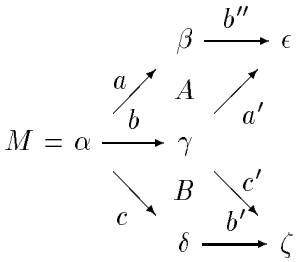
We wish to define **geometrically** how two paths of dimension one are to be considered equivalent in a scheduler. Let us look at the HDA from a different point of view. We slice paths into actions that occur at a given time: supposing that all paths we are interested in begin in $M_{0,0}$, we say that we are at time i when we look at actions in $M_{i,-i+1}$.

Let p and q be two paths. We say that p and q are **elementary equivalent** at time i if and only if p_i and q_i are two ends of a 2-transition A and $p_j = q_j$ for $j < i-1$ and $j > i$ and if p_{i-1} and q_{i-1} are two beginnings of the same 2-transition A. This corresponds to the idea of continuously deforming one path onto the other (or to use a commutation rule between two transitions) like one can see in Figure 7.9. We define equivalence to be the reflexive transitive closure of elementary equivalence.

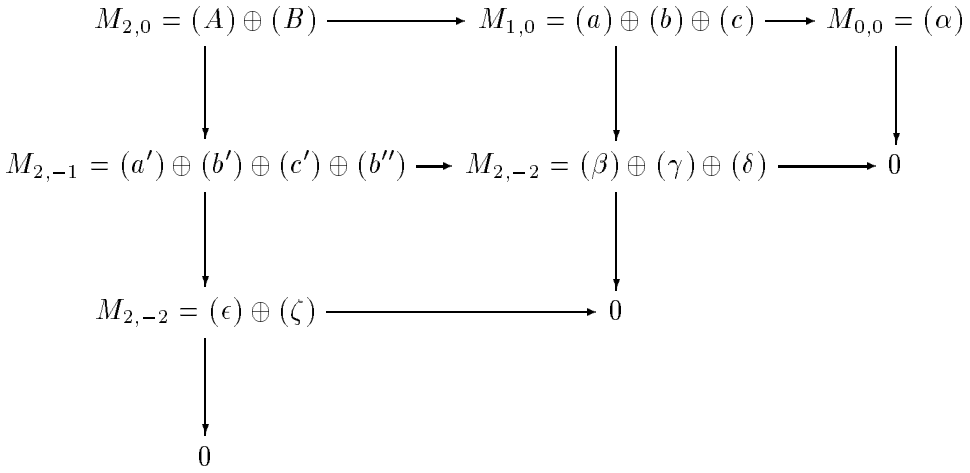
Figure 7.9: Step by step deformation (curved arrows) of one path onto an other



Let us picture an example. Consider the automaton



and suppose that α is in $M_{0,0}$. Thus, M can be algebraically defined as,



For instance, the 1-path (a, b'') is elementary equivalent to the 1-path (b, a') . Similarly (b, c') and (c, b') are elementary equivalent but not equivalent to any of the two former 1-paths.

Now, we are looking for an **algebraic definition** of this equivalence.

7.4 The fundamental group

The fundamental group of length k

Let $P_1^k(M)$ be the set of all 1-paths of length k from $I \subseteq M_{0,0}$ in the semi-regular HDA M (called the "elementary" 1-paths). It generates a sub-R-module of the

product module $\underline{M}_{1,0} \times \underline{M}_{2,-1} \times \ldots \times \underline{M}_{k,-k+1}$: the addition and external multiplication are defined on each component of the paths. By abuse of notation, we write $P_1^k(M)$ for this R-module.

Let $p = (p_i)_{1 \leq i \leq k}$ and $q = (q_i)_{1 \leq i \leq k}$ be two elements of $P_1^k(M)$. Then we say that p and q are equivalent or homotopic ($p \sim q$) if and only if $p - q$ is in $Im\,(\partial_0 - \partial_1)$ (see [ML63]) in $\underline{M}$. The first thing to prove is that it corresponds to our geometric definition of the last section.

Suppose that we have p and q elementary equivalent. Then we have $p - q = p_{i-1} - q_{i-1} + p_i - q_i$ with $p_i = d_0^1(A)$, $q = d_0^1(A)$, $p_{i-1} = d_1^0(A)$ and $p_{i-1} = d_0^0(A)$ (for instance) and A is a 2-transition. Therefore, $p - q = \partial_1(A) - \partial_0(A)$.

Now, suppose that M is acyclic and that we have two paths $p = (p_i)_{i=1,\ldots,k}$ and $q = (q_i)_{i=1,\ldots,k}$ such that $p - q = (\partial_0 - \partial_1)(X)$, $X \in \underline{M}$. M is acyclic, so we can decompose $X = \sum_{i=1,\ldots,k-1} X_i$ with $\partial_0(X_i) - \partial_1(X_{i-1}) = p_i - q_i$. Each X_i can be decomposed onto the basis M of $\underline{M}$, i.e. $X_i = \sum_j X_{i,j}$, $X_{i,j} \in M$. We suppose that p_i and q_i are elements of M (i.e. p and q are "natural" generators of $P_1^k(M)$). We can suppose that the $X_{1,j}$ (up to reordering and discarding of redundant ones) are such that (the a_j^i belong to M_1),

$$\partial_0(X_{1,1}) = p_1 - a_1^1$$

$$\partial_0(X_{1,2}) = a_1^1 - a_2^1$$

$$\ldots$$

$$\partial_0(X_{1,l_1}) = a_{l_1-1}^1 - q_1$$

We have also,

$$\partial_1(X_{1,1}) = b_1^1 - c_1^1$$

$$\partial_1(X_{1,2}) = b_2^1 - c_2^1$$

$$\ldots$$

$$\partial_1(X_{1,l_1}) = b_{l_1}^1 - c_{l_1}^1$$

where we have ordered the b's and $c's$ so that $\partial_0(c_j^1) = \partial_1(a_{j-1}^1)$ (where by convention we set $a_0^1 = p_1^1$) and $\partial_0(b_j^1) = \partial_1(a_j^1)$ (where by convention, $a_{l_1}^1 = q_1$). This implies that the (a_j^1, b_j^1) and (a_{j-1}^1, c_j^1) are paths of length 2 in M.

Then as $\partial_0(X_2) - \partial_1(X_1) = p_2 - q_2$, we can order the $X_{2,j}$ so that $\partial_0(X_{2,j}) = u_j - v_j$, with $u_j = b_{f(j)}^1$ or $u_j = c_{f(j)}^1$ and $v_j = b_{g(j)}^1$ or $v_j = c_{g(j)}^1$, f, g increasing maps. This enables to complete the paths of length 2 previously defined and have paths of length 3.

These paths are defined in such a way that they are elementary equivalent from one to the next through one of the $X_{i,j}$. This generalizes to $k \geq 3$ by carefully decomposing X at time slices greater or equal than 3. This tedious proof is left to the reader.

We define the **fundamental group** of M for paths of length k to be $\Pi_1^k(M) = P_1^k(M)/\sim$.

Proposition 11 *Suppose* $M = \sum_{p,0\leq q\leq k} M_{p,q}$ *is connected in the oriented sense with respect to all pairs* $(\alpha,\beta) \in I \times M_{k,-k}$. *Let* O *be the image of* $I \times M_{k,-k}$ *by* u *such that* $u(x,y) = (x-y) \in \underline{I} \oplus \underline{M}_{k,-k}$. *Then,*

$$\Pi_1^k(M) = H_1\left((\underline{M}, O), \partial_0 - \partial_1\right)$$

where $H_1\left((\underline{M}, O), \partial_0 - \partial_1\right)$ *is the first relative homology group of the pair [ML63]* $N = (\underline{M}, O) = \underline{M}/O$ *and boundary operator* $\partial_0 - \partial_1$, *i.e. is the quotient module* $Ker(\partial_0 - \partial_1)_{|N_1}/(\partial_0 - \partial_1)(N_2)$.

PROOF. $Ker(\partial_0 - \partial_1)_{|N_1}$ is the R-module generated by the set of 1-paths of M starting from I and of length k since,

- if $p = (p_1, \ldots, p_k)$ is such a path, $(\partial_0 - \partial_1)(p_1 + \ldots + p_k) = \partial_0(p_1) - \partial_1(p_k)$ which is null in N,
- reciprocally, if $p \in Ker(\partial_0 - \partial_1)_{|N_1}$ then, as M is acyclic, $p = p_l + \ldots + p_m$ with $p_i \in \underline{M}_{i+1,-i}$, $m \geq i \geq l$.

  Suppose that $p \neq 0$ i.e. that $p_l \neq 0$ and $p_m \neq 0$. We know that $(\partial_0 - \partial_1)(p) \in O$.

  This implies first that $\partial_0(p) \cap \underline{M}_{0,0} = \lambda_1 i_1 + \ldots + \lambda_r i_r$ ($\lambda_j \in R$, $i_j \in I$ and $\partial_1(p) \cap \underline{M}_{k,-k} = -\lambda_1 m_1 - \ldots - \lambda_r m_r$ ($m_r \in M_{k,-k}$). This entails that $l = 0$, $m = k-1$.

  Finally it implies that $\partial_0(p_{i+1}) = \partial_0(p_i)$ for all i with $1 \leq i \leq k-2$. Decomposing the p_i onto the basis $M_{i+1,-i}$ of $\underline{M}_{i+1,-i}$, we find elements p_i^j of $M_{i+1,-i}$ and μ_i^j of R such that

$$p_i = \sum_j \mu_i^j p_i^j$$

$$\partial_0(p_{i+1}^j) = \partial_1(p_i^j)$$

  These form the decomposition of p onto the set of paths of M from I to $M_{k,-k}$.

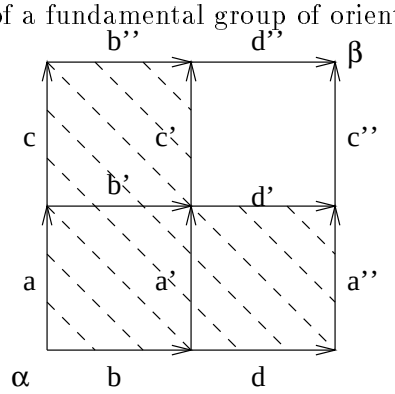
Quotienting by $(\partial_0 - \partial_1)(N_2)$ amounts to taking them modulo homotopy as we have seen already. □

The functor Π_1^k

Let f be a morphism from the semi-regular automaton M to the semi-regular automaton N. Then f induces a morphism $\tilde{f^k}$ from the pair $(\underline{M}, O)$ to the pair $(\underline{N}, O')$ for all k. Then $\tilde{f^k}$ induces $H_1(\tilde{f^k}) : H_1(\underline{M}, O) \to H_1(\underline{N}, O')$. This defines $\Pi_1^k(f) = H_1(\tilde{f^k})$ and makes Π_1^k into a covariant functor from the category of semi-regular automata to the category $R - Mod$ of R-modules.

All these definitions can be made starting from anywhere, not only $M_{0,0}$. For instance, if we consider initial states in $M_{p,q}$, we define in a similar manner the R-modules $P_1^{p,q,k}(M)$ and $\Pi_1^{p,q,k}(M)$, and the corresponding functors.

Figure 7.10: Example of a fundamental group of oriented paths.



We can also define submodules of these like $\Pi_1^{\alpha,\beta}(M)$ of paths from a state $\alpha \in M_{p,q}$ to a state $\beta \in M_{p+k-1,q-k+1}$. Proposition 11 has then the following counterpart,

Proposition 12 *Suppose* $M = \sum_{p,0\leq q\leq k} M_{p,q}$ *is an acyclic HDA with* $\alpha \in M_{0,0}$ *and* $\beta \in M_{k,-k}$*, connected in the oriented sense with respect to* (α, β)*. Then,*

$$\Pi_1^{\alpha,\beta}(M) = H_1\left((\underline{M}, (\alpha - \beta)), \partial_0 - \partial_1\right)$$

PROOF. Same as for the proof of Proposition 11 where we replace M_{k-k} by (β) and $I = (\alpha)$. □

If M was not connected in the oriented sense with respect to (α, β) then the isomorphism would not hold as one can see on Figure 7.8. The boundary of the forbidden region is a cycle for $\partial_0 - \partial_1$ but it is not generated by the set of paths from α to β. This is a condition that one would normally expect in ordinary homotopy theory (in Hurewicz theorem [May67]).

Example 32 *Let* M *be the acyclic semi-regular HDA whose geometric realization is shown in Figure 7.10. Then one can check that*

$$Ker\ (\partial_0 - \partial_1)_{|(\underline{M},(\alpha-\beta))} = (a + c + b'' + d'') \oplus (c + b' + c'' + b'')$$

$$\oplus(a + b + a' + b') \oplus (a' + d + a'' + d') \oplus (c' + d' + c'' + d'')$$

Of these generators, only $a+c+b''+d''$ *and* $c'+d'+c''+d''$ *are not in* $Im\ (\partial_0-\partial_1)$*. Therefore,*

$$\Pi^{\alpha,\beta}(M) = (a + c + b'' + d'') \oplus (c' + d' + c'' + d'')$$

There are two classes of equivalence of 1-paths modulo homotopy as expected (one can check that there are representants of them which are elementary paths, like $a + c + b'' + d''$ *and* $b + d + a'' + c''$*).*

Remark: There is no such technicality in the usual definitions of homotopy groups since we generally consider loops. Loops are composed in the obvious way, the same for loops of loops etc. This complexifies the definitions of "oriented" homotopy modules, and higher-order ones as well as we will see in the following.

7.4.1 Functors $\Pi_1^{p,q}$, Π_1^∞ and Π_1

(†) Functors $\Pi_1^{p,q}$ and Π_1^∞

In this section, we would like to identify paths of length k as the beginnings of some paths of greater length. This will enable us to define a fundamental group of infinite paths.

Let $t_k : P_1^{p,q,k+1}(M) \to P_1^{p,q,k}(M)$ be the module homomorphism defined by

$$t_k(x_1, \ldots, x_{k+1}) = (x_1, \ldots, x_k)$$

Then $x \sim y$ in $P_1^{p,q,k+1}(M)$ implies $t_k(x) \sim t_k(y)$ in $P_1^{p,q,k}(M)$. Therefore t_k induces (by an abuse of notation) $t_k : \Pi_1^{p,q,k+1}(M) \to \Pi_1^{p,q,k}(M)$. Consider the diagram

$$\Delta = \Pi_1^{p,q,0} \overset{t_0}{\leftarrow} \Pi_1^{p,q,1} \overset{t_1}{\leftarrow} \ldots \overset{t_k}{\leftarrow} \Pi_1^{p,q,k+1} \leftarrow \ldots$$

in the category of R-modules. Inverse limits exist in this category and we define $\Pi_1^{p,q} = \underset{\leftarrow}{lim}\, \Delta$.

Let

$$s_k^{p,q} : P_1^{p-1,q+1,k+1}(M) \to P_1^{p,q,k}(M)$$

be the module homomorphism defined by $s_k^{p,q}(x_1, \ldots, x_{k+1}) = (x_2, \ldots, x_{k+1})$. Then $s_k^{p,q}$ induces (by an abuse of notation again) $s_k^{p,q} : \Pi_1^{p-1,q+1,k+1}(M) \to \Pi_1^{p,q,k}(M)$. The diagram,

$$\begin{array}{ccc} \Pi_1^{p-1,q-1,k+1}(M) & \xrightarrow{s_k^{p,q}} & \Pi_1^{p,q,k}(M) \\ \uparrow t_{k+1} & & \uparrow t_k \\ \Pi_1^{p-1,q+1,k+2}(M) & \xrightarrow{s_{k+1}^{p,q}} & \Pi_1^{p,q,k+1}(M) \end{array}$$

commutes for all p, q and k. Therefore $s_k^{p,q}$ induces

$$s^{p,q} : \underset{\leftarrow}{lim}\, \Pi_1^{p-1,q+1,k}(M) \to \underset{\leftarrow}{lim}\, \Pi_1^{p,q,k}(M)$$

i.e.

$$s^{p,q} : \Pi_1^{p-1,q+1}(M) \to \Pi_1^{p,q}(M)$$

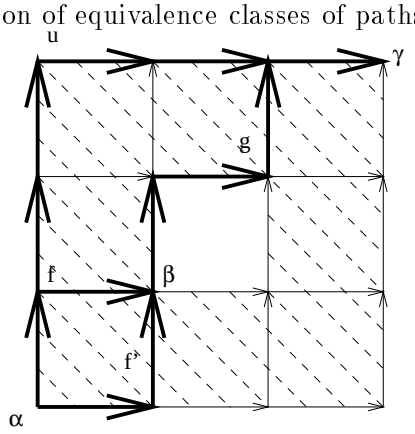
Let

$$\Lambda = \ldots \Pi_1^{p,q}(M) \overset{s^{p,q}}{\leftarrow} \Pi_1^{p-1,q+1}(M) \ldots$$

We define $\Pi_1^\infty(M) = \underset{\leftarrow}{lim}\, \Lambda$.

There is no obvious characterisation of Π_1^∞ in terms of homology. It is a well-known fact that homology does not preserve inverse limits in general (see [Mas78] for instance).

Figure 7.11: Composition of equivalence classes of paths modulo homotopy.



The functor Π_1

The real interesting homotopical object is the module of all *finite* paths modulo homotopy. The formal definition is as follows,

Definition 44 *The full fundamental group is*

$$\Pi_1(X) = \bigoplus_{\alpha,\beta \in X_0} \Pi_1^{\alpha,\beta}(X)/\{[f]_{\alpha,\beta} + [g]_{\beta,\gamma} = [f+g]_{\alpha,\gamma},\ f,\ g \text{ elementary paths}\}$$

The quotient condition means that a sum of two classes of paths that may compose is equated to the class of the sum of the two paths. In this homotopy group, we cannot reverse time, but we can consider collections of "oriented" paths.

The definition is valid since it is easy to see that if f and f' are two homotopic paths from α to β (respectively g and g' are two homotopic paths from β to γ) then $f+g$ and $f'+g'$ are two homotopic paths from α to γ.

Example 33 *Let M be the following semi-regular HDA whose geometric realization is shown in Figure 7.11. In this figure, we have pictured also four paths. One can verify that f and f' are homotopic elementary 1-paths, and that $[f]+[g] = [f+g] = [f'+g] = [u]$.*

Cycles

A 1-cycle of length k is a path $c \in P_1^k$ such that $\partial_0(c_1) = \partial_1(c_k) = 0$. They form a submodule of P_1^k called C_1^k. The homotopy relation induces a submodule $\tilde{\Pi}_1^k(M) = C_1^k(M)/\sim \subseteq \Pi_1^k(M)$. Then similarly to the construction of $\Pi_1(M)$ we have a reduced homotopy module $\tilde{\Pi}_1(M) \subseteq \Pi_1(M)$.

Lemma 27 $\tilde{\Pi}_1(M) = H_1(\underline{M}, \partial_0 - \partial_1)$.

7.5 (†) Homotopy of maps

Homotopy for topological spaces is an equivalence relation on loops (paths with the same beginning and ending) used to classify them up to "continuous deformation".

Let X be a topological space. If f and g are two continuous functions from the segment $[0,1]$ to X with $f(0) = f(1) = g(0) = g(1)$, i.e. if f and g are two (parametrizations of) loops, an homotopy between f and g is a continuous function G,

$$G : [0,1] \times [0,1] \to X$$

such that

$$\forall x, G(0,x) = f$$

$$\forall y, G(1,y) = g$$

G describes the process of continuous deformation between loops f and g.

As one would expect, it has an algebraic (discrete) counterpart, known as chain homotopy.

Let (I, ∂) be the complex such that I_0 is generated by s and t and I_1 is generated by u with $\partial(u) = t - s$, $\partial(t) = \partial(s) = 0$. I is the analogue of a unit segment. X is now a complex, and f and g are morphisms from the complex I to the complex X. We could define a chain homotopy to be, mimicking what we had in the continuous case,

$$G : I \otimes I \to X$$

with,

$$\forall x, G(s \otimes x) = f(x)$$

$$\forall y, G(t \otimes y) = g(x)$$

but this is not the standard definition. We can show that the existence of such a G is equivalent to the existence of α, a map of degree 1 between I and X such that,

$$\partial \circ \alpha + \alpha \circ \partial = f - g$$

To give a hint of the proof, just think of G and α as being related by $G(u \otimes x) = \alpha(x)$.

This gives us the definition,

Definition 45 *(see [ML63]) Let X and Y be two complexes and $f, g : X \to Y$ be two morphisms of complexes. A chain homotopy between f and g is a map of degree one $\alpha : X \to Y$ such that,*

$$\partial \circ \alpha + \alpha \circ \partial = f - g$$

In the case of bicomplexes, there is a notion of chain homotopy .

Definition 46 *(see [CE56]) Let $i : P \longrightarrow Q$ and $j : P \longrightarrow Q$ be two morphisms. Then i and j are homotopic, written $i \equiv j$, if and only if: $\exists \alpha_1, \alpha_2$ maps of degree (1,0) and (0,1), such that*

$$\alpha_0\partial_0 + \alpha_1\partial_1 + \partial_0\alpha_0 + \partial_1\alpha_1 = i - j$$

$$\alpha_1\partial_0 + \partial_0\alpha_1 = 0$$

$$\alpha_0\partial_1 + \partial_1\alpha_0 = 0$$

Then, we have,

Lemma 28 *Let $f : P \longrightarrow Q$ and $g : P \longrightarrow Q$ be two morphisms of HDA. If f and g are bicomplex-homotopic, then $Tot(f)$ and $Tot(g)$ are chain homotopic with a homotopy α such that $\alpha(P_{p,q}) \subseteq Q_{p+1,q} + Q_{p,q+1}$. The reciprocal is true for acyclic automata.*

PROOF. We have maps of degree $(1,0)$ and $(0,1)$, α_0 and α_1 respectively. Consider the map of degree 1 $\alpha = \alpha_1 - \alpha_0$ between $Tot(P)$ and $Tot(Q)$. Then,

$$\begin{aligned}(\partial_1 - \partial_0)\circ\alpha + \alpha\circ(\partial_1 - \partial_0) &= (\partial_0\alpha_0 + \alpha_0\partial_0 + \partial_1\alpha_1 + \alpha_1\partial_1) - (\partial_1\alpha_0 + \alpha_0\partial_1)\\ &\quad -(\partial_0\alpha_1 + \alpha_1\partial_0)\\ &= f - g\end{aligned}$$

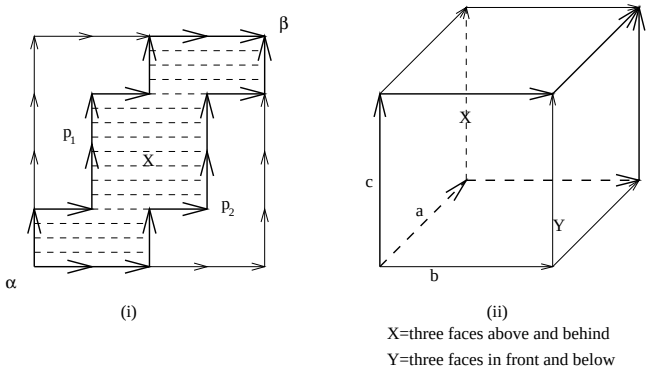
Therefore α is a chain map between f and g, and f and g are $(\partial_1 - \partial_0)$-chain homotopic.
We prove the reciprocal now. Suppose P and Q are bi-graded R-modules (that is they are the direct sum of $P_{m,n}$, $\partial_0 : P_{m,n} \longrightarrow P_{m-1,n}$ and $\partial_1 : P_{m,n} \longrightarrow P_{m,n-1}$) and suppose α is a map of degree 1 such that for all m,n, $\alpha : P_{m,n} \longrightarrow Q_{m,n+1} \oplus Q_{m+1,n}$. We can then decompose α into $\alpha_1 - \alpha_0$ where $\alpha_0 : P_{m,n} \longrightarrow Q_{m,n+1}$ and $\alpha_1 : P_{m,n} \longrightarrow Q_{m+1,n}$. We have $(\partial_1 - \partial_0)(\alpha_1 - \alpha_0) + (\alpha_1 - \alpha_0)(\partial_1 - \partial_0) = f - g$. Thus

$$\begin{aligned}(\partial_1 - \partial_0)\circ\alpha + \alpha\circ(\partial_1 - \partial_0) &= (\partial_0\alpha_0 + \alpha_0\partial_0 + \partial_1\alpha_1 + \alpha_1\partial_1) - (\partial_1\alpha_0 + \alpha_0\partial_1)\\ &\quad -(\partial_0\alpha_1 + \alpha_1\partial_0)\\ &= f - g\end{aligned}$$

Notice now that for $x \in P_{m,n}$, $\partial_1\alpha_0(x) \in Q_{m+1,n-1}$, $\partial_0\alpha_1(x) \in Q_{m-1,n+1}$, but $f - g \in Q_{m,n}$. Therefore, $\partial_1\alpha_0 + \alpha_0\partial_1 = 0$, $\partial_0\alpha_1 + \alpha_1\partial_0 = 0$, and $\partial_0\alpha_0 + \alpha_0\partial_0 + \partial_1\alpha_1 + \alpha_1\partial_1 = f - g$. □

The fundamental group has a characterization through homotopy of regular maps. This will be shown in Section 7.6.2.

Figure 7.12: A path X of dimension 2 between two paths p_1, p_2 of dimension 1.



7.6 Higher-order homotopy groups

We have at least two different ways to define higher-order homotopy groups.

The first one is a direct generalization of the definition for the fundamental group. We had two "limiting" $(n-1)$-cubes in between which we could deform any sequence of n-cubes. That was defined with $n = 1$. For $n = 2$ we have a homotopy group of dimension 2 parameterized with two 1-paths p_1 and p_2, having the same initial and final states (see Figure 7.12). Then in order to define a "full" homotopy group, we have to glue together all parameterized homotopy groups, and the combinatorics of this glueing operation is much more complex than in dimension one.

The second one is via a "suspension" like construction. This gives numerous properties, similar to those we have in "standard" homotopy theory of, say, singular simplexes.

7.6.1 First definition

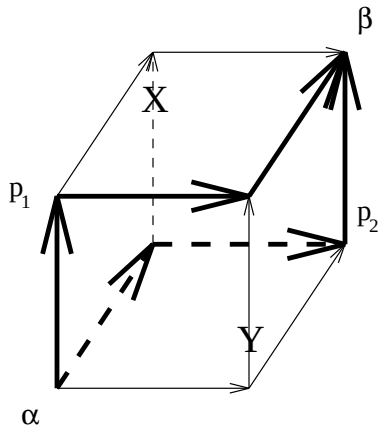
The definition we give now is "iterative" in the sense that we know what a 1-path between α and β is, or what is a 1-path of length k and that the definition of n-paths between $(n-1)$-paths depend on that definition.

Definition 47 *Let $n \geq 2$ and M be an acyclic HDA. Let p_1 and p_2 be two $(n-1)$-paths between two $(n-2)$-paths α and β. We suppose that $p_i = (p_i^1, \ldots, p_i^k)$ and that $p_i^1 \in M_{n-1+s,-s}$. The R-module of n-paths between p_1 and p_2 is the R-module of sequences $x = (x^1, \ldots, x^{k-1})$ such that there exists $\lambda \in R$ with,*

- $x^s \in M_{n+s,-s}$,
- $\partial_0(x^{i+1}) = \partial_1(x^i) + \lambda(p_1^{i+1} - p_2^{i+1})$,
- *module operations are pointwise addition and pointwise external multiplication.*

This R-module is named $P_n^{p_1,p_2}(M)$.

Figure 7.13: Two homotopic 2-paths.



X= three faces above and behind

Y=three faces in front and below

Let $Q_n^{(p_1,p_2;\alpha,\beta)}(M)$ be the set of normalized n-paths from p_1 to p_2 in M (p_1 and p_2 are normalized $(n-1)$-paths beginning at α and ending at β).

Supposing $p_i = (p_i^1, \ldots, p_i^k)$ and $p_i^1 \in M_{n-1+s,-s}$, its elements $x \in Q_n^{(p_1,p_2;\alpha,\beta)}(M)$ are $x = (x^1, \ldots, x^{k-1})$ such that

- $x^s \in M_{n+s,-s}$,
- $\partial_0(x^{i+1}) - \partial_1(x^i) = p_1^{i+1} - p_2^{i+1}$,

They form a basis of the R-module $P_n^{p_1,p_2}(M)$.

We say that two n-paths $p, q \in P_n^{p_1,p_2}(M)$ are homotopic, and we write $p \sim q$ if and only if $p - q \in Im\,(\partial_0 - \partial_1)$[5]. For instance, paths X and Y between p_1 and p_2 in the filled-in cube Figure 7.13 are homotopic.

We define $\Pi_n^{p_1,p_2}(M) = P_n^{p_1,p_2}(M)/\sim$.

As in Section 7.4.1 we can define n-cycles which are n-paths $c = (c_i)_{i=1,\ldots,k}$ such that $\partial_0(c_1) = \partial_1(c_k) = 0$. They form a submodule of $P_n^{p_1,p_2}(M)$ called $C_n^{p_1,p_2}(M)$. The homotopy relation defines a reduced homotopy module $\tilde{\Pi}_n(M)$. Then,

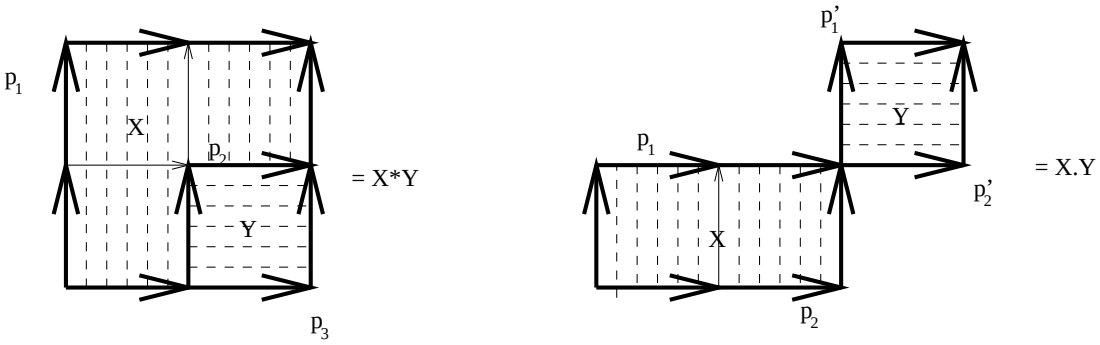
Claim 2 *Suppose M is connected with respect to the initial and final states of p_1 and p_2, and that all $Pi_k(M)$, $k \leq n-1$ are of dimension one. Then,*

$$\Pi_n^{p_1,p_2}(M) = H_n\left((\underline{M}, T), \partial_0 - \partial_1\right)$$

where T is the image of $p_1 \oplus p_2$ under the map u with $u(x,y) = x - y$.

[5]This actually means that $\forall i$, p_i and q_i are in the transitive closure of the union of the homotopy relation in the complex of modules $((M_{j,-i+1})_j, \partial_0)$ with the homotopy relation in $((M_{n-i+2,j})_j, \partial_1)$. This remark will enable us to generalize the homotopy of oriented paths we are defining on free general HDA generated by semi-regular HDA to the general case of combinatorial HDA, hence to all general HDA (see Chapter 10). The transitive closure of these homotopy relations seem somewhat related to the spectral sequence of the bicomplex, but this has not been formalized well enough yet.

Figure 7.14: $*$ and $\cdot$ operations on n-paths



This is an analogue of the general Hurewicz theorem.

Similarly to Section 7.4 we want to define $\Pi_n(M)$. This is much more difficult than it was in dimension one.

Let $p_i = (p_i^1, \ldots, p_i^k)$ $(i = 1, 2, 3)$ and $p'_j = (p'^1_j, \ldots, p'^{k'}_j)$ $(j = 1, 2)$ be normalized $(n-1)$-paths.

We define two "composition" operations on normalized n-paths (Figure 7.14),

$$* : Q_n^{(p_1,p_2;\alpha,\beta)} \times Q_n^{(p_2,p_3;\alpha,\beta)} \to Q_n^{(p_1,p_3;\alpha,\beta)}$$

$$\cdot : Q_n^{(p_1,p_2;\alpha,\beta)} \times Q_n^{(p'_1,p'_2;\beta,\gamma)} \to Q_n^{(p_1\cdot p'_1, p_2\cdot p'_2;\alpha,\gamma)}$$

by,

$$(x_1, \ldots, x_{k-1}) * (y_1, \ldots, y_{k-1}) = (x_1 + y_1, \ldots, x_{k-1} + y_{k-1})$$

$$(x_1, \ldots, x_{k-1}) \cdot (y_1, \ldots, y_{k'-1}) = (x_1, \ldots, x_{k-1}, y_1, \ldots, y_{k'-1})$$

given that for normalized 1-paths $x = (x_1, \ldots, x_{k-1})$ and $y = (y_1, \ldots, y_{k'-1})$ between α and β (respectively, β and γ),

$$x \cdot y = (x_1, \ldots, x_{k-1}, y_1, \ldots, y_{k'-1})$$

These are well defined operations.

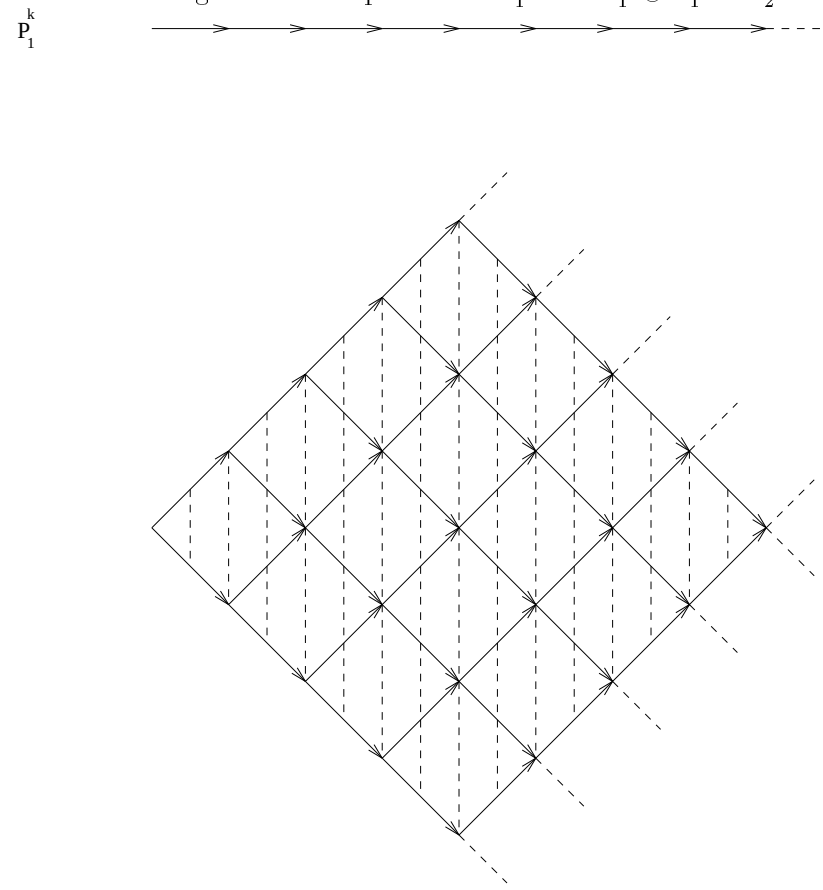
PROOF. We have $\partial_0(x_{i+1} + y_{i+1}) - \partial_1(x_i + y_i) = p_1^{i+1} - p_3^{i+1}$. This proves that $*$ is well defined.

Finally, $\partial_0(x_{i+1}) - \partial_1(x_i) = p_1^{i+1} - p_2^{i+1}$ $(i = 1, \ldots, k-2)$ and $\partial_0(y_{j+1}) - \partial_1(y_j) = p_1^{j+1} - p_2^{j+1}$ $(j = 1, \ldots, k'-2)$ implies that $\partial_0((x \cdot y)_{j+1}) - \partial_1((x \cdot y)_j) = (p_1 \cdot p'_1)^{j+1} - (p_2 \cdot p'_2)^{j+1}$ $(k = 1, \ldots, k + k' - 3)$. This proves that $\cdot$ is well-defined. □

Let $\cong$ be the least congruence relation (with respect to the R-module structure) on $\bigoplus_{p_1,p_2} \Pi_n^{(p_1,p_2)}(X)$ such that for all normalized $(n-2)$-paths α, β, and γ, for all normalized $(n-1)$-paths p_1, p_2 and p_3 between α and β, and p'_1, p'_2 between β and γ, and all normalized n-paths X between p_1 and p_2, Y between p_2 and p_3, Z between p'_1 and p'_2,

$$[X] + [Y] \cong [X * Y]$$

Figure 7.15: A picture of P_1^k and $P_1^k \otimes P_1^k = P_2^k$.



$$[X] + [Z] \cong [X.Z]$$

The full homotopy group of dimension n is now,

$$\Pi_n(X) = \bigoplus_{p_1,p_2} \Pi_n^{(p_1,p_2)}(X) / \cong$$

7.6.2 (†) Second definition

Let P_1^k be the regular HDA defined by (see Figure 7.15),

- $(P_1^k)_0 = \{\alpha_0, \ldots, \alpha_k\}$,
- $(P_1^k)_1 = \{u_1, \ldots, u_k\}$,
- $(P_1^k)_n = 0$ $(n \geq 2)$,
- $d_0^0(u_i) = \alpha_{i-1}$, $d_0^1(u_i) = \alpha_i$.

Definition 48 *We define the n-fpaths of length* $(k_1, \ldots, k_n)$ *in* M *to be*

$$Hom_0(P_1^{k_1} \otimes P_1^{k_2} \ldots \otimes P_1^{k_n}, \underline{M})$$

When we restrict this definition to paths of dimension one, it is easy to see the link between $P_1^{\alpha,\beta}(M)$ ($\alpha \in M_{0,0}$, $\beta \in M_{k,-k}$) and 1-fpaths of length k in M. There is in fact a bijection between the $x \in P_1^{\alpha,\beta}(M)$ and the $\tilde{x} \in Hom_0(P_1^k, \underline{M})$ with $\tilde{x}(\alpha_0) = \alpha$ and $\tilde{x}(\alpha_k) = \beta$. This bijection is given as follows,

- Given x, $\tilde{x}(u_i) = x \cap M_{i,-i+1}$,
- Given $\tilde{x}$, $x = \tilde{x}(\sum_{i=1,\ldots,k} u_i)$.

There is also a strong link between 2-fpaths of length $(1, k-1)$ and 2-paths between two 1-paths of length k, p_1 and p_2 (each of which beginning at α and finishing at β).

Definition and lemma 7 *Let A be a 2-path of length $k-1$ between p_1 and p_2, two 1-paths of length k from α to β. Therefore, we have (where $\lambda \in R$), $\partial_0(A_{i+1}) = \partial_1(A_i) + \lambda(p_1^{i+1} - p_2^{i+1})$, Then there exists a unique 2-fpath $f = fp(A)$ of length $(1, k-1)$ such that $(1 \leq i \leq k)$,*

- $f(\alpha_0 \otimes u_i) = \lambda p_1^i$ *$(1 \leq i \leq k-1)$*,
- $f(\alpha_1 \otimes u_i) = \lambda p_2^{i+1}$ *$(1 \leq i \leq k-1)$*,
- $f(u_1 \otimes \alpha_i) = \lambda p_1^{i+1} - \partial_0(A_{i+1})$ *$(0 \leq i \leq k-2)$*,
- $f(u_1 \otimes \alpha_{k-1}) = p_1^k$,
- $f(u_1 \otimes u_i) = A_i$.

PROOF. We have to verify that f defines a morphism of HDA from $P_1^1 \otimes P_1^{k-1}$ to $\underline{M}$ (see Figure 7.16).
We have,

$$\begin{aligned} \partial_0(f(u_1 \otimes u_i)) &= \partial_0(A_i) \\ &= \lambda(p_1^i - p_2^i) \end{aligned}$$

and,

$$\begin{aligned} f(\partial_0(u_1 \otimes u_i)) &= f(\alpha_0 \otimes u_i) - f(u_1 \otimes \alpha_{i-1}) \\ &= \lambda(p_1^i - p_1^i) + \partial_0(A_i) \end{aligned}$$

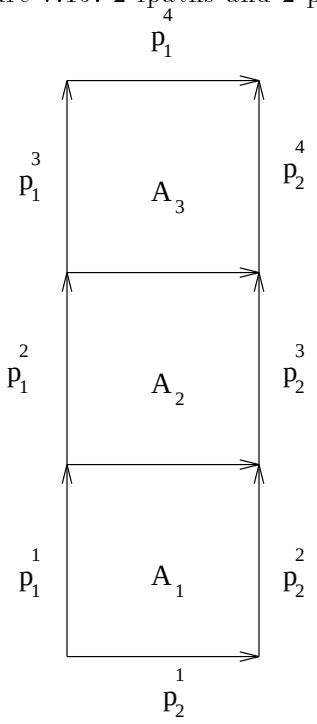
so the two terms are equal. Similarly,

$$\begin{aligned} \partial_1(f(u_1 \otimes u_i)) &= \partial_1(A_i) \\ &= f(\partial_1(u_1 \otimes u_i)) \end{aligned}$$

Then,

$$\begin{aligned} \partial_0(f(\alpha_0 \otimes u_i)) &= \lambda\partial_0(p_1^i) \\ &= f(\alpha_0 \otimes \alpha_{i-1}) \\ &= \lambda\partial_1(p_1^{i-1}) \end{aligned}$$

Figure 7.16: 2-fpaths and 2-paths compared.



and similarly for $f(\alpha_1 \otimes u_i)$ and p_2^i.
The last verification we have to make is,

$$\begin{aligned} \partial_0(f(\alpha_0 \otimes u_1)) &= f(\alpha_0 \otimes \alpha_0) \\ &= \lambda\alpha \\ &= \partial_0(f(u_1 \otimes \alpha_0) \end{aligned}$$

and similarly for $\partial_1(f(u_1 \otimes \alpha_k))$ and $\partial_1(f(\alpha_1 \otimes u_k))$. □

This definition generalizes easily to a function from $(n+1)$-paths of length $k-1$ between two n-paths of length k to $(n+1)$-fpaths of length $(1, \ldots, 1, k-1)$.
Let $\sim'$ be the homotopy of maps defined in Section 7.5. We write $\Pi'^k_n(X) = Hom_0(P_1^k \otimes P_1^1 \ldots P_1^1, X)/\sim'$. Notice that (by the adjunction $\otimes$, Hom)

$$\Pi'^k_n(X) = \Pi'^k_1(Hom(P_1^1, Hom(\ldots, Hom(P_1^1, X))))$$

We have even better: the higher-order homotopy groups can be computed through the suspension and first order homotopy functors.

Definition and lemma 8 *Let $S(X) = Hom(P_1^1, X)$ be the "suspension" of X. Then $\Pi'^k_n(X) = \Pi'^k_1(S^{k-1}(X))$.*

Proof. We just have to prove that,

(i) if we have two maps $f, g : P \otimes Q \to X$ such that $f \sim' g$, then their curried version $\hat{f}, \hat{g} : P \to Hom(Q, X)$ are such that $\hat{f} \sim' \hat{g}$ as well,

(ii) if we have $\hat{f}, \hat{g} : P \to Hom(Q, X)$ homotopic maps, then $f, g : P \otimes Q \to X$ their uncurried versions are homotopic maps as well.

We will only prove (i) since (ii) is very much similar. We are given a map $\alpha : (P \otimes Q)_{p,q} \to X_{p-1,q} \oplus X_{p,q-1}$ such that, $(\partial_1 - \partial_0)\alpha + \alpha(\partial_1 - \partial_0) = f - g$. Define $\hat{\alpha}(x) = (y \to \alpha(x \otimes y))$. $\hat{\alpha}$ is a well-defined map from $P_{p,q}$ to $(Hom(Q, X)_{p-1,q} \oplus Hom(Q, X)_{p,q-1})$. Now,

$$((\partial_1 - \partial_0)\hat{\alpha})(x)(y) + (\hat{\alpha}(\partial_1 - \partial_0))(x)(y)$$

$$= (\partial_1 - \partial_0)(\alpha(x \otimes y)) + \alpha((-1)^{dim\ x} x \otimes (\partial_1 - \partial_0)(y)) + \alpha((\partial_1 - \partial_0)(x) \otimes y)$$

$$(\partial_1 - \partial_0)(\alpha(x \otimes y)) + \alpha((\partial_1 - \partial_0)(x \otimes y))$$

□

We can now carry on the comparison between this definition of homotopy groups and the one of Section 7.6.

Lemma 29 *Let A, B be two $(n+1)$-paths in M of length $k-1$ between p_1 and p_2. Then, $A \sim B \Leftrightarrow fp(A) \sim' fp(B)$.*

PROOF. Suppose first that $A \sim B$. Then there is a $(n+2)$-transition C such that $A - B = (\partial_1 - \partial_0)(C)$. This means that C defines a $(n+2)$-path between A and B. $fp(C)$ can be identified with a map of degree one from $(P_1^1)^{\otimes^n} \otimes P_1^k$ to M which can actually be shown to be a homotopy between $fp(A)$ and $fp(B)$. Reciprocally, if we have a map α of degree one from $(P_1^1)^{\otimes^n} \otimes P_1^k$ to M which defines a homotopy from $fp(A)$ to $fp(B)$ then $\sum_{i=1,\ldots,k} C(u_1^{\otimes^n} \otimes u_i)$ is a $(n+2)$-transition defining a homotopy between A and B. □

Proposition 13 *If p_1 and p_2 are $(n-1)$-paths in X of length $k+1$ then $\Pi_n^{p_1,p_2}(X) \subseteq \Pi_n'^k(X)$ as R-modules.*

PROOF. The transformation fp from $P_n^{p_1,p_2}(X)$ to the R-module of n-fpaths between p_1 and p_2 preserves homotopy by Lemma 29. Moreover, fp is a R-module homomorphism. This entails the result. □

We do not know yet if the full homotopy groups in both definitions are isomorphic.

7.7 Some properties of the homotopy modules

7.7.1 Combinatorics of Π_1

There is first a pathological phenomenon to be described in HDA.

Definition 49 *Let M be a semi-regular HDA. A knot of dimension 1 in M is any 2-cube x such that $d_0^1 d_0^0(x) = d_0^1 d_1^0(x)$.*

Proposition 14 *Let M be a semi-regular HDA and let M' be the semi-regular HDA defined as*

$$M' = T_1(M)/\{d_0^0(x) = d_1^0(x), d_0^1(x) = d_1^1(x)/x \in M_2\}$$

Then if M is acyclic and has no knots, $\Pi_1(M) = P_1(M')$, i.e., M' is a "canonical" representant of the retracts of M.

Proof. Let $s : M \to M'$ be the canonical morphism associated with the quotient construction. We first prove that if $p, q \in P_1^{\alpha,\beta}(M)$ with $p \sim q$ then $s(p) = s(q)$. $p \sim q$ implies that $\exists A \in \underline{M_2}, p - q = (\partial_0 - \partial_1)(A)$. $\underline{M_2}$ is generated by 2-cubes in M therefore there exists $(x_i)_{i\in I}$, $x_i \in M_2$, I a finite index set, and $\alpha_i \in R$ such that $A = \sum_{i\in I} \alpha_i x_i$. Then, $p - q = \sum_{i\in I} \alpha_i (d_0^0 - d_1^0 - (d_0^1 - d_1^1))(x_i)$. As M is acyclic, we can choose the x_i in order to have $x_i \in M_{2+u_s,-u_s}$. Then, $p_i - q_i = (d_0^0 - d_1^0)(\sum_{u_j = i+s-1} \alpha_j x_j) + (d_0^1 - d_1^1)(\sum_{u_j = i+s-2} \alpha_j x_j)$ if we suppose that $\alpha \in M_{s,-s}$. As in M', $d_0^0 = d_1^0$ and $d_0^1 = d_1^1$ on 2-transitions, $s(p_i) = s(q_i)$, hence (by an abuse of notation), $s(p) = s(q)$. Notice that we have not used the hypothesis on knots.

Reciprocally, we must show that for all paths of M', p' and q', from, say α' to β', necessarily corresponding via s to paths p and q in M from α to β, $p' = q'$ implies $p \sim q$. We know that there exists $p_1, \ldots, p_k$ 1-paths of M, from α to β such that $p_1 = p$, $p_k = q$ and there exists $x_1, \ldots, x_{k-1} \in M_2$ such that for all u, some $(p_u)_{i_u} = d_l^k(x_u)$ and $(p_{u+1})_{i_u} = d_{l+1 mod 2}^k(x_u)$. If M is acyclic and has no knots then we show that necessarily, (restricting to the case where $k = 0$, the other case is symmetric) $(p_u)_{i_u+1} = d_{l+1 mod 2}^1(x_u)$ and $(p_{u+1})_{i_u+1} = d_l^1(x_u)$ (E). As a matter of fact, knowing that M has no knots implies that $d_0^1((p_u)_{i_u}) = d_0^0((p_u)_{i_u+1})$ is distinct from $d_0^1((p_{u+1})_{i_u}) = d_0^0((p_{u+1})_{i_u+1})$. This means that we have to use one of the equations defining M' as a quotient of M for going from $(p_u)_{i_u+1}$ to $(p_{u+1})_{i_u+1}$. By hypothesis, we only use one 2-transition and the equations associated with it per "move". Therefore, we can only have the equations (E). This entails that $p_u \sim p_{u+1}$ hence $p \sim q$.

Notice that whenever there is a knot in M, there is no representant of retracts of M as a subHDA of M, see Figure 7.17. □

In Figure 7.18, on the left hand side we have drawn "dependent" holes. Notice that there are four homotopy classes of paths from the bottom left corner to

Figure 7.17: A knot M (i) an "upper approximation" of the paths modulo homotopy and the corresponding M' (ii) that cannot be a retract of M.

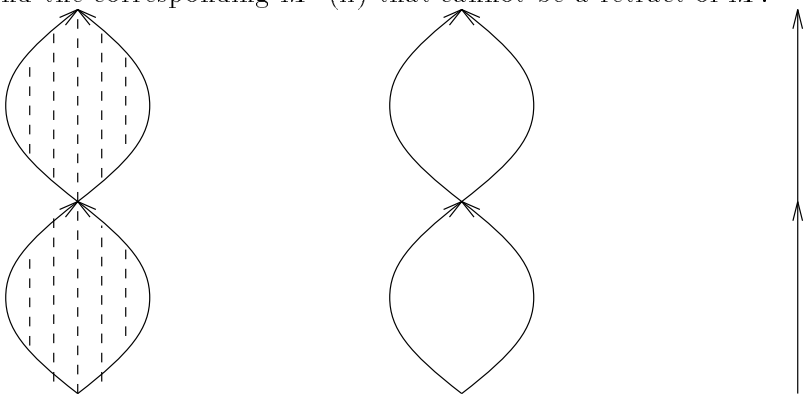
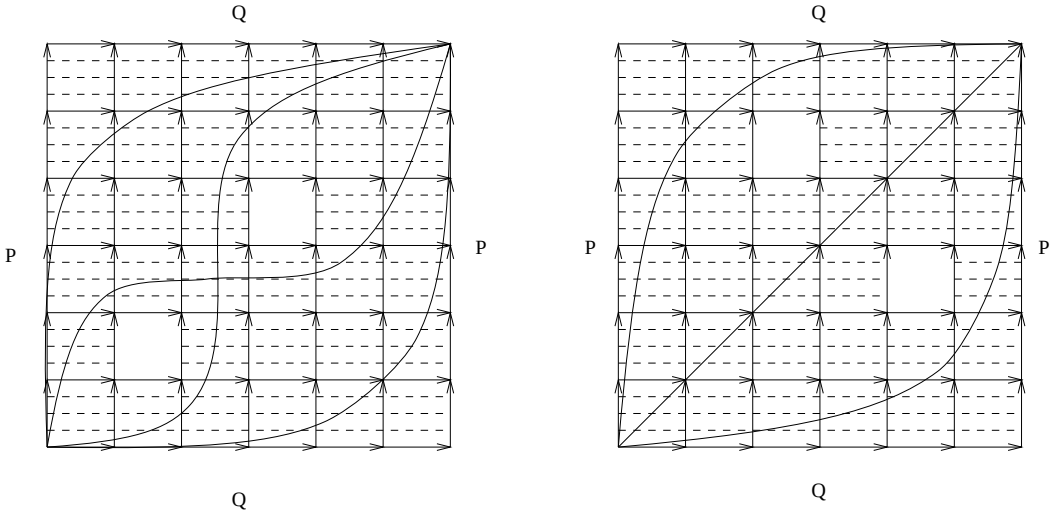


Figure 7.18: Two different configurations of holes: left is "dependent" holes, right is "independent" ones.



the top right one. On the right hand side, there is an example of "independent" paths. There are only three distinct homotopy classes of paths (still from bottom left to top right).

This shows that Π_1 is not characterized only by the number of holes as in ordinary homotopy theory but also by the way holes are dependent from each other. Apart from pathological phenomena (knots), Proposition 14 shows that semi-regular HDA M are homotopic to "posets of holes".

The exact relationship between number of cycles and number of homotopy classes of paths between two states α and β can be studied through the relative homology exact sequence [ML63].

It reads (where $S = (\alpha - \beta)$) and the homology groups in the sequence are taken with respect to the differential $d = \partial_1 - \partial_0$),

$$\ldots \longrightarrow H_{n+1}(S,0) \xrightarrow{i^*} H_{n+1}(Tot(M),0) \xrightarrow{j^*} H_{n+1}(Tot(M),S) \xrightarrow{d^*} H_n(S,0)\ldots$$

Here, $H_1(S,0) = 0$, $H_0(S,0) = S$, therefore,

$$0 \longrightarrow \mathrm{H_1(Tot(M),0)} \xrightarrow{j^*} \mathrm{H_1(Tot(M),S)} \xrightarrow{d^*} \mathrm{S} \xrightarrow{i^*} \mathrm{H_0(Tot(M),0)} \xrightarrow{j^*} \mathrm{H_0(Tot(M),S)} \longrightarrow 0$$

that is,

$$0 \longrightarrow \tilde{\Pi}_1(M) \longrightarrow \Pi_1(M) \longrightarrow S \longrightarrow \tilde{\Pi}_0(M) \longrightarrow \Pi_0(M) \longrightarrow 0$$

where $\Pi_0(M) = H_0(Tot(M),S)$. We suppose that α and β are connected, so $[\alpha] = [\beta] \in H_0(Tot(M))$ and $i_* = 0$, $Im\, d^* = S$. Therefore,

$$\Pi_1(M) = \tilde{\Pi}_1(M) \oplus d^{*^{-1}}(S)$$

$$\Pi_0(M) = \tilde{\Pi}_0(M)$$

Looking at the dimensions, we see that the number of generating paths modulo homotopy is always greater or equal than the number of generating cycles modulo homotopy.

7.7.2 Seifert/Van Kampen theorem

We prove the Seifert/Van Kampen theorem for semi-regular HDA, making this homotopy theory closer to the "standard" one.

Theorem 1 *Consider the following co-cartesian square defining $X_1 \cup X_2$ and $X_1 \cap X_2$ where X_1 and X_2 are two semi-regular HDA,*

$$\begin{array}{ccc} X_1 \cap X_2 & \xrightarrow{j_1} & X_1 \\ {\scriptstyle j_2}\downarrow & & \downarrow{\scriptstyle i_1} \\ X_2 & \xrightarrow{i_2} & X_1 \cup X_2 \end{array}$$

Then the following square is also co-cartesian,

$$\begin{array}{ccc} \Pi_1(X_1 \cap X_2) & \xrightarrow{\Pi_1(j_1)} & \Pi_1(X_1) \\ {\scriptstyle \Pi_1(j_2)}\downarrow & & \downarrow{\scriptstyle \Pi_1(i_1)} \\ \Pi_1(X_2) & \xrightarrow{\Pi_1(i_2)} & \Pi_1(X_1 \cup X_2) \end{array}$$

i.e. $\forall G$, $\forall f_1, f_2 : \Pi_1(X_i) \to G$ *homomorphisms of* R*-modules, there exists a unique homomorphism of* R*-modules* $g : \Pi_1(X_1 \cup X_2) \to G$ *such that* $g \circ \Pi_1(i_1) = f_1$ *and* $g \circ \Pi_1(i_2) = f_2$.

PROOF. Let $p \in P_1^{\alpha,\beta}(X)$ be a one dimensional path in X, such that $\alpha, \beta \in X_1 \cap X_2$. As $X = X_1 \cup X_2$, $\underline{X} \subseteq \underline{X_1} \oplus \underline{X_2}$ and p decomposes into $p_1 + p_2$, $p_1 \in P_1^{\alpha,\beta}(X_1)$ and $p_1 \in P_1^{\alpha,\beta}(X_2)$. Then $[p]_X = i_1^*([p_1]_{X_1}) + i_2^*([p_2]_{X_2})$. This entails that we have to set $f([p]) = f_1([p_1]) + f_2([p_2])$.

We have to show now that this definition does not depend on the "subdivision" chosen, i.e. the way we decompose p onto $p_1 \in P_1^{\alpha,\beta}(X_1)$ and $p_1 \in P_1^{\alpha,\beta}(X_2)$.

If $p = (p_1 - p') + (p_2 + p')$ with $p' \in P_1^{\alpha,\beta}(X_1 \cap X_2)$ then

$$\begin{aligned} [p]_X &= i_1^*([p_1 - p']_{X_1}) + i_2^*([p_2 + p']_{X_2}) \\ &= i_1^*([p_1]) - i_1^*([p']_{X_1}) + i_2^*([p_2]) + i_2^*([p']_{X_2}) \\ &= i_1^*([p_1]) + i_2^*([p_2]) + (i_2^* \circ j_2^*([p']_{X_1 \cap X_2}) - i_1^* \circ j_1^*([p']_{X_1 \cap X_2})) \\ &= i_1^*([p_1]) + i_2^*([p_2]) \end{aligned}$$

The definition of f does not depend on the representant of $[p]_X$ either.

If $[p]_X = [p']_X$ then $p - p' = (\partial_0 - \partial_1)(A)$ where $A \in \underline{X}_2$. We can write $A = A_1 + A_2$ with $A_1 \in \underline{X}_1$ and $A_2 \in \underline{X}_2$ and similarly $p = p_1 + p_2$, $p' = p_1' + p_2'$. Then

$$\begin{aligned} [p]_X &= i_1^*([p_1' + (\partial_0 - \partial_1)(A_1)]_{X_1}) + i_2^*([p_2' + (\partial_0 - \partial_1)(A_2)]_{X_2}) \\ &= i_1^*([p_1']_{X_1}) + i_2^*([p_2']_{X_2}) \end{aligned}$$

Therefore,

$$\begin{aligned} f([p]) &= f_1([p_1']) + f_2([p_2']) \\ &= f([p']) \end{aligned}$$

Now, we prove that Seifert/Van Kampen's theorem is true also for the full fundamental group

$$\Pi_1(X) = \bigoplus_{\alpha,\beta \in X_0} \Pi_1^{\alpha,\beta}(X) / \{[f]_{\alpha,\beta} + [g]_{\beta,\gamma} = [f+g]_{\alpha,\gamma}\}$$

Let $[f]^X \in \Pi_1(X)$. We can decompose it as

$$[f]^X = \sum_{(\alpha,\beta) \in S} [f_{\alpha,\beta}]^X_{\alpha,\beta}$$

where $S \subseteq M_0 \times M_0$ such that $(\alpha, \beta) \in S$ and $(\alpha', \beta') \in S$ implies $\alpha = \alpha'$ and $\beta = \beta'$ or $\alpha \neq \alpha'$ and $\beta \neq \beta'$. We have also $[f_{\alpha,\beta}]^X_{\alpha,\beta} \in \Pi_1^{\alpha,\beta}(X)$ and they decompose as $[f_{\alpha,\beta}]^X_{\alpha,\beta} = i_1^*([f^1_{\alpha,\beta}]^{X_1}_{\alpha,\beta}) + i_2^*([f^2_{\alpha,\beta}]^{X_2}_{\alpha,\beta})$. We have already defined the $g_{\alpha,\beta} : \Pi_1^{\alpha,\beta}(X) \to G$. This leads us to define $g : \Pi_1(X) \to G$ as $g([f]^X) = \sum_{(\alpha,\beta)\in S} g_{\alpha,\beta}([f_{\alpha,\beta}]^X_{\alpha,\beta})$. If well defined, this is the necessary morphism for completing the pushout diagram. To show that it is well defined, we have to show that the formula above does not depend on the "subdivision" or representant chosen of $[f]^X$.
Suppose that $[f]^X = \sum_{(\alpha,\beta)\in S'} [f_{\alpha,\beta}]^X_{\alpha,\beta}$ with an other $S' \subseteq M_0 \times M_0$ verifying the same condition as S. Then,

$$
\begin{aligned}
& g\left(\sum_{(\alpha,\beta)\in S} [f_{\alpha,\beta}]^X_{\alpha,\beta}\right) - g\left(\sum_{(\alpha,\beta)\in S'} [f_{\alpha,\beta}]^X_{\alpha,\beta}\right) \\
= & \sum_{(\alpha,\beta,\gamma)\in T} g([f+h]_{\alpha,\gamma} - [f]_{\alpha,\beta} - [h]_{\beta,\gamma}) \\
= & \sum_{(\alpha,\beta,\gamma)\in T} (g_1([f^1+h^1]_{\alpha,\gamma} - [f^1]_{\alpha,\beta} - [h^1]_{\beta,\gamma}) \\
& \quad + g_2([f^2+h^2]_{\alpha,\gamma} - [f^2]_{\alpha,\beta} - [h^2]_{\beta,\gamma})) \\
= & \ 0
\end{aligned}
$$

where we have used a decomposition of f and h in the last equation. □

Let M_1, M_2 be two sub-HDA of M such that $M_1 + M_2 = M$. Then as $S(M_1 + M_2) = S(M_1) + S(M_2)$ and Van Kampen's theorem holds for Π_1 hence Π_1' by transformation $M \to \tilde{M}$, it holds for Π_n' as well. We take for granted that Van Kampen's theorem holds also for Π_n, $n \geq 2$ (see [BH81b, BH81a] for a similar result).

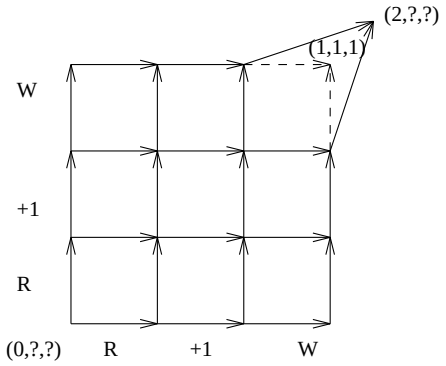
7.8 Some applications

7.8.1 Schedulers

A n-scheduler should basically execute all possible n-paths up to equivalence. This means that a n-scheduler of an automaton D is a choice of a subHDA M of D such that all n-paths of D are equivalent (homotopic) to a n-path of M. We call a scheduler a subautomaton of D which is an n-scheduler for all n. At the light of the previous sections, this is formalized as follows,

Definition 50 *A n-scheduler is a monomorphism (i.e. a cofibration in the homotopy theory we are considering, see [Bau89]) $s : M \to D$ such that $\Pi_n(s) : \Pi_n(M) \to \Pi_n(D)$ is an isomorphism.*

Figure 7.19: Conflict in a shared memory parallel machine/concurrent database.



A scheduler is now a cofibration inducing an isomorphism between all the $\Pi_n(D)$ and the $\Pi_n(M)$. This is known as a **weak equivalence** [Bau89]. A basic property of the homotopy theory we use is that weak equivalence is the same as strong equivalence, i.e. a scheduler is a cofibration such that there exists $s' : D \to M$ with $s \circ s'$ and $s' \circ s$ homotopic to the identity. A scheduler is thus the choice of a weak deformation retract [Spa66] of D.

Example 34

- *The serializability condition may be rephrased into "all 1-schedulers form a subset of the set of classes of the interleavings of the transactions".*

- *In Figure 7.19 we have pictured the semantics of the program $P_1 \mid P_2$ where $P_1 ::= READA; A := A + 1; WRITEA$ and $P_2 ::= READA; A := A + 1; WRITEA$. In the figure we have abbreviated $READA$, $A := A+1$ and $WRITEA$ by respectively R, $+1$ and W. The shapes (deformed squares) are all filled in, indicating concurrency. The states are given by the value of A. To make the picture easy to read we have chosen to unfold things a bit for the central squares, using the value of A read by P_1 (second component of the triple) and the value of A read by P_2 (third component of the triple). The picture thus contains ten squares, the two at the top right corner show the interference while writing the computed value into the shared variable A. From $A = 0$ we can have two different results, $A = 1$ or $A = 2$. Now the two-phase protocol added to the two processes will constrain the execution so that all of P_1 (respectively P_2) is executed before all of P_2 (respectively P_1). These are two equivalent 1-schedulers (linked together by the nine upper squares). The protocol is therefore sound in the sense that it is serializable.*

Obviously the algorithmic characterization of schedulers is given by the weak equivalence condition and not the strong one since we have practical means for computing the homotopy modules (see Part IV for this).

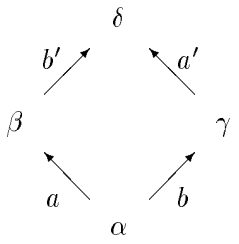
7.8.2 Mutual exclusion

Two actions a and b are in mutual exclusion in a regular automaton M means that all 1-schedulers contain the interleaving $ab' + ba'$. This is equivalent by what we have just seen to $(a - b + b' - a')$ generator of $H_1(M, \partial_0 - \partial_1)$. More generally,

Definition 51 *A k-mutual exclusion or mutual exclusion of dimension k is an element of $\tilde{\Pi}_k(M) = H_k(M, \partial_0 - \partial_1)$.*

Example 35

- *The element X of Υ_{sr} realized geometrically as,*



  has a mutual exclusion of dimension one described by $b-a+a'-b' \in \Pi_1(X)$ since $\partial_0(a' - b') = \gamma - \beta = \partial_1(b - a)$.

- *Suppose X is the boundary of a 3-dimensional cube. X has a mutual exclusion of dimension 2: any two actions can be fired concurrently but no three actions can (a real example is INTEL's Pentium processor, see the introduction).*

7.9 Approximation of schedulers, branchings and mergings

Let M be a complex of modules with differential ∂. Suppose that we have a filtration[6] F on M, i.e.

$$0 \subset F^0M \subset F^1M \subset \ldots \subset F^nM \subset F^{n+1}M = M$$

Then we may define the graded object $GrM = \oplus_{p\geq 0} Gr^pM$ where $Gr^pM = F^pM/F^{p-1}M$. It is a differential module with the map $d : Gr^pM \to Gr^pM$ induced by ∂.

A *spectral sequence* is a sequence $\{E_r, d_r\}$ $(r \geq 0)$ of bigraded objects $E_r = \underset{p,q\geq 0}{\oplus} E_r^{p,q}$ together with homomorphisms $d_r : E_r^{p,q} \to E^{p-r,q+r-1}$ satisfying $d_r^2 = 0$ (i.e. they are differentials). Moreover, for all r, E_{r+1} is the homology of E_r with boundary operator E_{r+1}, i.e. $H_*(E_r) = E_{r+1}$.

The main result is,

[6] It is finite here just for the sake of simplicity. For more technical details, see [Lan93a] or [MC85].

(*) Lemma 12 *There exists a spectral sequence* $\{E_r\}$ *with,*

$$E_0^{p,q} = F^p M_{p+q}/F^{p-1}M_{p+q}$$

$$E_1^{p,q} = H_{p+q}(Gr^p M)$$

$$E_\infty^{p,q} = Gr^p(H_{p+q}(M))$$

(*) PROOF. Here we have to define the limit term $E_\infty^{p,q}$. We are going to construct $E_r^{p,q}$ as the quotient module $Z_r^{p,q}/B_r^{p,q}$ with,

$$B_0^{p,q} \subseteq B_1^{p,q} \subseteq \ldots \subseteq B_\infty^{p,q} \subseteq Z_\infty^{p,q} \subseteq \ldots Z_1^{p,q} \subseteq Z_0^{p,q}$$

and $B_\infty^{p,q} = \bigcup_r B_r^{p,q}$. Under some nice conditions [CE56], we can assume that $Z_\infty^{p,q} = \cap_r Z_r^{p,q}$ making precise the notion of convergence. Here we restrict to an easier case when M is *regular*, i.e. when for all n, there exists an integer $u(n)$ such that $H_n(F^p(A)) = 0$ for $p < u(n)$. This implies that $Z_r^{p,q} = Z_\infty^{p,q}$ for all $r > u(p+q+1) - p$. Therefore we have the convergence of the spectral sequence[7].

The construction promised is then as follows. Let $F^{p,q}M = F^p M_{p+q}$ then,

$$Z_r^{p,q} = \{x \in F^{p,q}M / \partial(x) \in F^{p-r,q+r-1}M\}$$

$$B_r^{p,q} = \partial(Z^{p+r-1,q-r+1}) + Z_{r-1}^{p-1,q+1}$$

□

In all cases, $E_{r+1}^{p,q}$ is a quotient of a submodule of $E_r^{p,q}$. In particular it is of a type at most the type of $E_r^{p,q}$. The spectral sequence process consists in computing the homology of M by *upper approximations*.

The main application for us is the spectral sequence associated with a bicomplex $M = \bigoplus_{p,q \geq 0} M_{p,q}$ with boundary operators $\partial_0 : M_{p,q} \to M_{p-1,q}$ and $\partial_1 : M_{p,q} \to M_{p,q-1}$. We write $((M_n)_n, \partial)$ for the total complex associated with this bicomplex (also named $Tot(M)$). Then we may consider in particular two filtrations on M,

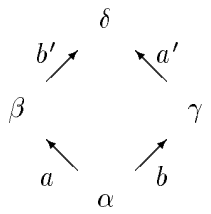
- the *first filtration*, $'F^p M_n = \bigoplus_{p'+q=n, p' \leq p} M_{p',q}$,
- the *second filtration*, $''F^p M_n = \bigoplus_{p+q'=n, q' \leq q} M_{p,q'}$.

Then there are two spectral sequences $\{'E_r\}$ and $\{''E_r\}$ both abutting to $H(Tot(M))$.

Notice that this result still holds if M was only a weak bicomplex (e.g. coming from a cyclic automaton) because we only need to consider $Tot(M)$ which is a complex of modules anyway.

[7] Which can actually be shown to be a direct limit of its terms in this case [CE56].

Example 36 *We show here the computation of the spectral sequence* $\{'E_r\}$ *for a simple HDA (weak bicomplex)* M*,*



All steps of computation are represented geometrically,

$$E_0^{p,*} = \left(\begin{array}{ccccccccc} & \beta & & & & \gamma & & \delta & \\ \alpha, & & \nwarrow & & \nearrow & , & b' \nearrow & & \nwarrow a' \\ & a & & 0 & & b & & 0 & & 0 \end{array} \right)$$

$$E_\infty^{p,*} = E_1^{p,*} = \left(\alpha, 0, 0 \xrightarrow{a'-b'} 0 \right)$$

The spectral sequence associated with the first filtration show how cycles are extracted from the set of mergings (upper approximation of the cycles) whereas the spectral sequence associated with the second filtration show how cycles are extracted from the set of branchings.

Summary We have shown that quite a few notions in computer-science rely on the concept of scheduler, or on some "protocol" for scheduling actions or events in order to have a well-behaved systems. This is the case for protocols for concurrent databases, robust (wait-free, t-resilient) protocols for distributed systems and even for the implementation of parallel languages on constrained architectures (i.e. with finite number of processes and resources).

We have sketched a homotopy theory for semi-regular HDA in which two executions are homotopic means they are serially equivalent, i.e. the essential scheduling properties are preserved between the two executions. We have shown that the homotopy theory for semi-regular HDA was actually a homology theory in general HDA. We will see an extension of this in Chapter 10.

The main difference with "ordinary" homotopy theory is that we do not allow the paths to go in the reverse direction of time. Hence the homotopy groups we defined is some completion of a homotopy monoid in which the monoid operation is concatenation of paths. We defined also in two different ways the higher-order homotopy groups. One is through the definition of higher-order paths. The other uses a "suspension"-like construction. The two are shown to relate in a nice way. We do not know yet if they are fully equivalent.

We also proved Van-Kampen's theorem for the fundamental group (to be used in Chapter 9) and sketched a few combinatorial properties of the fundamental group. It was shown in particular that the difference between the fundamental group we define and the ordinary one is that the relative positions of holes do matter in the former whereas they do not in the latter.

We ended the chapter by the formal definition of schedulers, mutual exclusions and a relationship between the local geometric properties of Chapter 6 (branchings and mergings) and mutual exclusion properties through a spectral sequence.

Chapter 8

Applications of scheduling properties

8.1 Word problems in monoids

The first application we give here deals with the computability of equality using (parallel) term rewriting systems. This application is interesting for two main reasons. First, it deals with the language side of HDA, which we have not developed up to now. Secondly, the computability problem we are interesting in is about confluence of term rewriting systems, which is a geometric property. We relate this problem to the serializability issues of the last chapter.

We first need to recall a few elements about the presentation of monoids and about the homology of monoids.

8.1.1 Presentation of monoids

A presentation of a monoid M is a pair (S, R) with,

- S is a set of generators,
- R is a set of relations $v \sim w$ between words v, w over S.

such that M is the quotient of the free monoid S^* by the congruence associated with R.

When both S and R are finite we say that M is finitely presented.

A rewriting system is nothing but a presentation (S, R) (of a monoid of words) where we orient each relation of R. We write $r : v \to w$ for such an oriented rule, or reduction rule r in R.

8.1.2 Homology of monoids

Let $(M, 1, \cdot)$ be a monoid. The ring $\mathbb{Z}M$ of M is the free $\mathbb{Z}$-module generated by M. The internal multiplication is a simple extension of the multiplication $\cdot$ in

M. To be more precise, an element of $\mathbb{Z}M$ is a finite formal sum $\sum_i n_i m_i$ where $n_i \in \mathbb{Z}$ and $m_i \in M$. Then, given two elements $x = \sum_i n_i m_i$ and $x' = \sum_i n'_i m'_i$, their product is $z = x \cdot y = \sum_{i,j} (n_i n'_j) m_i \cdot m'_j$.

A (left) $\mathbb{Z}M$-module K is therefore a $\mathbb{Z}$-module together with a (left) linear action of M on K, i.e. a function $* : M \times K \to K$ such that $1 * k = k$ and $(m.m') * k = m * (m' * k)$. Then we can understand the external multiplication $x \cdot k$ of an element k of K by an element $x = \sum_i n_i m_i$ as meaning $\sum_i n_i (m_i * k)$. $\mathbb{Z}$ itself can be endowed with a $\mathbb{Z}M$-module structure by specifying a trivial action of M onto $\mathbb{Z}$. This will be the structure of $\mathbb{Z}M$-module for $\mathbb{Z}$ throughout this chapter.

Now, a free resolution of $\mathbb{Z}$ by (left) $\mathbb{Z}M$-modules is an exact sequence,

$$\ldots \xrightarrow{\partial_3} C_2 \xrightarrow{\partial_2} C_1 \xrightarrow{\partial_1} C_0 \xrightarrow{\epsilon} \mathbb{Z} \longrightarrow 0 \qquad (\mathcal{R})$$

Such resolutions always exist (for instance the Bar resolution, [ML63]). We will specifically construct one for monoids presented by rewrite rules.

We will actually present these resolutions in a geometric manner, where the C_i will present different steps of the construction of a geometric shape. This geometric shape is nothing but a contractible space X on which M acts freely on the left.

As there are many different such resolutions, or geometric constructions for the same object, we are in a need for an algebraic structure characterizing these in the sense that they will not differ on different resolutions. This can be called an invariant as for, say, Betti numbers for topological spaces modulo homotopy. The invariant we consider here is the homology groups defined as follows.

First, we need to define the tensor product of a $\mathbb{Z}M$-module C by $\mathbb{Z}$ over $\mathbb{Z}M$, $\mathbb{Z} \otimes C$. $\mathbb{Z} \otimes C$ is the $\mathbb{Z}$-module whose elements are those of C modulo the identification of $x \in C$ with $m.x$ for $m \in M$. This tensorization is not a (left) exact functor, i.e. it transforms $(\mathcal{R})$ into a sequence which may not be exact. The defect of exactness is measured by the homology groups (or homology $\mathbb{Z}$-modules),

$$H_n(M) = Ker\,(\mathbb{Z} \otimes \partial_n)/Im\,(\mathbb{Z} \otimes \partial_{n+1})$$

It can be shown that this sequence does not depend on a particular choice of a free resolution, but only on M.

Geometrically, this tensorized sequence represents the space of orbits on X under the action of M. Its homology is the "algebraic" homology we have just been defining.

8.1.3 An introduction: Squier's method

Here, we construct a particular resolution for a monoid M which is finitely presented by rewrite rules (S, R) in the style of [LP90]. The basic idea is that the monoid M represents a set of traces (on the alphabet S) modulo

some equivalence relation (generated by R). A natural space on which M acts can be based on the automaton which accepts the "language" M, at least in dimension less or equal than one. Higher-dimensional transitions will describe the relations R in such a way that "equivalent" traces modulo R will correspond to "equivalent" traces modulo serializability.

For $x \in S^*$ we write $\overline{x}$ for the equivalence class of x induced by R. We construct explicitely the first few terms C_i of a free resolution of M.

C_0 is the $\mathbb{Z}$-module of points. We associate one point to every element of M. This means that $C_0 = \mathbb{Z}M$ and as M is presented by (S, R), the generators of C_0 are the $\overline{x}$, $x \in S^*$. We can set $\epsilon(m) = 1$ for all $m \in M$, therefore $\epsilon(\sum_i n_i m_i) = \sum_i n_i$.

C_1 is the $\mathbb{Z}$-module of edges between points. We ask for all traces of S^* to be accepted so C_1 should be generated (as a $\mathbb{Z}$-module) by all elements of S^*, i.e. words on S. But these traces may start from any point of C_0. A trace s starting at $\overline{x}$ is identified with the formal element $\overline{x}[s]$ (compare with the sequential composition in the HDA semantics of CCS, Section 5.4). So, as a $\mathbb{Z}$-module, C_1 is generated by $\overline{x}[s]$, $x, s \in S^*$ and as a $\mathbb{Z}M$-module, it is generated by S^*. The start boundary of $\overline{x}[s]$ is obviously $\partial_1^0(\overline{x}[s]) = \overline{x}$ whereas its end boundary should be $\partial_1^1(\overline{x}[s]) = \overline{x}\,\overline{s} = \overline{xs}$. It is easy to verify that the total boundary operator $\partial_1 = \partial_1^1 - \partial_1^0$ is such that $\epsilon \circ \partial_1 = 0$.

Let us stop for a moment and give a few examples of what we have constructed up to now. It is nothing but the skeleton of dimension one of an HDA.

Example 37 *Let M be the monoid presented by (S, R) with,*

- $S = \{a, b\}$,
- $R = \{ab \to ba\}$.

Then M can be identified with $\mathbb{N} \times \mathbb{N}$ with pointwise addition since canonical words are $b^i a^j$ and $(b^i a^j).(b^{i'} a^{j'}) = b^{i+i'} a^{j+j'}$. It is easy to check that the construction above gives rise to the grid shaped HDA of Figure 8.1.

It is perfectly clear that the skeleton of dimension one we have just built is connected and that the monoid acts freely on the left on it. What remains to be done is to interpret the process of normalization of words, i.e. the equivalence generated by R. In the example above, all interleavings of a and b are equivalent if they have the same count of a and b. This can be seen by deforming locally all ab in some trace onto ba. In the HDA context, the deformation should occur through a two transition whose boundary will be the interleaving $ab + ba$. We do the same in the construction of the resolution.

Let C_2 be the $\mathbb{Z}M$-module generated by the reduction rules $r : v \to w$. As a $\mathbb{Z}$-module, it is generated by all the $\overline{x}[r]$ where $r \in R$ and $x \in S^*$: these are the "translated" reduction rules i.e. the reduction rules applied only at the part of traces beginning at $\overline{x}$.

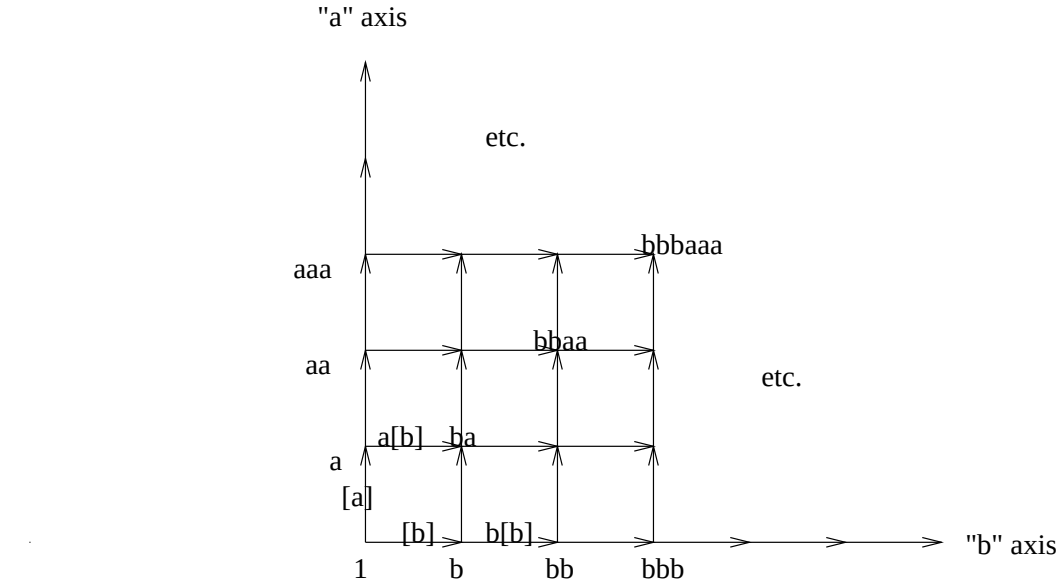
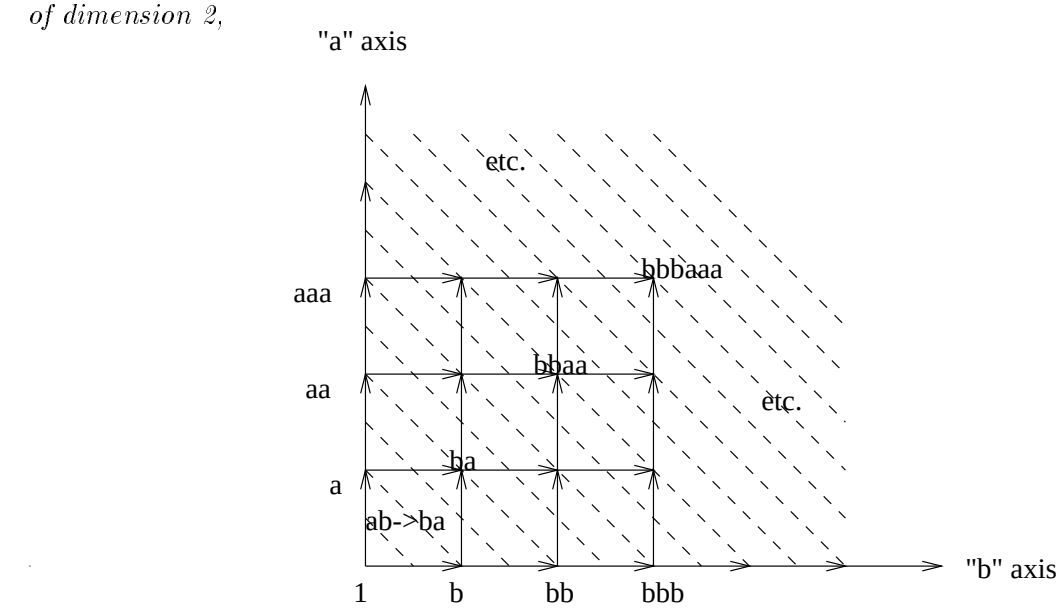


Figure 8.1: A resolution up to dimension one .

Now, the only clear thing is the definition of the "total" boundary ∂_2 from C_2 to C_1. We set $\partial_2(\overline{x}[r]) = \overline{x}[v] - \overline{x}[w]$. We can verify that $\partial_1 \circ \partial_2 = 0$.

In the case where M does correspond to some partially commutative monoid (i.e. to a Mazurkiewitz trace model, see Chapter 1), as in Example 37, it is easy to find a decomposition of the total boundary operator into a start and end operator, making the geometric shape we are building into a skeleton of dimension 2 of an HDA. It is exemplified below,

Example 38 *We are carrying on with Example 37, now picturing the skeleton of dimension 2,*



The reduction rule $r : ab \to ba$ has start boundary $\partial_2^0(r) = [a] - [b]$ and end

boundary $\partial_2^1(r) = \overline{b}[a] - \overline{a}[b]$. *Notice that the sense in which we decide to orient the reduction rule decides of the signs (i.e. for semi-regular automata, decides of the order in which we take all start and end boundaries) in the start and end boundary operators.*

What should we do in higher dimensions? Remember that we want a contractible space X and the space we have been constructing might well not be. It is the case when two reduction rules apply to the same word and do not lead to the same result. Let $u = w_1 r w_2$ and $v = w_1 s w_2$ be two elementary reductions of the same word for which w_1 and w_2 are maximal subwords and r and s are two different reduction rules. Then the pair (r, s) is called a critical pair. Critical pairs measure the conflicts in the reduction system. The idea now is to have the elements of X_3 being the critical pairs. This enables us to fill the holes in X in order to construct a contractible space. Then we have to fill in the holes in X between the 3-cells (the critical pairs). This is done by setting X_4 to the set of "critical triples" and so on. In this way, the resolution procedure amounts to a simple Knuth-Bendix completion procedure.

If there is a finite canonical term rewriting system presenting M, then the completion procedure terminates. This means that the set of critical pairs, critical triples etc. are finite. This in turn implies that all the X_i are finitely generated, hence the homology groups (which are quotient of some submodules of the X_i) are finitely generated. This is known as property "FP_∞". It can be shown that some monoids which have a decidable word problem are not FP_∞, hence cannot be decided by any finite canonical rewriting system. This result is thus a geometric characterization of what sort of monoid can compute a finite canonical rewriting system.

Unfortunately, Squier's construction is uneasy to understand in terms of reduction machines or to formalize within the HDA framework. In order to be more precise about what geometry there is in finite canonical systems, we use another construction of resolutions proposed by Groves.

8.1.4 Groves' construction

Groves in [Gro91] constructs a resolution of the RM-module R using cubical complexes. We recast his construction into the HDA framework, hence the "homotopy relations" of [SOK94] will be real homotopies in the homotopy theory of oriented paths we have defined in Section 7.4.

The principle of Groves' resolution is to have the contractible space X on which the monoid M acts built as follows,

- the vertices (generating C_0 in the resolution) are the words of Σ^*. Elements of M are identified with R-irreducible words,
- the edges (generating C_1) are the instances of a single application of a rewriting rule in R,
- then a suitable covering of this 1-skeleton is constructed using squares (generating C_2), cubes (C_3) etc.

We briefly review this construction below. Notice that X is the HDA describing the reduction of words of Σ^*.

First, we need a few graph-theoretic notions.

Let $\underline{n} = \{0, \ldots, n-1\}$. $2^{\underline{n}}$ is a directed graph with vertices $V(2^{\underline{n}}) = 2^{\underline{n}}$ and edges $E(2^{\underline{n}}) = \{(S,x)/S \in 2^{\underline{n}}, x \in \underline{n} \backslash S\}$. An edge (S,x) of Γ has beginning $d^0(S,x) = S$ and end $d^1(S,x) = S \cup \{x\}$.

If Δ is a directed graph, a n-cube μ in Δ consists of a pair of maps,

$$\mu_V : V(2^{\underline{n}}) \to V(\Delta)$$

$$\mu_E : E(2^{\underline{n}}) \to P(\Delta)$$

where $P(\Delta)$ is the set of paths in Δ, such that μ_V takes the initial and terminal points of $e \in E$ to the initial and final points of $\mu_E(e) \in P(\Delta)$.

These graph-theoretic notions give rise to an HDA structure. Let $\delta_i : \underline{n-1} \to \underline{n}$ $(i = 0, \ldots, n-1)$ be defined by,

$$\delta_i(j) = \begin{cases} j & \text{if } j < i \\ j+1 & \text{if } j \geq i \end{cases}$$

It is well known (from the simplicial world, see [May67] and Appendix B) that $\delta_i \delta_j = \delta_j \delta_{i+1}$ $(0 \leq i < j \leq n-1)$.

We now have boundary maps,

$$\delta_i^\epsilon : 2^{\underline{n-1}} \to 2^{\underline{n}}$$

for $\epsilon = 0, 1$ and $i = 0, \ldots, n-1$ defined by, for $\{x_1, \ldots, x_k\} \in 2^{\underline{n-1}}$,

$$\delta_i^0(\{x_1, \ldots, x_k\}) = \{\delta_i(x_1), \ldots, \delta_i(x_k)\}$$

$$\delta_i^1(\{x_1, \ldots, x_k\}) = \delta_i^0(\{x_1, \ldots, x_k\}) \cup \{i\}$$

which verify,

$$\delta_i^k \delta_j^l = \delta_j^l \delta_{i-1}^k$$

for all i, j, k, l with $k = 0, 1$, $l = 0, 1$ and $0 \leq j < i \leq n-1$.

PROOF. For $\{x_1, \ldots, x_k\} \in 2^{\underline{n}}$ and $i < j$,

$$\begin{aligned} \delta_i^0 \delta_j^0(\{x_1, \ldots, x_k\}) &= \delta_i^0(\{\delta_j(x_1), \ldots, \delta_j(x_k)\}) \\ &= \{\delta_i \delta_j(x_1), \ldots, \delta_i \delta_j(x_k)\} \\ &= \{\delta_j \delta_{i-1}(x_1), \ldots, \delta_j \delta_{i-1}(x_k)\} \\ &= \delta_j^0 \delta_{i-1}^0(\{x_1, \ldots, x_k\}) \end{aligned}$$

Now,

$$\begin{aligned} \delta_i^0 \delta_j^1(\{x_1, \ldots, x_k\}) &= \delta_i^0 \delta_j^0(\{x_1, \ldots, x_k\}) \cup \delta_i^0(\{j\}) \\ &= \delta_j^0 \delta_{i-1}^0(\{x_1, \ldots, x_k\}) \cup \{j\} \\ &= \delta_j^1 \delta_{i-1}^0(\{x_1, \ldots, x_k\}) \end{aligned}$$

The other commutation relations are proved in a similar way. This is left to the reader. □

This implies that a n-cube of a graph Δ creates a HDA generated by one n-transition, the n-cube itself, and whose objects of lower dimensions are its iterated boundaries,

$$d_i^\epsilon(f) = f \circ \delta_i^\epsilon : 2^{\underline{k-1}} \to \Delta$$

where $f : 2^{\underline{k}} \to \Delta$. This construction will be used in Chapter 10 where it will be generalized.

The fact that some boundaries of a 2-cube may not be just instances of reduction rules, i.e. edges in the graph Γ, but rather paths in Γ is a problem for casting Groves' construction into the HDA framework. We choose to construct the 1-transitions to be the paths in Γ. Let X be the HDA corresponding to Groves construction, we set,

- $X_0 = \Sigma^*$,
- $X_1 = P(\Gamma)$.

with the obvious boundary maps. n-cubes in X now correspond to morphisms of HDA

$$\mu : T_1(2^{\underline{n}}) \to X$$

where $2^{\underline{n}}$ is considered as the n-dimensional HDA described above and T_1 is the truncation functor of Section 2.2.1.

Some shapes in the reduction system are of interest here. We know (see previous section) that we are interested in some critical pairs, i.e. in conflicts, or non-determinism in the reduction relation. These are introduced here under the name "n-stars". n-stars are "morally" branchings of dimension n in X, that should be filled in the resolution. These are $[w; e_1, \ldots, e_n]$ where $w \in X_0$ and the e_i are 1-transitions in X_1 which begin at w. These 1-transitions can be empty paths, in that case the n-star is called degenerate, and can also be repeated in the sequence $e_1, \ldots, e_n$. There is a natural product on n-stars induced by the concatenation in Σ^*.

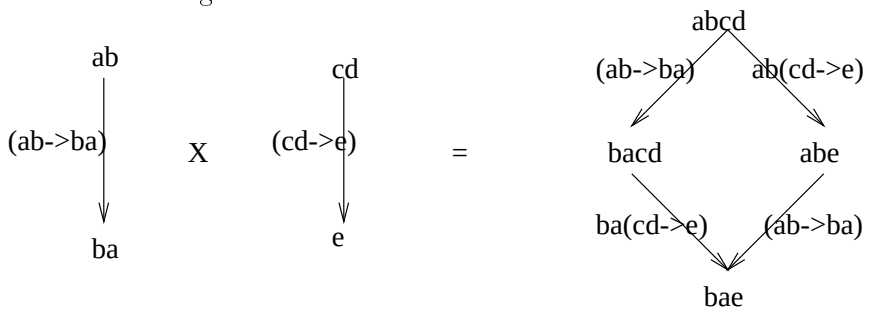
Let $[w_1; e_1, \ldots, e_k]$ be a k-star and $[w_2; e_{k+1}, \ldots, e_{k+l}]$ be a l-star. Then their product yields a $(k+l)$-star as follows,

$$[w_1; e_1, \ldots, e_k].[w_2; e_{k+1}, \ldots, e_{k+l}] = [w_1.w_2; e_1.w_2, \ldots, e_k.w_2, w_1.e_{k+1}, \ldots, w_1.e_{k+l}]$$

As Σ^* is a free monoid, every n-star admits a unique decomposition as a product of indecomposable stars. Indecomposable n-stars which have neither empty edges nor repeated edges are called critical. In particular,

- a critical 0-star is an element of Σ,
- critical 1-stars are in $1-1$ correspondence with rules in R,
- critical 2-stars can be identified with critical pairs.

Figure 8.2: Product of two 1-cubes



We can also define a product for n-cubes. Let

$$\mu : T_1(2^{\underline{k}}) \to X$$

$$\nu : T_1(2^{\underline{l}}) \to X$$

be a k-cube and a l-cube respectively. Their product $\mu \times \nu$ is a $(k+l)$-cube,

$$\mu \times \nu : T_1(2^{\underline{k}} \otimes 2^{\underline{l}}) \to X$$

with

$$\mu \times \nu(x, y) = \mu(x).\nu(y)$$

The interpretation of the product of n-cubes is easy (look at Figure 8.2). A k-cube μ (respectively a l-cube ν) in X represents the parallel execution of k (respectively l) reduction rules. Their product represents the parallel reduction of the k reduction rules acting on the left part of the word $\mu(\emptyset).\nu(\emptyset)$ (i.e. $\mu(\emptyset)$) and of the l reduction rules acting on the right part of the same word (i.e. $\nu(\emptyset)$). The product of cubes is therefore the parallel product without interference of disjoint reduction rules.

To every non-degenerate n-cube μ we can associate a n-star $[\mu(\emptyset); \mu(\{0\}), \ldots, \mu(\{n-1\})]$. Conversely, to any n-star (which we suppose first indecomposable) $[w; e_1, \ldots, e_n]$ we can associate a n-cube as follows (see Example 39). Let $S \subseteq \underline{n}$ be a vertex in $2^{\underline{n}}$. We decompose the word w on which the reduction rules e_i act as the concatenation

$$w = a_1 u_1 a_2 \ldots u_{l-1} a_l$$

where the u_i are the largest blocks on which some rules e_i, $i \in S$ overlap. We set

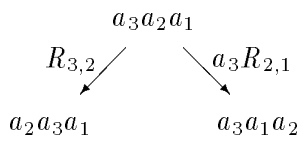
$$\mu(S) = a_1 \overline{u_1} a_2 \ldots \overline{u_{l-1}} a_l$$

and $\mu(S, i)$ is any (reduction) path from $\mu(S)$ to $\mu(S \cup \{i\})$.

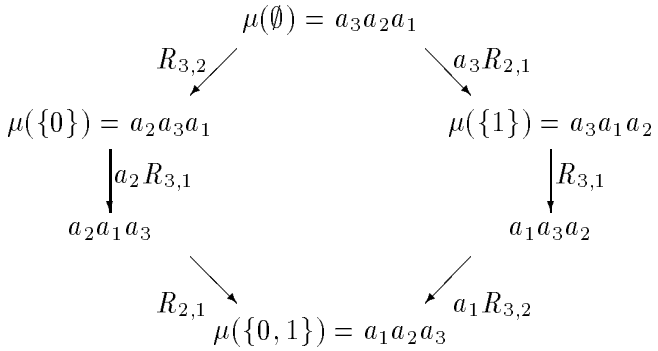
Example 39 *Let* $\Sigma = \{a_1, a_2, a_3\}$ *and* $R = \{R_{j,i} / 1 \leq i < j \leq 4\}$ *where the* $R_{j,i}$ *are the rules,*

$$a_j a_i \to a_i a_j$$

(Σ, R) presents the free commutative monoid on three generators. One critical 2-star is,



and its associated (canonical) 2-cube μ is,



The d_0^1 boundary of this 2-cube is the sequence of reductions $(a_2R_{3,1}, R_{2,1})$. The d_1^1 boundary of μ is the sequence of reductions $(R_{3,1}, a_1R_{3,2})$.

Now the main result of [Gro91] is,

Theorem 2 *There is a RM-resolution of R,*

$$\ldots \to P_n \to P_{n-1} \to \ldots \to P_0 \to P_{-1} \to R$$

where,

- $P_{-1} = RM$,
- P_n $(n \geq 0)$ *is the RM module generated by the critical n-stars.*

The proof is rather difficult, and in particular, the definition of the boundary operators depends on auxiliary operators on n-cubes. We will only describe this construction in a particular case where it gets simpler.

To illustrate the relationship with the HDA approach, we prove now that there exists a bigger resolution in terms of HDA when we begin with a **strongly confluent rewriting system** (Σ, R) (as the standard presentation of a monoid),

Theorem 3 *There is a RM resolution of R by an HDA X,*

$$\ldots \to X_n \to X_{n-1} \to \ldots \to X_0 \to RM \to R$$

Where,

- X_0 *is the RM-module generated by Σ^*,*
- X_1 *is the RM-module generated by rules of R,*

- *X_n ($n \geq 2$) is the R-module generated by the n-cubes of the graph Γ defined by X_0 (vertices), X_1 (edges).*

where the boundary operators $\delta_n : X_n \to X_{n-1}$ are the total boundary operators $\partial_0 - \partial_1$ of HDA.
Moreover, each X_n is finitely generated as a RM-module. This implies that M has property FP_∞.

SKETCH OF PROOF. X_n ($n \geq 2$) can be given the structure of RM-module as follows. Let $\mu : T_1(2^{\underline{n}}) \to \Gamma$ be an element of X_n and $m \in M$. Then the action of m on μ is given by the product of the 0-cube m by the n-cube μ which is an element of X_n. As an RM-module, X_n is generated by the n-critical pairs of R. Therefore it is finitely generated if (Σ, R) is a finite strongly confluent rewriting system.

The total boundary operator $\delta_n = \partial_0 - \partial_1 : X_n \to X_{n-1}$ is a R-homomorphism verifying $\delta_n \delta_{n-1} = 0$. We first have to verify that it is a RM-homomorphism. Let $m \in M$ and $\mu \in X_n$, then,

$$\mu : T_1(2^{\underline{n}}) \to \Gamma$$

$$m : \{\emptyset\} \to \Gamma$$

$$m.\mu : T_1(2^{\underline{n}}) \to \Gamma$$

with $(m.\mu)(S) = m.(\mu(S))$. Therefore

$$\begin{aligned}
\delta_n(m.\mu)(S) &= \sum_{i=0}^{n-1} (d_i^0 - d_i^1)(m.\mu)(S) \\
&= \sum_{i=0}^{n-1} \left((m.\mu) \circ \delta_i^0 - (m.\mu) \circ \delta_i^1 \right)(S) \\
&= \sum_{i=0}^{n-1} m.\left(\mu \circ \delta_i^0 - \mu \circ \delta_i^1 \right)(S) \\
&= m.(\delta_n(\mu))(S)
\end{aligned}$$

Finally, we have to verify that the sequence,

$$\ldots \xrightarrow{\delta_{n+1}} X_n \xrightarrow{\delta_n} \ldots \xrightarrow{\delta_1} X_0 \xrightarrow{\delta_0} RM \xrightarrow{\epsilon} R$$

is exact, i.e. that it defines a resolution of the RM-module R. We just have to verify it for $n \geq 2$. Let $x \in X_n$ be a cycle for δ_n. We can suppose that x is a cycle of length 2 as all cycles are generated by cycles of length 2 (by strong confluence). x generates a sum of n-stars $x_0 = r_1[w_1; e_1^1, \ldots, e_n^1] + \ldots + r_n[w_k; e_1^k, \ldots, e_n^k]$ (with coefficients $r_i \in R$) by considering the summands x_0 in x with $\partial_0(x_0) = 0$. By strong confluence, all these n-stars $[w_i; e_1^i, \ldots, e_n^i]$ can be completed by $(n+1)$-cubes c_i. It is an easy verification to see that $r_1 c_1 + \ldots + r_k c_k$ is a $(n+1)$-cube with total boundary x. □

A similar resolution by HDA in the case of confluent (and not only strong confluent) rewriting systems exists with X_1 being generated by all finite sequences of rules in R. It is unfortunately too big to prove the FP_∞ property since this resolution is infinite dimensional in general.

In a very particular case, things are getting even closer to the standard computational means used by mathematicians to calculate the homology of monoids (or more generally of associative algebras). This case is as follows.

The standard presentation by a rewriting system (Σ, R) for a monoid M is given by,

- $\Sigma = M \backslash \{1\}$,
- $R = \{ab \to \overline{ab} / a, b \in \Sigma\}$.

The n-stars $[w; e_1, \ldots, e_n]$ are,

- $w = m_1 \ldots m_{n+1}$,
- e_i is an application of the rule $m_i m_{i+1} \to \overline{m_i m_{i+1}}$.

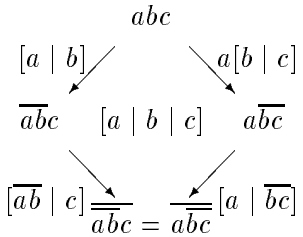
The boundary operators $d_n : P_n \to P_{n-1}$ of Groves' resolution are defined as, writing w as $[m_1 \mid \ldots \mid m_{n+1}]$,

$$\begin{aligned} d_n(w) &= m_1[m_2 \mid \ldots \mid m_{n+1}] \\ &\quad + (-1)^{n+1}[m_1 \mid \ldots \mid m_n] \\ &\quad + \sum_{i=1}^{n}(-1)^i[m_1 \mid \ldots \mid \overline{m_i m_{i+1}} \mid \ldots \mid m_{n+1}] \end{aligned}$$

We recognize here the normalized Bar resolution [ML63]. We would have got the unnormalized one if we had chosen a similar presentation with $\Sigma = M$.

Notice that the boundary operators (Bar resolution) in the case of the standard presentation of a monoid are exactly the total boundary operators of the HDA X (look at Example 40).

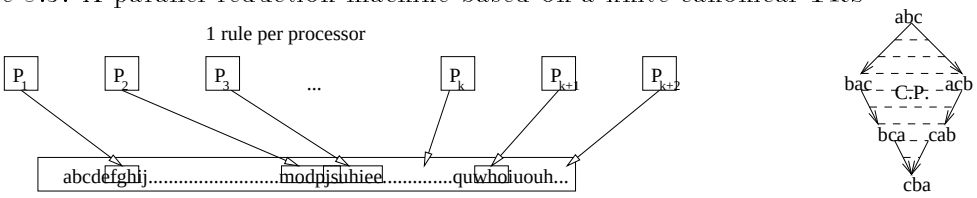
Example 40



and the boundary d_2 is the total boundary operator,

$$d_2([a \mid b \mid c]) = a[b \mid c] - [a \mid b] - [\overline{ab} \mid c] + [a \mid \overline{bc}]$$

Figure 8.3: A parallel reduction machine based on a finite canonical TRS



The coincidence between the methods and results proved here on monoids and what we have seen about serializability and homotopy for HDA can actually be explained operationally.

Squier's result is about what can compute the normalization function of a finite canonical rewriting system. To be more precise and relate this computability result to results in distributed computing we define a parallel reduction machine $\mathcal{M}$ (see Figure 8.3) for a given finite canonical rewriting system (Σ, R).

Let $R = \{R_i / i = 1, \ldots, n\}$. Then $\mathcal{M}$ has n processors, each of which having to execute the reduction only by rule R_i on the shared word w. Each time any of the processor has to manipulate part of w, it locks the letters to be reduced and after the reduction unlock this part of the word. Thus M is nothing but a finite transaction system as we have seen in Section 7.1.2.

Now, the geometrical property of this system (or we should say of its HDA semantics) is that all paths of execution converge to the same result, i.e. that all paths are serializable. In fact we have even more in that all serializations give the same result, but this property is not one of the "geometric" properties we have characterized. This implies that this machine has a finitely generated fundamental group of oriented paths. This corresponds to the finiteness condition (in terms of "homotopy relations") of [SOK94].

8.2 Results in protocols for distributed systems

8.2.1 A quick survey

The early results about protocols for distributed systems were using graph theory.

In [BMZ88] a characterization of a class of problems solvable in asynchronous message-passing systems in the presence of a single failure was given. No generalization to more failures have since been solved using the same kinds of graph techniques.

It was then a rather shared belief that one would have to use more powerful techniques in that case.

The conjecture [Cha90] that the k-set agreement problem cannot be solved in certain asynchronous systems was finally proven in three different papers independently, [BG93], [SZ93] and [HS93].

The renaming task, first proposed in [ABND$^+$90] was finally solved in [HS93]. There is a wait-free protocol for the renaming task in certain asynchronous

systems if the output name space is sufficiently large. It was already known that there is a wait-free solution for the renaming task for $2n+1$ or more output names on a system of $n+1$ asynchronous processors and none for $n+2$ or fewer output names. Herlihy and Shavit refined this result and showed that there was no solution for strictly less that $2n+1$ output names.

It was known since [FLP85] that the consensus task was impossible to solve if processes could fail. We actually explain this result in geometric terms using HDA and Herlihy's construction in Section 8.2.3. But it was also long known that a FIFO queue could solve wait-free consensus on a system of two asynchronous processors.

In fact, if we define the consensus number of a data type as the maximal number of asynchronous processors (having atomic read and write) on which it can implement wait-free consensus, then,

- atomic Read/Write registers have consensus number 1,
- test&set and fetch&add registers, queues, and stacks have consensus number 2,
- n-register assignment has consensus number $2n-2$,
- load-locked, store-conditional and compare and swap registers have consensus number ∞.

These facts motivated the introduction of the following general problem, really about the power of the architecture of distributed machines. We say that a datatype, or object, is a (m,j)-consensus object if it allows any m-processes to solve j-set agreement tasks. Herlihy and Rajsbaum in [HR94] (see also [BG93]) proved that is is impossible to implement $(n+1,k)$-consensus using (m,j)-consensus objects if $n/k > m/j$.

For wait-free protocols, it has be shown [HS94] how to derive the protocol from the decision maps in a constructive manner. We recast this in a more general semantic framework in Chapter 10.

8.2.2 Herlihy's framework

The geometric framework of Herlihy et al. [Her94] is based on a representation of the input and output specifications in the form of input and output simplicial complexes (see Appendix B).

A simplex is associated to the states of processes in the following manner,

- vertices v are pairs $(val(v), id(v))$ of local values of process having identifier $id(v)$,
- we have an edge between v and v' if and only if v and v' are compatible with the specification we have of the state of the system. It means in all cases that v and v' must have distinct process identifiers.

Figure 8.4: Input complex for the binary consensus.

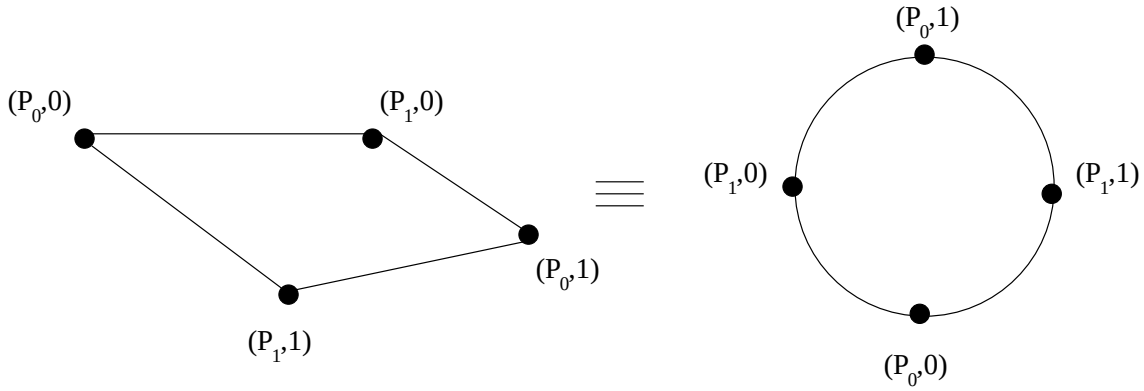
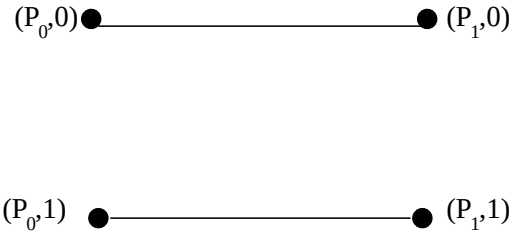


Figure 8.5: Output complex for the binary consensus.



- higher-dimensional simplexes include states $v_1, \ldots, v_n$ if and only if these states are compatible with the specification (input or output one). Again, all the $id(v_i)$ are distinct.

As an example, the consensus task with $S = \{0, 1\}$ and N processes, called the *binary consensus task* has as specification complexes,

- the input complex has simplexes of the form $((P_0, b_0), \ldots, (P_n, b_n))$ with $0 \leq n \leq N - 1$ and $b_i \in \{0, 1\}$ since all processes can take whatever boolean value they want. It is homeomorphic to a N-sphere (see Figure 8.4),

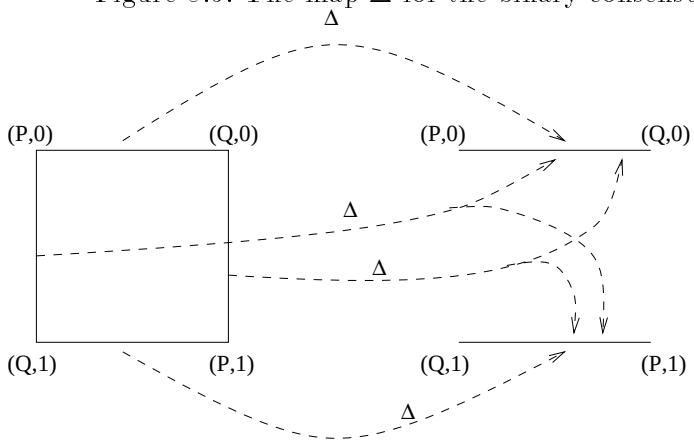
- the output complex has simplexes of the form $((P_0, 0), \ldots, (P_n, 0))$ or $((P_0, 1), \ldots, (P_n, 1))$ since all processes should agree on some common boolean value. This complex has exactly N connected components (see Figure 8.5).

In full generality, a $(n + 1)$-process decision task $\langle I, O, \Delta \rangle$ is given by an input complex I, an output complex O and a recursive (computable) map Δ carrying each m-simplex of I $(0 \leq m \leq n)$ to a subcomplex of O of dimension n such that,

- for all $s \in I$, $dim\ s = m$, $id(s) = id(\Delta(s))$,

- if $s \subseteq s'$ are two simplexes of I then $\Delta(s) \subseteq \Delta(s')$.

Figure 8.6: The map Δ for the binary consensus.



The map Δ specifies what are the allowed outcomes of computations from given starting values in I (look at Figure 8.6 for an example).

A t-resilient protocol solves the $(n+1)$-process decision task if and only if there is a simplicial set P, called the protocol complex, and a simplicial map $\delta : P \to O$ such that

$$\forall s \in I, n - t \leq dim\ s \leq n, \forall u \in P(s), \delta(u) \in \Delta(s)$$

where $P(s)$ is the subcomplex of the protocol complex generated by the executions in which only processes in $id(s)$ take steps, starting with input values from $val(s)$.

This is actually just saying that a protocol solves a decision task if and only if it yields an output value permitted by the decision task from any given input state. Here we have gained the fact that simplicial maps preserve some topological properties of complexes. Hence impossibility results for finding protocols for some decision tasks arise from the topologically incompatible nature of its input and output complexes.

Then, an algorithm for solving such and such decision tasks can be given in geometrical terms, relating the geometry of the input and output complexes.

8.2.3 Wait-free protocols

Here we are interested in the main result of [HS94] which can be roughly stated as follows: "There is a wait-free protocol for solving a given decision task if and only if its input complex can be continuously stretched and folded to cover its output complex".

To relate this to our framework, we first need to define input and ouput complexes.

Let $P_0, \ldots, P_{N-1}$ be N sequential processes which, when run in parallel verify a protocol $\mathcal{P}$ solving a decision task $\mathcal{D}$. For the sake of simplicity, we suppose that the processes P_i are reduced to one action a_i. The HDA representing the

program in the shared-memory paradigm is then $(a_0) \otimes \ldots \otimes (a_{N-1})$. The local input values are collected in the initial state $\alpha_0 \otimes \ldots \otimes \alpha_{N-1}$ and the vertices v_i in the input complex $(val(v_i), Id(v_i))$ with $val(v_i) = \alpha_i$ can be identified with the 1-transitions a_i. Compatibility of the vertices means that they begin a fully asynchronous execution of the processes $Id(v_i)$. This fully asynchronous execution is represented in the HDA model by the N-transition $a_0 \otimes \ldots \otimes a_{N-1}$ and in Herlihy's model by an $N-1$ simplex containing all the v_i. Once more, there is a perfect geometric correspondence between the two models, except the dimension has to be shifted by one. We should think of the HDA model as the "extension in time" of the simplex-based model. In other terms, the input/output complexes are a kind of very useful denotational approximation of the operational behaviour given by HDA. Let us be a bit more precise about that.

The final state $\beta_0 \otimes \ldots \otimes \beta_{N-1}$ contains all output values of the processes. In a similar manner, the part of the output complex with vertices v_i such that $val(v_i) = \beta_i$ can be identified to (up to a shift of dimension one) the "vertical" complex composed of all end boundaries of $a_0 \otimes \ldots \otimes a_{N-1}$.

More generally, let I and O be input and output complexes of a protocol $\mathcal{P}$. Let $(s_i)_i$ and $(t_i)_i$ be the maximal simplexes in I and O respectively with $s_i = (v_i^0, \ldots, v_i^k)$ $(dim\ s_i = k)$ and $t_i = (w_i^0, \ldots, w_i^k)$ $(dim\ t_i = k)$. We suppose we have a semi-regular HDA D (called domain of HDA in Chapter 5) in which we can fire any transition that the distributed system we are considering can execute, from any state of this machine. By what we have seen previously, the simplices s_i are in one-to-one correspondance with $(dim\ s_i + 1)$-transitions in D with initial state $v_i^0 \otimes \ldots \otimes v_i^{dim\ s_i}$. As a matter of fact suppose that the initial states permitted by the protocol are all in $D_{0,0}$ then there is a sub-semi-simplicial complex of D, isomorphic to the input complex (seen as a semi-simplicial set). This subcomplex is the "horizontal" complex $((D_{n+1,0})_n, (d_i^0)_i)$. The output complex is isomorphic to some subcomplex of D, which we identify now with $((D_{k,n+1-k})_n, (d_i^1)_i)$ if all final states are in $D_{k,-k}$. In between, there are all the paths transforming the input into the output complex (see Figure 8.7).

We need now to see what is a wait-free computation in the HDA model, and why this implies that some topological properties are preserved from the input to the output complexes.

Look at Figure 8.8. In (i), the initial state is an internal non-deterministic choice. If we are not so lucky, the execution will begin by P_2 which fails to terminate: P_1 will never proceed and the computation is certainly not wait-free. In (ii), the execution is asynchronous between P_1 and P_2 and whatever happens to P_2, P_1 will terminate (one of the possible 1-paths is pictured). Hence, intuitively, to go from state α to state β in a wait free manner, we must have $\Pi_1^{\alpha,\beta}$ reduced to one class of paths. In the schedulers' point of view, this is the same as asking for the possible reordering for all executions of the failing processes after the terminating ones. We prove now that this implies that the input complex can be stretched and folded onto the output complex,

Figure 8.7: Input and output complex for some domain of HDA D (a 3-cube C), drawn in Herlihy's way at the right hand side.

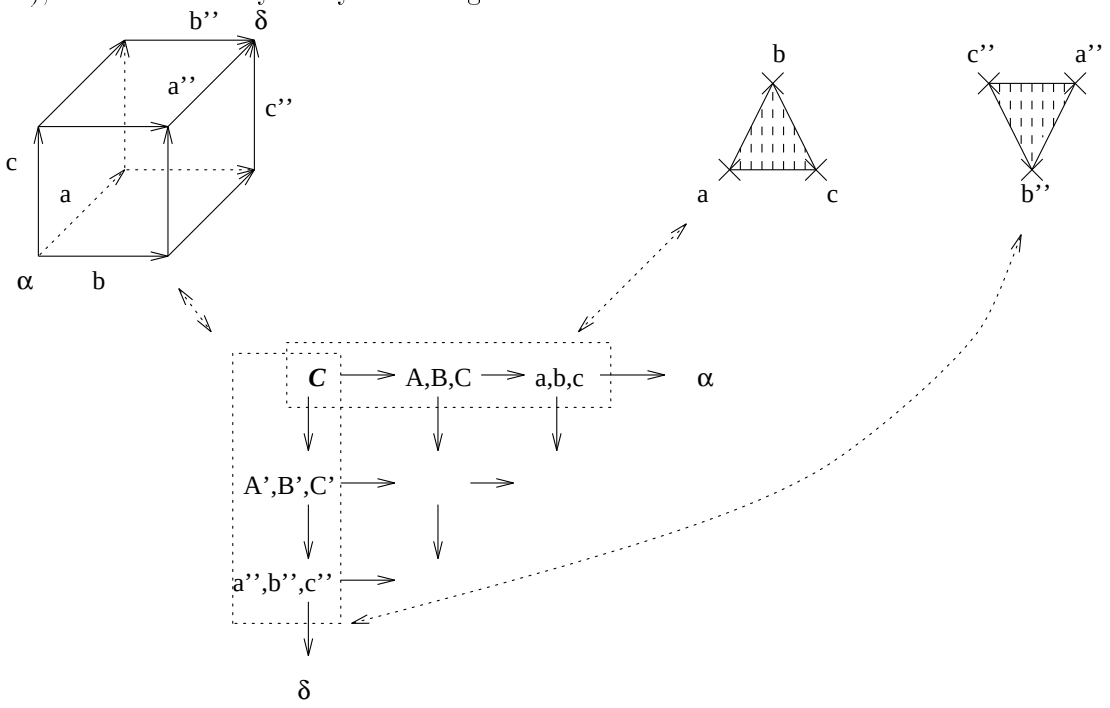


Figure 8.8: The effect of a failure of one process in (i)-a mutual exclusion, (ii)-a truly concurrent execution.

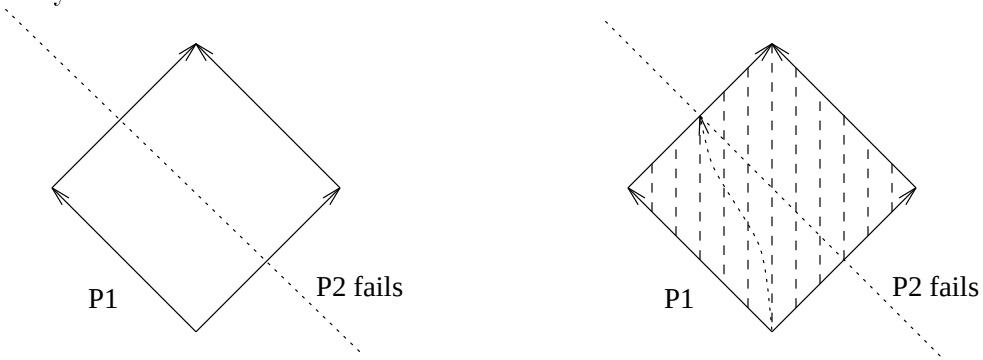
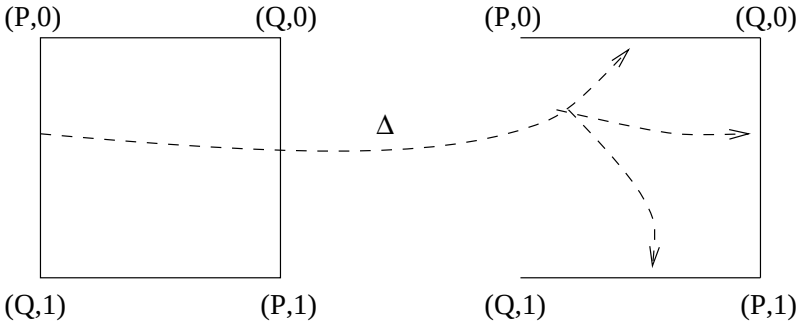


Figure 8.9: Binary pseudo-consensus input, output complexes and decision map.



Lemma 30 *Let D be a semi-regular HDA, I and O its input and output complexes respectively. Suppose D has only one initial state α, one final state β and there is exactly one class of paths modulo homotopy from α to β in D. Then $H_0(I) = H_0(O)$.*

PROOF. Let $p = (p_1, \ldots, p_k)$ and $p' = (p'_1, \ldots, p'_k)$ be two paths in D from α to β. By hypothesis they are homotopic. In particular there exist A and B with $p_1 - p'_1 = \partial_0(A)$ and $p_k - p'_k = \partial_1(B)$. $p_1, p'_1 \in I$ and $p_k, p'_k \in O$ are therefore connected in the input and output complexes respectively.

As there is exactly one initial state and one final state in D, given $u \in I$ and $v \in O$, we can find a path p from α to β with $p = (u, \ldots, v)$. This proves that $H_0(I) = H_0(O) = (\cdot)$ (i.e. is generated by a unique element). □

More generally, if

- D is deterministic in the sense that to each initial state α of D there corresponds a uniquely connected final state β,
- D is wait-free in the sense that there is exactly one class of paths modulo homotopy from α to β,

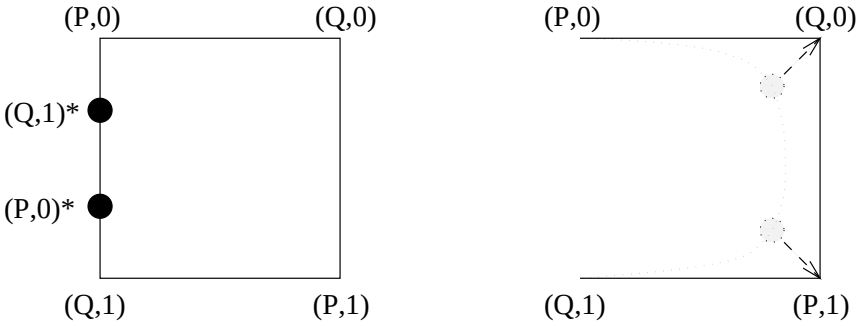
then, $H_0(I) = H_0(O)$ is isomorphic to the R-module of initial states of D.

In fact, we cannot say anything purely topological more than that on the input and output complexes. In particular, we cannot say anything about $H_1(I)$ and $H_1(O)$. Look at Figure 8.9) defining the binary pseudo-consensus tak. $H_1(I) = (\cdot)$ whereas $H_1(0) = 0$. But there is actually a way to construct the protocol from I and O. It can be shown [HS94] that there is a subdivision of I ("protocol complex") which is mapped on O by a simplicial map. The extra states added to the ones of I are used by the protocol for solving some internal choices (look at Figure 8.10). We will be more precise about that in Chapter 10.

8.3 Application: no algorithm for mutual exclusion

Now, we give an application of the HDA semantics of the CCS-like language we defined earlier and of the characterization of mutual exclusions. We actually

Figure 8.10: Binary pseudo-consensus protocol complex, its deformation onto the output complex and its implementation.



The pseudo-consensus between two processes P (variable x) and Q (variable y) on a shared-memory machine with atomic read and write is realized as follows,

$$\begin{array}{|lcl|}\hline P & := & \mathit{if}\ y = 1\ \mathit{then}\ x := 1; \\ Q & := & \mathit{if}\ x = 0\ \mathit{then}\ y := 0; \\ \hline\end{array}$$

prove a very similar result as the one of the previous section in that we prove that a wait-free portion of CCS cannot implement any mutual exclusion.

Lemma 31 *(i)* $H_n(Tot([\![\mathbf{nil}]\!])) = 0$ *for* $n \geq 0$,

(ii) $H_n(Tot([\![a]\!])) = 0$ *for* $n \geq 0$,

(iii) $H_n(Tot([\![p+q]\!])) = H_n(Tot([\![p]\!])) \oplus H_n(Tot([\![b]\!]))$ *for* $n \geq 1$,

(iv) $H_n(Tot([\![p;q]\!])) = H_n(Tot([\![p]\!])) \oplus H_0([\![p]\!], \partial_0) \otimes H_n(Tot([\![q]\!]))$ *for* $n \geq 1$,

(v) $H_n(Tot([\![p \mid q]\!])) = \sum_{i+h=n} H_i(Tot([\![p]\!])) \otimes H_h(Tot([\![q]\!]))$ *for* $n \geq 0$,

(vi) $H_n(Tot([\![\mathbf{rec}\ x.q(x)]\!])) = \varinjlim H_n(Tot([\![q^n(\mathbf{nil})]\!]))$.

Proof. (i) and (ii) are direct computations.

For (ii), we have $Tot([\![p+q]\!])_n = Tot([\![p]\!])_n \oplus Tot([\![q]\!])_n$, which entails the result by Lemma 9.

(iv) is an instance of Lemma 9 and Künneth formula.

(v) is Künneth formula.

Finally, (vi) is a consequence of Lemma 2. □

We can prove now that a synchronous subset of CCS cannot implement any mutual exclusion at all.

As a subset of CCS, we consider terms built with the usual operators but with no complementary action, making synchronizations impossible. Notice that in this case, $\mid$ and $\|$ coincide.

For convenience, we choose to call labels of actions (defining the labelling automaton of the semantic domain) by $(a_i)_{i\in\mathbb{N}}$.

We therefore only have to show that for all terms t, the (unlabeled) HDA $[\![t]\!]$ contains no mutual exclusion. This is done by induction on the syntax, using the computation of the homology we have carried out in Section 6.4.

(**nil**): $H_n(Tot([\![\mathbf{nil}]\!])) = 0$ for $n > 0$ thus no mutual exclusion.

(a_i): Same as above.

$p + q$: for $i \geq 1$, we know that $H_i(Tot([\![p+q]\!])) = H_i(Tot([\![p]\!])) \oplus H_i(Tot([\![q]\!]))$. But by induction, there is no mutual exclusion in q nor in p, so there cannot be any in $p + q$.

$p;q$: we know that $H_i(Tot([\![p;q]\!])) = H_i(Tot([\![p]\!])) \oplus H_0([\![p]\!], \partial_0) \otimes H_i(Tot([\![q]\!]))$. Same conclusion as above.

$p \mid q$: for $i \geq 0$, we know that

$$H_n(Tot([\![p \mid q]\!])) = \sum_{i+h=n} H_i(Tot([\![p]\!])) \otimes H_h(Tot([\![q]\!]))$$

therefore there cannot be any mutual exclusion in $p \mid q$ if there was not in p and q.

rec $x.q(x)$: The homology modules are the limit of the homology modules of the unfolding of **rec** $x.q(x)$. By induction hypothesis, there is no mutual exlusion in the unfolding. This entails that there is no mutual exclusion in the direct limit.

8.4 Some properties of the interpretations of transition systems

We know that general HDA are an abstraction of semi-regular HDA. The same holds between general HDA whose elements are of dimension zero or one and semi-regular HDA of dimension zero and one. T_1 and $\mathcal{G}_1$ lift to general HDA. In these categories, we can speak internally of some geometric properties. This provides us with a means to describe the functor $\mathcal{G}_1$ in geometric terms[1] and gives another proof of Lemma 4 in a more general setting.

Proposition 15 *Let X be a general HDA whose elements are of dimension zero and one. $\mathcal{G}_1(X)$ is the smallest HDA[2] Y containing X such that*

$$(i) : \forall k \geq 1, \forall (v,w) \neq (0,0) \in H_k(Y, \partial_0) \times H_k(Y, \partial_1), \partial_1^{*0}(v) \neq \partial_0^{*1}(w)$$

[1] And answers a question raised by Alan Mycroft in WSA'93.

[2] Here we restrict to HDA M such that $\forall x \in M$, $x = 0$ if and only if $\partial_0(x) = \partial_1(x) = 0$. This means that we do not consider HDA with elements being at the same time initial and final deadlocks (actually, these transitions can never be fired).

PROOF. The fact that $\mathcal{G}_1(X)$ contains X is obvious.

Now, suppose that (i) does not hold. Then there exists u, $v \in H_k(\mathcal{G}_1(X), \partial_0)$, $w \in H_k(\mathcal{G}_1(X), \partial_1)$, $(v, w) \neq (0, 0)$ with $u = \partial_1^{*0}(v) = \partial_0^{*1}(w)$. Considering particular representatives of u, v and w (still written u, v and w), we have u such that $u = \partial_1(v) = \partial_0(w)$. Let A be a transition of dimension $k+1$ defined by $\partial_0(A) = v$, $\partial_1(A) = -w$. It is well-defined since

- $\partial_0\partial_0(A) = \partial_0(v) = 0$ because v is by definition a cycle for ∂_0,
- $\partial_1\partial_1(A) = \partial_1(w) = 0$ because w is by definition a cycle for ∂_1,
- $\partial_0\partial_1(A) = -\partial_0(w) = -u = -\partial_1(v) = -\partial_1\partial_0(A)$.

Consider the HDA $Y = \mathcal{G}_1(X) + (A)$. The counit arrow $\epsilon_X : T_1(\mathcal{G}_1(X)) \to X$ corresponds in a unique manner via the adjunction to the identity arrow $Id : \mathcal{G}_1(X) \to \mathcal{G}_1(X)$. As $T_1(Y) = T_1(\mathcal{G}_1(X))$, $\epsilon_X : T_1(Y) \to X$ corresponds in a unique manner via the adjunction to $h : Y \to \mathcal{G}_1(X)$. Therefore $h_{|\mathcal{G}_1(X)} = Id$. Let $i : \mathcal{G}_1(X) \to Y$ be the inclusion arrow from $\mathcal{G}_1(X)$ to Y. Then $h \circ i = Id$ and $h^{*0} \circ i^{*0} = Id$, $h^{*1} \circ i^{*1} = Id$ on the homology groups. But necessarily, $i^{*0}(v) = 0$, $i^{*1}(w) = 0$. One of v or w has to be different from zero, say v. Then $h^{*0}(0) = v \neq 0$. This is impossible. Therefore, it is necessary to have property (i) for $\mathcal{G}_1(X)$.

Suppose now that there exists an HDA Y with $X \subseteq Y \subseteq \mathcal{G}_1(X)$ with $Y \neq \mathcal{G}_1(X)$, satisfying property (i). Let $u \in \mathcal{G}_1(X)$, $u \notin Y$. It is necessary to have $dim\ u \geq 2$ since otherwise u would be already in X and then in Y. Let $Z = \mathcal{G}_1(X)/(u)$. Z is a HDA containing $\partial_0(u)$ and $\partial_1(u)$. But $(v, w) = (\partial_1(u), -\partial_0(u)) \in H_k(Z, \partial_0) \times H_k(Z, \partial_1)$, $\partial_1^{*0}(v) = \partial_0^{*1}(w)$. One of v, w is different from 0, say v. The injection $i : Y \hookrightarrow Z$ induces $i^{*0} : H_k(Y, \partial_0) \to H_k(Z, \partial_0)$ and $i^{*1} : H_k(Y, \partial_1) \to H_k(Z, \partial_1)$. We have necessarily $i(0) = v$ which contradicts the fact that i is the identity. This proves the result. □

This property can be generalized to the pair of adjoint functors $(T_n, \mathcal{G}_n)$ (for all $n \geq 1$). $\mathcal{G}_n(X)$ is the smallest HDA Y containing X such that

$$\forall k \geq n, \forall (v, w) \neq (0, 0) \in H_k(Y, \partial_0) \times H_k(Y, \partial_1), \partial_1^{*0}(v) \neq \partial_0^{*1}(w)$$

Notice that for acyclic HDA, the R-submodule U of $H_k(Y, \partial_0) \times H_k(Y, \partial_1)$ composed of elements (v, w) such that $\partial_1^{*0}(v) = \partial_0^{*1}(w)$ is isomorphic to the submodule V of $H_k(Tot(Y))$ generated by cycles of length 2 of Y.

PROOF. Let $y \in U$. We can choose representants v and w such that,

- $y = ([v], [w])$,
- $\partial_0(v) = 0$,
- $\partial_1(w) = 0$,
- $\partial_1(v) = \partial_0(w)$

Then $(\partial_1 - \partial_0)(v + w) = \partial_1(v) - \partial_0(w) = 0$. This defines a map $i : U \to H_k(Tot(y))$ which is actually a homomorphim of R-modules.

As Y is acyclic, $H_k(Y, \partial_0)$ and $H_k(Y, \partial_1)$ are generated by elements of the form $[x]$, with $x \in M_{k-i,i}$ for some i. This implies that $H_k(Y, \partial_0) \times H_k(Y, \partial_1)$ is generated by elements of the form $([x], [y])$, $x \in M_{k-i,i}$, $y \in M_{k-j,j}$ for some i, j. $\partial_1^{*0}([x]) = \partial_0^{*1}([y])$ imposes $i = j + 1$. This shows that a generating set for U is the $([x], [y])$, $x \in M_{k-j-1,j+1}$, $y \in M_{k-j,j}$ for some j. Its image by i consists of a generating set for V which only contains cycles of length 2.

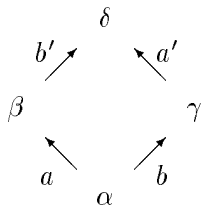
Inversely, let $[x] \in V$. As x is of length 2 we can decompose x as $x = x_1 + x_2$ with $x_1 \in M_{k-i,i}$ and $x_2 \in M_{k-i+1,i-1}$. This decomposition is unique as Y is acyclic. Now, $[x] \in H_k(Tot(Y))$ therefore,

- $\partial_0(x_1) = 0$,
- $\partial_1(x_1) = \partial_0(x_2)$,
- $\partial_1(x_1) = 0$

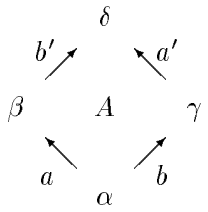
Therefore $([x_1], [x_2]) \in U$. This defines an homomorphism of R-modules $j : V \to U$ inverse of i. □

This actually proves Lemma 4 as follows. Let $f : \dot{D}_{[n]} \to Y$ be a monomorphism from the boundary of an n-cube ($n \geq 1$) to a semi-regular HDA Y. $f(\dot{D}_{[n]})$ is a cycle of length 2 (this is an abuse of notation, since we should see that in $\underline{Y}$). The previous result shows that it should be filled in so there is a $(n+1)$-transition A such that $(\partial_0 - \partial_1)(A) = f(\dot{D}_{[n]})$. It is easy to see that f can be extended to a morphism of general HDA to $D_{[n]}$ by putting $f(Id_{[n]}) = A$.

Example 41 *The element X of Υ_{01} realized geometrically as,*



has a mutual exclusion of dimension one described by $(b-a, a'-b') \in H_1(X, \partial_0) \times H_1(X, \partial_1)$ since $\partial_0(a' - b') = \gamma - \beta = \partial_1(b - a)$. Looking at the proof of the fact that $\mathcal{G}_1$ verifies property (i) as an algorithm for computing $\mathcal{G}_1$ as the least fixed point of the operation "fill in the k-mutual-exclusions by k-transitions"[3], $\mathcal{G}_1(X)$ is easily seen to be equal to (in one iteration),



[3]This is the classical way we can compute homology of CW-complexes by adding cells to "kill" the homotopy.

This exemplifies the fact that $\mathcal{G}_1(X)$ is the automaton in which all actions are scheduled on as many as processors as it can be.

Summary We have given the first four applications of the theory of Chapter 7. We have first recast Groves' construction for computing the homology of a monoid presented by a term rewriting system into the HDA framework, showing that this construction is about the serializability of a finite number of non-conflicting reduction rules put in parallel. This enabled us to prove (again!) a particular case of Squier's theorem: the class of monoids presented by a finite strongly confluent rewriting system is strictly included in the class of monoids with decidable word problem.

We have then recast Herlihy's proof that there was no wait-free protocol for consensus in a semantic framework based on the homotopy theory of HDA. This differs from the approach taken by Herlihy et al. in that we consider also the geometry of all allowed executions in the distributed machines we consider.

Then we proved that a synchronous subset of CCS cannot implement any mutual exclusion. At first, the aim was to prove that in a concurrent machine, there needed to be an internal mechanism (semaphore etc.) in order to make mutual exclusion expressible. We have not yet completely fulfilled this goal.

Our last application was to prove the m-connectedness result of Chapter 3 about the right-adjoint of the truncation functor. We used a characterization of some mutual exclusions in homological terms to show that all maps from the boundary of the standard n-cube could be extended to maps on the standard n-cube.

Part IV

Analysis of Programs

Chapter 9

Analysis of programs

9.1 Abstract interpretation

9.1.1 Introduction

Having defined denotational semantics of languages using HDA, we would like to formalize the abstraction process (in the style of [CC77]) enabling us to compute the properties which are of interest for us. In particular, we would like to be able to approximate (by folding the execution traces etc.) HDA to make the computation of the geometric properties of Chapters 6, 7 tractable. But it is not natural to define an order on HDA to fit in the usual abstract interpretation framework. Surely, a natural order would be the "inclusion" ordering, or "subobject" ordering. This can be defined in the same algebraic style as we have used up to now, since subobjects of X are monomorphisms into X. This calls for generalization. From now on, we consider any arrow into X to be an "element" of X.

9.1.2 Definitions

Let D_c be a domain of HDA in which we have given the semantics of some language $\mathcal{L}$. D_c is called the concrete domain. Let D_a be another domain of HDA, called the abstract domain. Consider $\mathcal{C}$ and $\mathcal{A}$ to be two subcategories of Υ/D_c and Υ/D_a respectively. Now, an abstract interpretation between D_c and D_a is a pair of adjoint functors (α, γ) between $\mathcal{C}$ and $\mathcal{A}$,

$$\mathcal{C} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{A}$$

That is α and γ are functors such that there exists natural transformations $\epsilon : \alpha \circ \gamma \to Id$ and $\eta : Id \to \gamma \circ \alpha$ such that the following composite arrows are the identities [ML71],

$$\gamma \xrightarrow{\eta\gamma} \gamma\alpha\gamma \xrightarrow{\gamma\epsilon} \gamma$$

$$\alpha \xrightarrow{\alpha\eta} \alpha\gamma\alpha \xrightarrow{\epsilon\alpha} \alpha$$

A more general formulation would be to have a category of observations (as in [AM93]) O and a functor $F : \Upsilon \to O$ to observe HDA. Then an abstract interpretation is given by an abstract domain $D_a \in O$ and a pair of adjoint functors between $\mathcal{C}$ and $\mathcal{D}$, subcategories of Υ/D_c and of the comma category of arrows $F(x) \to D_a$ of O respectively,

$$\mathcal{C} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{A}$$

For the time being, we restrict to the first case ($F = Id$, $O = \Upsilon$). We will see that it is enough for looking at some interesting dynamic properties of HDA.

The standard semantics is given in a subcategory of the category of subobjects of D_c. The category Sub of subobjects of D_c is a subcategory of the slice category Υ/D_c, and as such is a good candidate for being used as domain (or codomain) of an abstraction functor. We recall (see Section 5.2) that as Υ is complete and co-complete, Sub is actually a complete lattice with the following operations,

- Intersection $X \wedge Y$ of subobjects X and Y of A is the pullback of the corresponding morphisms,

$$\begin{array}{ccc} X \wedge Y & \xrightarrow{i} & X \\ {\scriptstyle j}\downarrow & & \downarrow{\scriptstyle x} \\ Y & \xrightarrow{y} & A \end{array}$$

- Union $X \vee Y$ is the pushout of $X \wedge Y \xrightarrow{i} X$ and $X \wedge Y \xrightarrow{j} Y$
- The order to which these lattice operations correspond is $(X \xrightarrow{x} A) \leq (Y \xrightarrow{y} A)$ iff there is a monomorphism $f : X \to Y$ such that $y \circ f = x$.

Then, if C and A are the subcategories of subobjects of some HDA then a pair of adjoint functors between C and A is actually a Galois connection.

But this will not be the case in general, so we will find it convenient to use Freyd's special adjoint functors' theorem ([ML71] or [FS90]) in the following, to prove the existence of a right-adjoint to a given functor $\alpha : \Upsilon/D_c \to \Upsilon/D_a$. First, we have to show that certain pre-conditions on Υ/D_c and Υ/D_a are verified.

(i) Υ/A is small co-complete.

(ii) Υ/A and Υ/B have small hom-sets.

(iii) Υ/A is well-co-powered.

(iv) Υ/A has a small generating set.

PROOF. Υ is small co-complete, thus (i).

(ii) is entailed by the fact that Υ has small Hom-sets: a morphism between A and B is in particular a set-theoretic function between the sets underlying A and B.

For (iii) we have to prove that epimorphisms in Υ/A from a given object P, modulo composition with isomorphisms, is a set. A sufficient condition is to have the same property holding in Υ. In Υ, epimorphisms from P and monomorphisms into P, modulo composition with isomorphisms, are in bijection (consider the kernel of the epimorphism in one way, and the projection onto the image of the monomorphism in the other way). But subobjects of P form a set since P is a set.

We now build a small generating set S for Υ/A.

Let $T^{p,q}$, for $p, q \in \mathbb{Z}$, be the free HDA generated by one transition $t_{p,q} \in T^{p,q}_{p,q}$. We set S to be the set of all morphisms from a $T^{p,q}$ to A. Let now $p : P \to A$ and $q : Q \to A$ be two elements of Υ/A and $h, h' : p \to q$ be two morphisms in Υ/A such that $h \neq h'$. Therefore, there exists $x \in P$ such that $h(x) \neq h'(x)$.

We have then $p, q \in \mathbb{Z}$ with $x \in P_{p,q}$ and by the freeness of $T^{p,q}$, there is a morphism f from $T^{p,q}$ to P with $f(t_{p,q}) = x$. Thus, $h \circ f \neq h' \circ f$. This proves (iv). □

We know now (see [ML71]) that α has a right-adjoint if and only if it is co-continuous, i.e. it commutes with all colimits.

9.1.3 An abstraction: denotational semantics

Let D_c be a concrete domain on which we give semantics to some language $\mathcal{L}$. Let $D_a = H_0(D_c, \partial_1) \otimes H_0(D_c, \partial_0)$ the abstract domain. We consider the following "abstraction" functor Ω:

$$\Omega : \Upsilon/D_c \to \Upsilon/D_a$$

with

$$\Omega(x : X \to D_c) = x^{*_1} \otimes x^{*_0} : H_0(X, \partial_1) \otimes H_0(X, \partial_0) \to D_a$$

This functor maps an element of D_c to the pair (initial states, final states).

We know that all the homology functors commute with direct limits, and that the tensor product (as it has a right-adjoint) commutes with all colimits. To verify that Ω is co-continuous, and then that Ω has a right-adjoint by the application of the result of last section, we only need to prove that $H_0(\cdot, \partial_i)$ commutes with all finite colimits, i.e. commutes with direct sums and with co-equalizers. The commutation with direct sums is well-known [Mas78].

Suppose now that we restrict to automata M such that

$$\forall s \in M_0,\ \partial_0(s) = \partial_1(s) = 0$$

that is, "no state contains proper events".

Let r be the coequaliser of $f, g : a \to b$,

$$a \underset{g}{\overset{f}{\rightrightarrows}} b \xrightarrow{\quad r \quad} c$$

then we know that $c \cong \frac{b}{(f-g)(a)}$ and r is the canonical projection from b to c. The commutation of $H_0(\cdot, \partial_i)$ is thus expressed as

$$H_0\left(\frac{b}{(f-g)(a)}, \partial_i\right) \cong \frac{H_0(b, \partial_i)}{(f^{*_i} - g^{*_i})(H_0(a, \partial_i))}$$

With the hypothesis that no state of a and b contains proper events, it is equivalent to

$$\partial_i\left(\frac{b_1}{(f-g)(a_1)}\right) \cong \frac{\partial_i(b_1)}{(f-g)(\partial_i(a_1))}$$

This is straightforward since $\partial_i \circ (f-g) = (f-g) \circ \partial_i$ (f and g are morphisms).

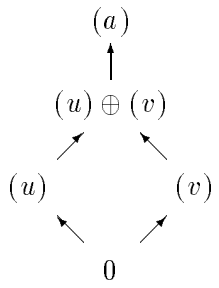
Example 42 (1) *Let D_c be the following domain,*

$$u \xrightarrow{\quad a \quad} v$$

Then D_a is,

$$u \otimes v$$

The lattice of subobjects of D_c is,



Now, the image of this lattice by Ω is

- $\Omega(0) = 0 : 0 \to D_a$,
- $\Omega((u)) = 0 : (u \otimes u) \to D_a$,
- $\Omega((v)) = 0 : (v \otimes v) \to D_a$,
- $\Omega((u)) = Id_{(u \otimes v)} : (u \otimes u) \oplus (u \otimes v) \oplus (v \otimes u) \oplus (v \otimes v) \to D_a$,
- $\Omega((a)) = Id : D_a \to D_a$.

(2) *We give here a case which is not generally captured by frameworks based on lattices. Let D_c be the domain,*

$$\alpha_1 \xrightarrow{\quad a_1 \quad} \alpha_2 \xrightarrow{\quad a_2 \quad} \ldots$$

and consider the abstraction functor "final state", $F = H_0(\cdot, \partial_0)$. The abstract domain D_a is 0.

Consider now the three subobjects X, Y and Z of D_c,

$$X = 0$$
$$Y = \alpha_1 \xrightarrow{a_1} \alpha_2$$
$$Z = D_c$$

Then we have inclusion morphisms,

$$X \overset{i}{\hookrightarrow} Y \overset{j}{\hookrightarrow} Z$$

and in the abstract domain,

$$F(X) = 0 \overset{F(i)}{\to} F(Y) = \alpha_2 \overset{F(j)}{\to} 0$$

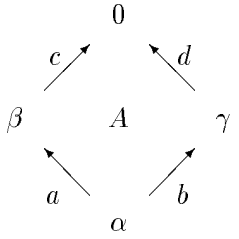
Therefore, the image of the lattice of subobjects of D_c by F contains the graph,

$$0 \underset{F(j)}{\overset{F(i)}{\rightleftarrows}} \alpha_2$$

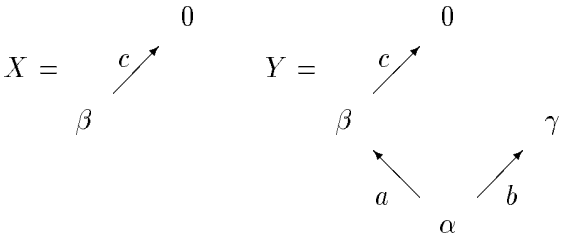
Thus, this situation cannot be described by a Galois connection between the lattice of subobjects of D_c and another lattice.

Notice that Ω and F commute with direct limits. In particular, the end state of an infinite trace is the direct limit of the end states of all its finite approximations, since the inclusion from one finite approximation to the next one induces a null map by F (in homology).

An other example, with the same abstraction functor is, defining the abstract domain D_c as,



and considering the three subobjects,



$Z = D_c$

Then,

$$X \overset{i}{\hookrightarrow} Y \overset{j}{\hookrightarrow} Z$$

and the image by F is,

$$0 \underset{F(j)}{\overset{F(i)}{\rightleftarrows}} \gamma$$

The interpretation of such a graph structure is that, we may execute (Z) more transitions but get less observable results (than Y), due to a deadlocking behaviour. We may as well get more results (Y) if we execute more transitions (than X).

9.1.4 A technical abstraction: the image functor

We define here a simple tool used for being able to represent objects in the slice category Υ/D as subobjects of D. Let Sub be the category of sub-HDA of D.

Define an abstraction functor $\mathcal{I}m$ from Υ/D to Sub by,

$$\mathcal{I}m(x : X \to D) = Id : x(X) \hookrightarrow D$$

$$\mathcal{I}m\left(\begin{array}{ccc} X & \xrightarrow{x} & D \\ {\scriptstyle f}\downarrow & \nearrow_{y} & \\ Y & & \end{array}\right) = Id : x(X) \to y(Y)$$

It maps every arrow x to the subobject of D representing its image.

The concretization functor, its right-adjoint, is $\mathcal{J}$ given by,

$$\mathcal{J}(y : Y \to D) = y \circ p_1 : Y \times D \to D$$

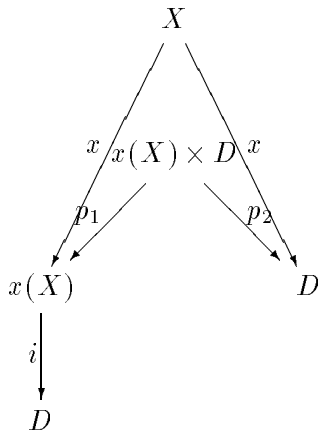
where p_1 is the canonical projection for the cartesian product, $p_1 : Y \times D \to Y$.

PROOF. We have

$$\begin{aligned} \mathcal{I}m \circ \mathcal{J}(y : Y \to D) &= \mathcal{I}m(y \circ p_1 : Y \times D \to D) \\ &= Id : y(Y) \to D \end{aligned}$$

But if $y : Y \to D$ belongs to the category Sub, then $y = Id$ and $\mathcal{I}m \circ \mathcal{J} \cong Id$.

Now, consider the following (non-commutative) diagram,



By definition of the cartesian product $x(X) \times D$, there exists a unique h_X such that $x = p_1 \circ h_x = p_2 \circ h_x$. Therefore h_x is a morphism between $x : X \to D$ and $\mathcal{J} \circ \mathcal{I}m(x) = i \circ p_1 : x(X) \to D$ in the slice category Υ/D. It is obviously natural in X and defines what is to be the counit of the adjunction.

Finally, we have to verify that, first, for all $x : X \to D$ the composition,

$$\mathcal{I}m(x) \xrightarrow{\mathcal{I}m(h_{\mathcal{I}m(x)})} \mathcal{I}m \circ \mathcal{J} \circ \mathcal{I}m(x) \xrightarrow{Id} \mathcal{I}m(x)$$

is the identity. This is obvious since $\mathcal{I}m(x) = Id$.

Then, we have to prove that for all $y : Y \to D$ in Sub, the composition,

$$\mathcal{J}(y) \xrightarrow{h_{\mathcal{J}(y)}} \mathcal{J} \circ \mathcal{I}m \circ \mathcal{J}(y) \xrightarrow{\mathcal{J}(Id)} \mathcal{J}(y)$$

is the identity. But $\mathcal{J}(Id) \circ h_{\mathcal{J}(y)} = y = Id$ since $y \in Sub$. □

9.1.5 An abstraction: truncation

We consider here a way to reduce the number of transitions of a HDA by reducing the amount of asynchrony we are allowed to observe.

Let n be an integer and $D_{a,n}$ the abstract domain defined by $(D_{a,n})_{p,q} = (D_c)_{p,q}$ if $p + q \leq n$ and $(D_{a,n})_{p,q} = 0$ if $p + q > n$. It is the domain of processes of dimension at most n.

Let now $x : X \to D_c$ be an element of Υ/D_c. Let X' be the sub-HDA of X consisting of transitions up to dimension n ("truncation" of X of order n). We define[1] $T_n(x)$ to be the induced morphism from X' to D_a. For f a morphism between $x : X \to D_c$ and $y : Y \to D_c$, we define $T_n(f)$ to be the induced morphism between the truncations of X and Y of order n. This defines the abstraction functor.

Take A in $\Upsilon/D_{a,n}$. Let $Y(A)$ be the diagram in Υ/D_c, whose objects are all elements x of Υ/D_c such that $T_n(x)$ is isomorphic to A, and whose arrows are

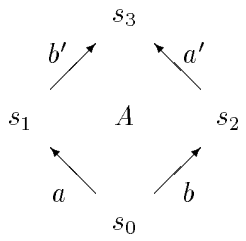
[1] By slight abuse of notation

all possible morphisms in Υ/D_c between these objects. We define a functor $\mathcal{G}_n : \Upsilon/D_{a,n} \longrightarrow \Upsilon/D_c$ to be $\mathcal{G}_n = \varinjlim Y(\cdot)$. Then,

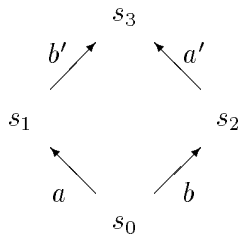
Lemma 32 *$(T_n, \mathcal{G}_n)$ is a pair of adjoint functors.*

This pair of adjoint functors induces a Galois connection between the lattices of sub-HDA of D_c and $D_{a,n}$ (viewed as a sub-category of Υ/D_c and $\Upsilon/D_{a,n}$ respectively).

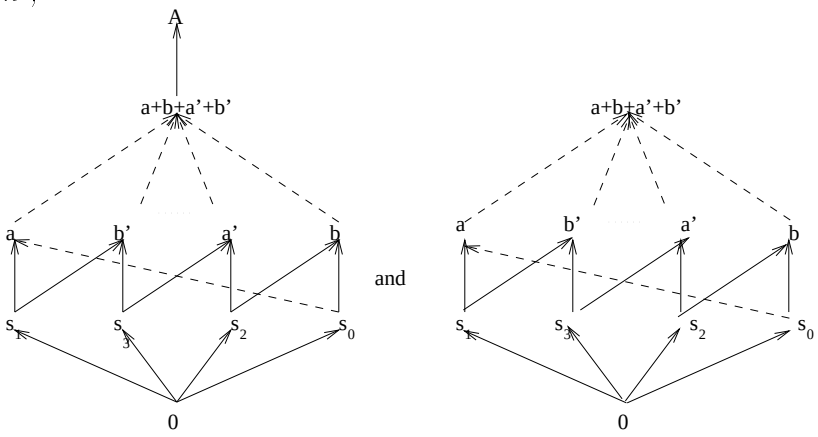
Example 43 *Let D_c be the HDA,*



and set n to 1. Then D_a is,



If we look at the lattices of subobjects of D_c and D_a we have a Galois connection between[2]*,*



9.1.6 An abstraction: folding

In this section, we define a way to reduce the number of states, transitions, etc. in a HDA by folding together parts of the execution traces.

[2] We only picture here the lattice of connected subobjects of D_c and D_a.

Suppose that we are given an epimorphism p from D_c to a domain D_a. p can be interpreted (similar to the labelling morphism) as a folding morphism.

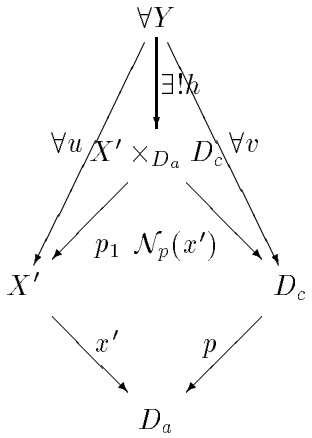
This can be lifted to the slice categories Υ/D_c and Υ/D_a as an abstraction functor,

$$\mathcal{M}_p(x : X \to D_c) = p \circ x : X \to D_a$$

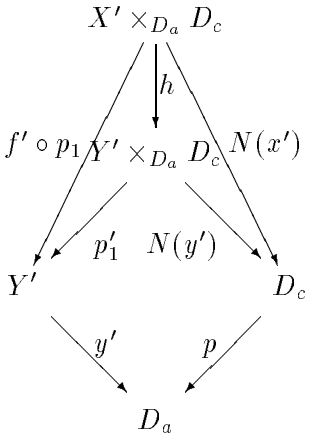
$$\mathcal{M}_p(f : (x : X \to D_c) \to (y : Y \to D_c)) = f : (\mathcal{M}_p(x)) \to (\mathcal{M}_p(y))$$

Let now $\mathcal{N}_p$ be the functor from Υ/D_a to Υ/D_c defined by:

- for $x' : X' \to D_a$, $\mathcal{N}_p(x')$ is the pullback of x' along p, i.e. is the "greatest" morphism $\mathcal{N}_p(x') : X' \times_{D_a} D_c \to D_c$ such that $p \circ N(x') = x' \circ p_1$ where $p_1 : X' \times_{D_a} D_c \to X'$ is given by the pullback diagram (see [ML71]),



- and for $f' : (x' : X' \to D_a) \longrightarrow (y' : Y' \to D_a)$, $\mathcal{N}_p(f') : X' \times_{D_a} D_c \to Y' \times_{D_a} D_c$ is the unique morphism h in the following pullback diagram:

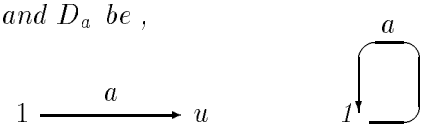


Then,

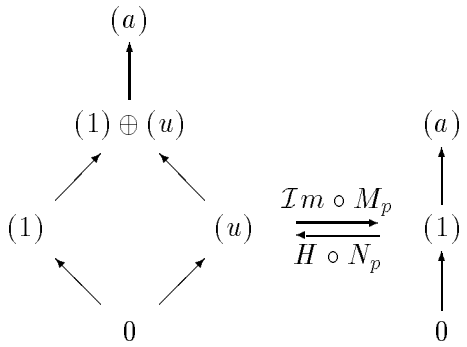
Lemma 33 *($\mathcal{M}_p$,$\mathcal{N}_p$) is a pair of adjoint functors.*

PROOF. See for instance [MM92]. □

Example 44 *Let D_c and D_a be ,*



and $p : D_c \to D_a$ be the epimorphism defined by $p(1) = p(u) = 1$ and $p(a) = a$, then $(\mathcal{I}m \circ \mathcal{M}_p, \mathcal{J} \circ \mathcal{N}_p)$ is a Galois connection between the lattices of subobjects of D_c and D_a,



What are the abstract operators now, i.e. the abstract counterparts of $+$, $\otimes$ and $\varinjlim$? This will not be possible to compute them in general, and we may have to use safe approximations of them.

For H any endofunctor on Υ/D_c, we say that G, endofunctor on Υ/D_a, is a *safe approximation* of H if and only if there exists a natural transformation from $\alpha H \gamma$ to G. Notice that it reduces to the usual definition when (α, γ) is a Galois connection. The fact that (α, γ) is a pair of adjoint functors implies that colimits in Υ/D_a are safe approximations of colimits in Υ/D_c. For instance, we can take as abstraction of $+$ and $\varinjlim$, $+$ and $\varinjlim$ respectively. This does not hold for $\otimes$ and its abstract version $\otimes_a$. But we can prove the following:

- For the adjunction $(T_n, \mathcal{G}_n)$, there is an "expansion law", $x \otimes_a y = x \otimes T_0(y) + T_0(x) \otimes y + \sum_{0<k<n} T_k(x) \otimes T_{n-k}(y)$.
- For the adjunction $(\mathcal{M}_p, \mathcal{N}_p)$, if p is a multiplicative morphism then $x \otimes_a y = x \otimes y$.

9.1.7 Schedulers as an abstract interpretation

Let Sc be the category whose objects are equivalence classes of elements of $R - Mod/\Pi_n(D)$ modulo isomorphisms.

Define now

$$\alpha_n : Sub \to Sc$$

by

$$\alpha_n(i : M \hookrightarrow D_c) = (\Pi_n(i) : \Pi_n(M) \to \Pi_n(D))$$

$\alpha_n(x)$ provides us with all n-schedulers of automaton x: $\alpha_n(x)$ basically returns the equivalence class of all retracts of dimension n of x (see Figure 9.1).

An analogue to Van Kampen's theorem holds (Section 7.7.2). Recall that, for all X_1 and X_2, $X_1 \cup X_2$ and $X_1 \cap X_2$ are well defined by the cocartesian square

$$\begin{array}{ccc} X_1 \cap X_2 & \xrightarrow{j_1} & X_1 \\ \downarrow{\scriptstyle j_2} & & \downarrow{\scriptstyle i_1} \\ X_2 & \xrightarrow{i_2} & X_1 \cup X_2 \end{array}$$

Then Van Kampen's theorem asserts that the following diagram in $R-Mod$ is cocartesian as well,

$$\begin{array}{ccc} \Pi_n(X_1 \cap X_2) & \xrightarrow{\Pi_n(j_1)} & \Pi_n(X_1) \\ \downarrow{\scriptstyle \Pi_n(j_2)} & & \downarrow{\scriptstyle \Pi_n(i_1)} \\ \Pi_n(X_2) & \xrightarrow{\Pi_n(i_2)} & \Pi_n(X_1 \cup X_2) \end{array}$$

Therefore α_n commutes with (binary) least upper bounds. In case $\Pi_n(D_c)$ is of **finite type** (implied by D_c finite for instance), this proves the existence of a right-adjoint $\gamma_n : R-Mod/\Pi_n(D_c) \to Sub$ to α_n by Freyd's special adjoint functors theorem [ML71]. (α_n, γ_n) is a pair of adjoint functors or a **Galois connection**[3]. We strongly believe that this generalizes to modules $\Pi_n(D_c)$ of infinite type but we do not have yet a proof of that.

Figure 9.1 should then be understood as follows. When we have only one processor, A, B and D have exactly the same schedulers, i.e. they have essentially one and only 1-path (D retracts to any of A or B). This means that the best approximation of A and B (by $\gamma_1 \circ \alpha_1$) is D. Only $C' = \alpha_1(C)$ (two paths, i.e. two generators) is different from (non-isomorphic to) $A' = \alpha_1(A)$, $B' = \alpha_1(B)$ and $D' = \alpha_1(D)$ (one path, i.e. one generator). The arrow in Sc going up from A' to C' is the image by α_1 of the inclusion morphism from A to C. Similarly, the arrow going downwards from C' to D' comes from the inclusion morphism of C into D and whose action is to project the hole of C onto 0 (the hole is filled in D).

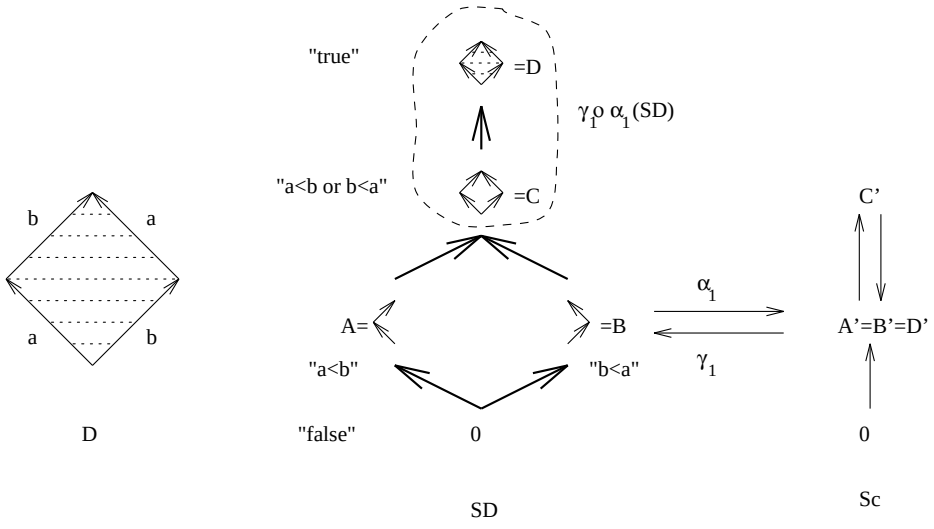
9.1.8 Dependence orderings and abstract interpretation

We wish here to give constraints on scheduling and schedulers the same presentation via **dependence orderings** of the form "$a < b$" meaning action b must be scheduled just before action a. The logical language $\mathcal{L}$ will be constructed out of predicates "$a < b$", variables ranging over actions, quantifiers and connectives from classical logic. We authorize infinite formulas, so that quantifiers are just syntactic sugar. This is the same intuition (but in a more general context) as dependence orderings in Mazurkiewitz trace theory [Maz88] or more recently in concurrent automata [BDK94]. Notice that it is enough to specify the scheduling properties we had in mind up to now,

[3] We do not ask to have a poset as an abstract domain, it may be a general category (or a preorder as in [CC94]). Notice that (as pictured in Figure 9.1) $\gamma_n \circ \alpha_n$ is an upper closure operator on Sub, [CC77].

Figure 9.1: A domain of automata D, a subposet SD of Sub and its abstraction to Sc

.



- in Example 31 it suffices to give a constraint on add.s_P^i (the i-th add.s action executed on the same processor P; we forget the i index in the sequel) of the form $\exists x, y \neq \text{add.s}_P, \text{add.s}_P < x < y < \text{add.s}_P$ which is just syntactic sugar for $\vee_{x,y \neq \text{add.s}_P}((\text{add.s}_P < x) \wedge (x < y) \wedge (y < \text{add.s}_P))$.
- the two-phase protocol on two processes P and Q consists in the following. Let A_P and A_Q be respectively the items that P, respectively Q, access via actions P_a and Q_b respectively ($a \in A_P$, $b \in A_Q$). Then the two-phase protocol is a scheduling constraint of the form

$$(\wedge_{a,c \in A_P}((\text{lock}a < P_c) \wedge (P_c < \text{unlock}a))) \wedge$$

$$\left(\wedge_{b,d \in A_Q}((\text{lock}b < Q_d) \wedge (Q_d < \text{unlock}b))\right)$$

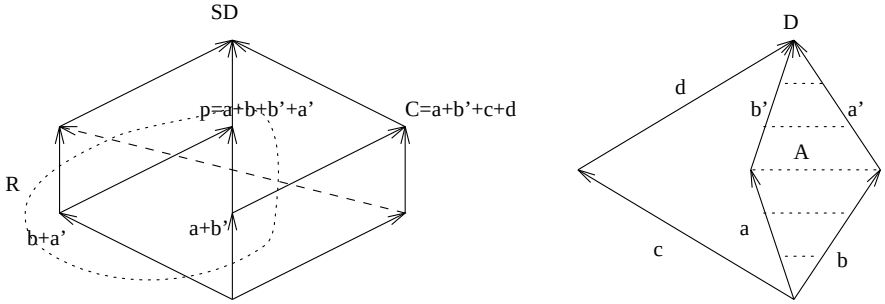
- mutual exclusion properties between two actions a and b can be formalized by the formula $(a < b') \vee (b < a')$.

More formally, if D is a domain of semi-regular HDA, let $\mathcal{L}$ be the following (infinitary) logic where terms are formed by the following constructions,

- for all $n > 0$ and for all actions or variables a, b in $(D)_n$ $a < b$ is in $\mathcal{L}$,
- for all formulas p and q in $\mathcal{L}$, $p \vee q$, $p \wedge q$, $\forall x.p(x)$ and $\exists x.p(x)$ are formulas in $\mathcal{L}$.

The syntacticly different predicates on actions are declared non comparable by $\leq$. Then $\leq$ is logical implication. $(\mathcal{L}, \leq)$ is a complete lattice with $\vee$ as the least upper bound operation and the greatest lower bound being $\wedge$. It is straightforward now to show that $(\mathcal{L}, \leq)$ and Sub are **equivalent** lattices,

Figure 9.2: Sub (simplified, called SD), the denotation p of the program, its subposet of retracts R and the constraint C.



- let $\gamma : \mathcal{L} \to Sub$ be defined by
  - for a and b of dimension n, $\gamma(a < b) = \vee\{x \in Sub / \forall p \in P_n(x), p = (p_i)_{i \geq 0}, \forall i, p_i = a \Rightarrow p_{i+1} = b)\}$,
  - $\gamma(p \vee q) = \gamma(p) \vee \gamma(q)$.
- and let $\alpha : Sub \to \mathcal{L}$ be as usual $\alpha(x) = \wedge\{y \in \mathcal{L} / x \leq \gamma(y)\}$.

In Figure 9.2 we have pictured a constraint on scheduling C which can be described as $C = c < d \vee a < b'$ as well as a program semantics p. In Figure 9.1, the constraints are written next to the corresponding elements of SD.

9.1.9 Verification of protocols

Given a constraint (or protocol) $C \in \mathcal{L}$ and a program p (identified with its semantics in D), can we find a best scheduler for p under constraint C? This question is two-fold: first (verification), can p be scheduled with constraint C? and second, as a side-effect, (inference) what is a best way, i.e. a way in which we add the minimum number of constraints to C?

This problem can be expressed in our framework as follows. What is the maximal element of the intersection of the subposet of retracts of p with the left-closed set of elements satisfying the constraint C: $\{y \in SD / y \leq C\}$? or using a geometric image: "can we retract p onto $C \cap p$?". As an example, $a + b'$ is this maximal element in Figure 9.2. An algorithm is given in next section (which mainly computes a part of the α of last section).

9.1.10 Inference of a best parallelization

Here, we are given a sequential program p (identified with a HDA of dimension one) and we want to give a meaning to the problem of finding the "best" parallelization of it. The way we do this is by considering p to be embedded into a domain D specifying all possible actions. Practically, this is done by considering all traces in which all actions of p are put in parallel. This may obviously create some interferences or demonstrate the ability to perform some parts of p in parallel. Now, instead of retracting paths onto p, we wish to extend

p as much as we can: we wish to find the greatest subHDA of D that retracts onto p. This makes sense since Sub is complete. We derive an algorithm in next section.

9.2 Algorithmic details

As we do not want to specialize to a particular language or subproblem here we choose to give generic algorithms which may be optimized for some specific areas (see the conclusion). The generic algorithm relies on finding a solution to a central problem **cryptographists** have (finding dependence relations among lines of huge sparse matrices for quadratic sieves for instance). Huge progress is made everyday on finding good algorithms for solving this problem and these are then of direct interest for our abstract interpretation. For keeping things understandable we use only fairly well known algorithms in next sections.

9.2.1 Representation of HDA

The first simplification is to work with $R = \mathbb{Z}/2\mathbb{Z}$. This makes coefficients into simple booleans. Now boundary operators can be represented as boolean matrices, and even sparse ones: this means that lines (or line vectors) of the matrices are represented as ordered lists of integers indicating the occurrences of ones. Addition of line-vectors is a fusion of ordered lists which can be performed in linear time in terms of the number of elements in the lists.

HDA are represented by the matrices of their boundary operators in every dimension. Whenever we have to work on two HDA one included into the other, we mark some of the line vectors of the greater one to indicate that they generate the other HDA as well.

Example 45 *Let M be the HDA (a square),*

$$\begin{array}{ccccc} (A) & \xrightarrow{\partial_0} & (a)\oplus(b) & \longrightarrow & (\alpha) \\ \partial_1\downarrow & & \partial_1\downarrow & & \\ (a')\oplus(b') & \xrightarrow{\partial_0} & (\beta)\oplus(\gamma) & & \\ \partial_1\downarrow & & & & \\ (\delta) & & & & \end{array}$$

with $\partial_0(A) = a - b$, $\partial_1(A) = a' - b'$, $\partial_0(a) = \partial_0(b) = \alpha$, $\partial_0(a') = \partial_1(b) = \gamma$, $\partial_0(b') = \partial_1(a) = \beta$ and $\partial_1(a') = \partial_1(b') = \delta$. Then the matrix representing the total boundary operator in dimension 2 is,

$$\begin{pmatrix} & a & b & a' & b' \end{pmatrix}$$

$$\begin{pmatrix} A: & 1 & 1 & 1 & 1 \end{pmatrix}$$

and in dimension 1,

$$\begin{array}{c} (\quad\quad \alpha \quad \beta \quad \gamma \quad \delta \ ) \\ \left(\begin{array}{lcccc} a: & 1 & 1 & 0 & 0 \\ b: & 1 & 0 & 1 & 0 \\ a': & 0 & 0 & 1 & 1 \\ b': & 0 & 1 & 0 & 1 \end{array} \right) \end{array}$$

9.2.2 Representation of program semantics and constraints

We generate the program semantics by **compositional methods** like in [Gou93, GJ92] or in Section 5.5 and then compute the abstract operators using standard methods from homology theory or preferably here by **SOS-like rules** (Chapter 2) that generate all possible transitions. The constraints (given in $\mathcal{L}$) filter the application of the SOS rules: the ones that verify the constraints are then transformed into marked lines. For the inference problem, the domain is generated by applying all valid rules for all instructions (this should better be done lazily) and p is marked.

Example 46 *The CCS term $a \mid b$ has total boundary matrices as in Example 45 using the SOS-rules (or the denotational rules) of Chapter 5.*

9.2.3 Verification

The algorithm can then be described as follows. Are given n, the representation of the HDA $C \cap p$ and p, initial $(n-1)$-transitions I (a line vector) and final $(n-1)$-transitions F (a line vector). The algorithm says if we can n-schedule p under the constraint C. In order to be able to characterize the homotopy through homology, we have to suppose that p and $C \cap p$ do not contain any deadlock. This means that I and F must represent in an exact manner the sets of initial and final states of C and $C \cap p$.

We suppose that we have already implemented the following functions:

- $shift(M : HDA, k : integer)$ which shifts the dimension index of M by k, i.e. $shift(M,k) = N : HDA$ with $N_n = M_{n+k}$,
- $quotient(M : HDA; I, F : vector)$ which returns the HDA M' where all states in $I - F$ are replaced by 0,
- $tot(M : HDA)$ which returns the matrices M_1 for the boundary operator $\partial_0 - \partial_1$ in dimension one and M_2 in dimension two,

We first program $triangular(U : matrix) = (U' : matrix, P : matrix)$ where U' is a triangular form of both the submatrix of marked lines of U and a triangular form of U, and P is the matrix of change of coordinates. In this way, null lines

Figure 9.3: The triangular algorithm

$$
\begin{array}{l}
P = Id \\
\sigma = [1, 2, \ldots] \\
u = 1 \\
for\ i = 1\ to\ m \\
\quad find\ l_j,\ marked(l_j)\ and\ first(l_j) = i \\
\quad if\ found \quad then \left\{ \begin{array}{l}
\sigma(u) \leftarrow i \\
\sigma(i) \leftarrow u \\
u \leftarrow u + 1 \\
for\ k = 1\ to\ m,\ k \neq j \\
\qquad if\ first(l_j) = i \quad then \left\{ \begin{array}{l} l_k \leftarrow l_j + l_k \\ p_k \leftarrow p_j + p_k \end{array} \right.
\end{array} \right. \\
for\ i = u + 1\ to\ m \\
\quad find\ l_j,\ not\ marked(l_j)\ and\ first(\sigma(l_j)) = \sigma(i) \\
\quad if\ found \\
\quad then \left\{ \begin{array}{l}
for\ k = 1\ to\ m\ k \neq j \\
\qquad if\ first(\sigma(l_j)) = \sigma(k) \\
\qquad then \left\{ \begin{array}{l} l_k \leftarrow l_j + l_k \\ p_k \leftarrow p_j + p_k \end{array} \right.
\end{array} \right.
\end{array}
$$

where m, l_j, p_j and $first(l_j)$ denote respectively the number of line vectors in U, the jth line of the matrix U, the jth line of the matrix P and the first index i for which $l_j(i)$ is one.

of U' correspond to generators (whose expression can be read in P) of $Ker\ U$ and the non null lines of U' give a basis of $Im\ U$.

If implemented by (a version of) Gauss method as outlined in Figure 9.3 then the worst case complexity of $triangular(U)$ is in $O(ij^2)$ where j is the number of lines in U and i is the maximum number of non-null elements per line in U.

If $tot(quotient(M, i, F)) = (M_1, M_2)$ where M is replaced by $shift(M, n-1)$ then the null lines of $triangular(M_1)$ represent the generators of $Ker(\partial_0 - \partial_1)$ in dimension one, i.e. the generators of the set of 1-paths of M (see [Gou95] or Chapter 7).

They can be computed in $O(nk_n^2)$ where k_n is the number of n-transitions in M. Similarly, the non-null lines of $triangular(M_2)$ represent the generators of $Im(\partial_0 - \partial_1)$ in dimension two. This can be computed in $O(nk_{n+1}^2)$.

Figure 9.4: The algorithm $n - scheduler$

1) $p = s(p, n-1)$
2) $(M_1, M_2) = tot(quotient(p, I, F))$
3) $(U_1, P_1) = triangular(M_1)$
4) $(U_2, P_2) = triangular(M_2)$
5) $N = \begin{pmatrix} N_u = \text{lines } q_j \text{ of } P_1 \text{ with } l_j = 0 \text{ in } U_1 \\ N_l = \text{non-null unmarked lines of } U_2 \end{pmatrix}$
6) $(N_1, Q_1) = triangular(N)$
7) $result = (card\{j/n_j = 0\} == card\{unmarked\ lines\ of\ N_u\})$

The calculus of representants of generators for $\Pi_1(M)$ could be done in $O((n + k_1 + k_2)k_1^2 + (n + k_1)k_2^2)$ in the worst case. But here we are interested in a somewhat more specific problem.

The algorithm $n - scheduler(p, C \cap p : HDA; I, F : vector) : boolean$ for verifying if we can n-schedule p under constraint C is described in Figure 9.4 and runs in $O(n(k_n^2 + k_{n+1}^2))$ where k_n and k_{n+1} are the number of n transitions (resp. $(n+1)$-transitions) in p.

The algorithm works as follows.

At lines 1) and 2) are computed the matrices of the boundary operator $\partial_0 - \partial_1$ for the transitions of dimension n and $n+1$ of the pair $(p, I - F)$.

The triangulations of lines 3) and 4) are used at line 5) for generating a matrix N whose first part (N_u whose lines which correspond to paths in $C \cap p$ are marked) is composed of generators of n-paths of p and whose second part (N_v) is composed of generators of $(\partial_0 - \partial_1)(p_{n+1})$. The triangulation makes explicit all dependency relations between N_u and N_v, that is, shows how many n-paths are homotopic in p.

The last line verifies that all n-paths of p are equivalent to some path of $C \cap p$ by an easy argument on dimensions.

Example 47 *Take as domain and constraint those of the example pictured in Figure 9.2. For p, take the filled in square A. Then, the matrix representation of p is (where the marked lines are underlined), in dimension 1,*

$$\begin{pmatrix} a: & \underline{1} & \underline{1} & \underline{0} & \underline{0} \\ b: & 1 & 0 & 1 & 0 \\ a': & 0 & 0 & 1 & 1 \\ b': & \underline{0} & \underline{1} & \underline{0} & \underline{1} \end{pmatrix}$$

and in dimension 2,

$$D_2 = \begin{pmatrix} A: & 1 & 1 & 1 & 1 \end{pmatrix}$$

If we run the algorithm 1*-scheduler on these data, we first add column 4 to column 1 and discard 4 since it is all 1s (this gives the quotient by* $I - F$*). in the matrix representing the objects of dimension 1. Then we find,*

$$U_1 = \begin{pmatrix} \underline{1} & \underline{0} \\ 0 & 1 \\ 0 & 0 \\ \underline{0} & \underline{0} \end{pmatrix}$$

$$P_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

and $U_2 = D_2$*. Then,*

$$N = \begin{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 0 \\ \underline{1} & \underline{0} & \underline{0} & \underline{1} \end{pmatrix} \\ \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix} \end{pmatrix}$$

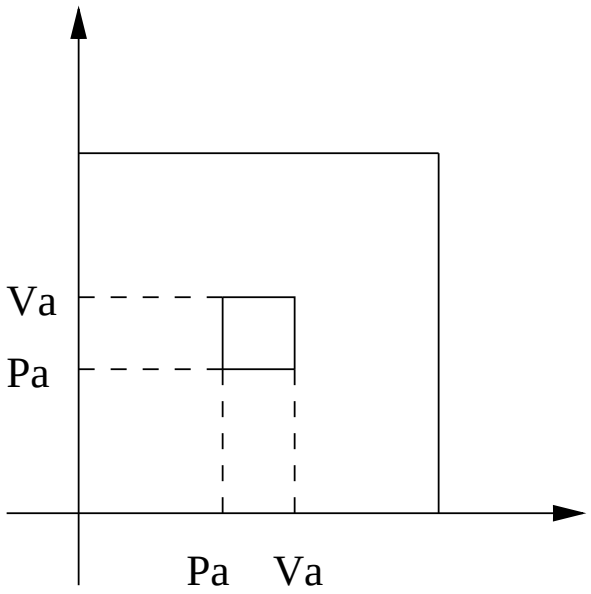
and

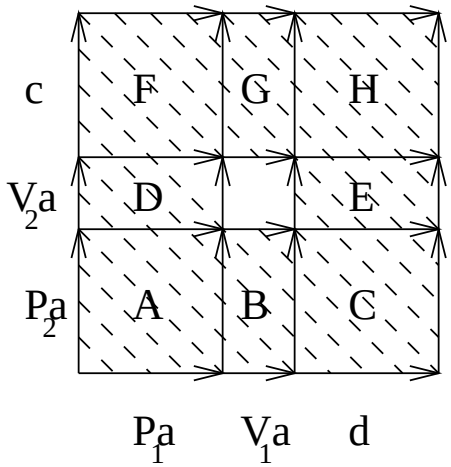
$$N_1 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ \underline{0} & \underline{0} & \underline{0} & \underline{0} \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

This entails $res = (1 = 1)$ *is true, hence* p *can be implemented on a machine with constraint* C*.*

Example 48 *Let us come back to the process graph for the two transactions* $T_1 = PaVa$ *and* $T_2 = PaVa$*. Its discretization as an HDA can be pictured as in Figure 9.5. It is obviously 2-phase locked hence serializable. Let us check this with our verification algorithm. We must check that it is equivalent to*

Figure 9.5: A process graph discretized as an HDA (8 2-transitions, 24 1-transitions and 16 states).





$p = T_1.T_2 + T_2.T_1$. *The matrix* M, *input of the algorithm 1-scheduler is then,*

$$D_1 = \begin{pmatrix}
\underline{1} & \underline{0} & \underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} \\
\underline{1} & \underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} \\
\underline{0} & \underline{1} & \underline{0} & \underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\underline{0} & \underline{0} & \underline{1} & \underline{0} & \underline{0} & \underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} \\
\underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\underline{0} & \underline{0} & \underline{0} & \underline{1} & \underline{0} & \underline{0} & \underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} \\
\underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{1} & \underline{0} & \underline{0} & \underline{1} & \underline{0} & \underline{0} & \underline{0} \\
\underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{1} & \underline{0} & \underline{1} & \underline{0} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
\underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{1} & \underline{0} & \underline{0} & \underline{1} & \underline{0} & \underline{0} \\
\underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{1} & \underline{0} & \underline{1} \\
\underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{1} & \underline{1}
\end{pmatrix}$$

And,

$$D_2 = \begin{pmatrix}
1&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\
0&0&1&1&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&1&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1
\end{pmatrix}$$

Therefore, adding last column to first one and discarding the last one, then triangulating these two matrices, we obtain,

$$N = \begin{pmatrix}
\underline{1}&\underline{0}&\underline{0}&\underline{0}&\underline{0}&\underline{1}&\underline{1}&\underline{0}&\underline{0}&\underline{0}&\underline{0}&\underline{0}&\underline{0}&\underline{0}&\underline{0}&\underline{0}&\underline{0}&\underline{1}&\underline{1}&\underline{0}&\underline{0}&\underline{0}&\underline{0}&\underline{1}\\
\underline{0}&\underline{1}&\underline{1}&\underline{0}&\underline{0}&\underline{0}&\underline{0}&\underline{0}&\underline{0}&\underline{0}&\underline{0}&\underline{1}&\underline{1}&\underline{0}&\underline{0}&\underline{0}&\underline{0}&\underline{0}&\underline{0}&\underline{0}&\underline{0}&\underline{1}&\underline{1}&\underline{0}\\
0&0&0&0&0&0&0&0&1&1&0&0&0&0&1&1&0&0&0&0&0&0&0&\\
1&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\
0&0&1&1&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&1&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1
\end{pmatrix}$$

$$\begin{pmatrix}
1&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\
0&0&1&1&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&1&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1
\end{pmatrix}$$

and,

$$N_1 = \begin{pmatrix}
\underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{1} & \underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{1} & \underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{1} \\
\underline{0} & \underline{1} & \underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{1} & \underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{0} & \underline{1} & \underline{1} & \underline{0} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1
\end{pmatrix}$$

We check that $card\{j/n_j \in N_1 = 0\} = 9 = card\{unmarked\, lines\, of\, N_u\}$. *Therefore it is serializable.*

9.2.4 Inference

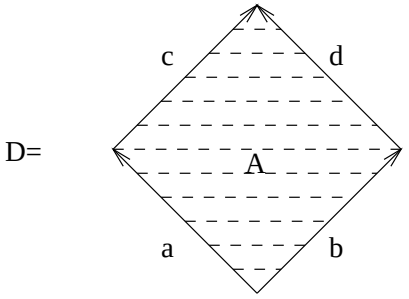
Are given, m, HDA D and p (of dimension one). p consists of the submatrix of marked lines of the matrix representation of D. The algorithm is an iteration on the following algorithm parameterized by the dimension n (we call Add_n), $for\ n = 1\ to\ m-1$, Add_n. We suppose that p does not contain any deadlock.

It is basically the same as the one in Figure 9.4 except the *result* assignment is replaced by a marking of M_2 which describes the $(n+1)$-transitions to be added to p to get its parallelization. The lines marked in U_2 are the lines corresponding to the $(n+1)$-transitions with non-null coefficient in line q_j of Q_1 such that $n_j \in N_1$ is null and marked. These are the $(n+1)$-transitions used for deforming some path in p to another path in the domain D. This marking is then translated into a marking of M_2 using the matrix of change of coordinates P_2. It is correct to use the homological characterization of homotopy since it is a “resolution”-like procedure: we start off with some connected k-connected

Figure 9.6: The 1-step inference algorithm Add_n

1) $p = s(p, n-1)$
2) $(M_1, M_2) = tot(quotient(p, I, F))$
3) $(U_1, P_1) = triangular(M_1)$
4) $(U_2, P_2) = triangular(M_2)$
5) $N = \begin{pmatrix} N_u = \text{lines } q_j \text{ of } P_1 \text{ with } l_j = 0 \text{ in } U_1 \\ N_l = \text{non-null unmarked lines of } U_2 \end{pmatrix}$
6) $(N_1, Q_1) = triangular(N)$
7) $mark\{$2-transitions with non-null coefficient in line q_j of Q_1 s.t. $n_j = 0$ in N_1 and $marked(n_j)\}$

Figure 9.7: A parallelization example.



shape to deduce a connected $(k+1)$-connected shape, so that Hurewicz holds at each step of the computation. The complexity is bounded by a function of order $O(n^2 max_{1 \leq i \leq n-1} k_i^2)$ (see Figure 9.6).

Example 49 *Take as domain and program those of the example pictured in Figure 9.7. Then, the matrix representation of D is (where the marked lines are underlined), in dimension 1,*

$$\begin{pmatrix} a: & 1 & 1 & 0 & 0 \\ b: & \underline{1} & \underline{0} & \underline{1} & \underline{0} \\ c: & 0 & 1 & 0 & 1 \\ d: & \underline{0} & \underline{0} & \underline{1} & \underline{1} \end{pmatrix}$$

and in dimension 2,

$$D_2 = \begin{pmatrix} A: & 1 & 1 & 1 & 1 \end{pmatrix}$$

If we run the algorithm Add_1 on these data, we first add column 4 to column 1 and discard 4 since it is all 1s (this gives the quotient by $I - F$) in the matrix representing the objects of dimension 1. Then we find,

$$U_1 = \begin{pmatrix} 1 & 0 \\ \underline{0} & \underline{1} \\ 0 & 0 \\ \underline{0} & \underline{0} \end{pmatrix}$$

$$P_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

and $U_2 = D_2$. *Then,*

$$N = \begin{pmatrix} \begin{pmatrix} 1 & 0 & 1 & 0 \\ \underline{0} & \underline{1} & \underline{0} & \underline{1} \end{pmatrix} \\ \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix} \end{pmatrix}$$

and

$$N_1 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ \underline{0} & \underline{0} & \underline{0} & \underline{0} \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

$$Q_1 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

We see that line n_2 *of* N_1 *is marked and is null. It corresponds to line* $q_2 = (1, 1, 1)$ *in* Q_1. *The third component of* q_2 *is one and corresponds to the 2-transition A. This means that it was used for deforming path* $b + d$. *This entails that A has to be marked as well as a and c. Therefore, p parallelizes to D.*

9.2.5 Optimizations

We can think of a number of optimizations,

- we can replace Gauss elimination technique by a faster algorithm like structured elimination or Wiedemann's probabilistic algorithm [Mas69]. The latter is quasi-linear in average in terms of the number N of non-null entries of the sparse matrix from which we want to extract linear dependencies. No experiments have been made yet as to know from which N on this algorithm gives practical pay offs.
- we might also enter an already triangulated matrix for some well-known domains. This would obviously greatly simplify the verification algorithm.

Summary We have defined a particular case of abstract interpretation of HDA in which all pre-orders we consider are suborders of the simulation pre-order (the one given by arrows in Υ). This was most suitably formalized using pairs of adjoint functors instead of the classical Galois connections. We have shown that there were some interesting abstractions, like the denotational one (looking only at initial and final states in the semantics) and the folding one (which identifies some n-transitions together). We developed in particular an abstract interpretation of the HDA semantics giving all the schedulers of a program, using the theory of Chapter 7. This allowed us to develop a verification algorithm (for checking if a program could be implemented using a given protocol on a given machine) and a parallelization algorithm.

The reader should realize that, as usual, all these abstractions can be composed together. In particular, we might use the folding abstraction to reduce the semantics of programs to finite HDA on which we can run the verification and parallelization algorithms. For more applications of these abstractions, the reader can look at [CG93, Cri95].

Part V

Extensions

Chapter 10

Combinatorial HDA

The homotopy theory we developed in Chapter 7 was quite degenerate in the sense that it was actually a homology theory. As a matter of fact it was restricted to very few HDA, namely those which are generated by the free construction on semi-regular HDA. We wish here to extend this to more general HDA. It is clear from Chapter 7 that what we need is essentially to use some kind of simplicial homotopy theory for both d^0 and d^1 boundary operators. This requires to add some "degeneracy" operators corresponding to these boundary operators. We construct such a model (called combinatorial HDA) in next section. Then we develop a homotopy theory which we finally compare with the one of Chapter 7.

There is another motivation for this construction. In [HR95] it is proven that "there is no t-resilient k-set agreement protocol on a shared memory machine with atomic read and write". The proof relies on a form of the "Acyclic Carrier Theorem" [Mun84] which holds only in the simplicial framework because we do need some "degeneracy operators" (i.e. some idle transitions of any dimension). In [HS94], it is also shown how to extract wait-free protocols from decision tasks in a constructive manner. This is done by subdividing the input complex in such a manner that the decision map becomes a simplicial map. We use the combinatorial construction of the first section to describe this subdivision. Notice that the semantic model is more powerful than the "static" one used by Herlihy et al. in that we do not ask for a fixed number of processes interacting throughout the distributed computation. The result of [HS94] is therefore generalized to distributed machines with, for instance, forking abilities.

10.1 Combinatorial HDA

Let $\underline{n}$ ($n \geq 0$) be the linear order $0 < 1 < \ldots < n-1$. It is well known (Appendix B and [May67, GZ67]) that the non-decreasing maps between any $\underline{k}$ and $\underline{l}$ are generated by the maps ($0 \leq i \leq n$), (Face maps)

$$u_i : \underline{n} \to \underline{n+1}$$

(Degeneracy maps)

$$v_i : \underline{n+1} \to \underline{n}$$

defined as,

$$u_i(k) = \begin{cases} k & \text{if } k < i \\ k+1 & \text{if } k \geq i \end{cases}$$

$$v_i(k) = \begin{cases} k & \text{if } k \leq i \\ k-1 & \text{if } k > i \end{cases}$$

They verify the following commutation rules, the "simplicial relations",

$$\begin{cases} u_j u_i = u_i u_{j-1} & (i < j) \\ v_j v_i = v_i v_{j+1} & (i \leq j) \\ v_j u_i = \begin{cases} u_i v_{j-1} & (i < j) \\ Id & (i = j, j+1) \\ u_{i-1} v_j & (i > j+1) \end{cases} \end{cases}$$

Now, instead of dealing with simplices, we want to deal with hypercubes.

"Oriented" hypercubes can be thought of as the Hasse diagrams of partial order between the states, "to occur before". This leads to considering non-strict-cover preserving maps, i.e. maps which do not reverse time and which are in some way simulations.

We are in particular interested in some maps between $\boxed{\text{k}} = \wp(\underline{k})$ and $\boxed{\text{l}} = \wp(\underline{l})$. These are generated by the following functions ($0 \leq i \leq n$),

$$\delta_i^0 : \boxed{\text{n}} \to \boxed{\text{n+1}}$$

$$\boxed{\text{n}} \underset{\theta_i}{\overset{\tau_i}{\rightrightarrows}} \boxed{\text{n}}$$

$$\boxed{\text{n+1}} \underset{\sigma_i^1}{\overset{\sigma_i^0}{\rightrightarrows}} \boxed{\text{n}}$$

defined as follows,

$$\delta_i^0(\{x_1, \ldots, x_k\}) = \{u_i(x_1), \ldots, u_i(x_k)\}$$

$$\tau_i(S) = \{i\} \cup S$$

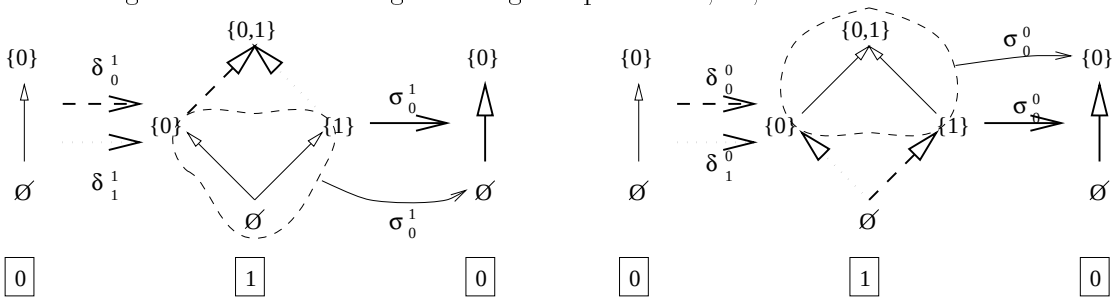
$$\theta_i(S) = S \backslash \{i\}$$

$$\sigma_i^0(\{x_1, \ldots, x_k\}) = \{v_i(x_1), \ldots, v_i(x_k)\}$$

$$\sigma_i^1(\{x_1, \ldots, x_k\}) = \cup(\{\!\{v_i(x_1), \ldots, v_i(x_k)\}\!\} - \{\!\{i\}\!\})$$

where $\{\!\{x_1, \ldots, x_k\}\!\}$, $-$ and $\cup$ in the last equality are respectively the multiset containing $x_1, \ldots, x_k$ (may have multiple occurences of some x_j), the difference between multisets and the union which from a multiset returns a set where we forget multiplicities.

Figure 10.1: Action of generating morphisms δ^1, δ^0, σ^1 and σ^0.



Notice that only σ_i^1 is not additive (i.e. does not always commute with $\cup$).

We conjecture that these morphisms generate all cover-preserving maps from any $(\boxed{\text{k}}, \subseteq)$ to any $(\boxed{\text{l}}, \subseteq)$ (look at Figure 10.1 for seeing the actions of these operators on the lattices $\boxed{\text{k}}$, represented geometrically).

These morphisms verify the following commutation rules,

$$\begin{aligned}
\delta_j^0\delta_i^0 &= \delta_i^0\delta_{j-1}^0 \quad (i<j)\\
\sigma_j^0\sigma_i^0 &= \sigma_i^0\sigma_{j+1}^0 \quad (i\le j)\\
\sigma_j^1\sigma_i^1 &= \sigma_i^1\sigma_{j+1}^1 \quad (i\le j)\\
\tau_i\tau_j &= \tau_j\tau_i\\
\theta_i\theta_j &= \theta_j\theta_i
\end{aligned}$$

$$\sigma_j^0\delta_i^0 = \begin{cases} \delta_i^0\sigma_{j-1}^0 & (i<j)\\ Id & (i=j,j+1)\\ \delta_{i-1}^0\sigma_j^0 & (i>j+1)\end{cases}$$

$$\sigma_j^1\delta_i^0 = \begin{cases} \delta_i^0\sigma_{j-1}^1 & (i<j)\\ \theta_i & (i=j,j+1)\\ \delta_{i-1}^0\sigma_j^1 & (i>j+1)\end{cases}$$

$$\delta_j^0\tau_i = \begin{cases} \tau_i\delta_j^0 & (i<j)\\ \tau_{i+1}\delta_j^0 & (i\ge i)\end{cases}$$

$$\delta_j^0\theta_i = \begin{cases} \theta_i\delta_j^0 & (i<j)\\ \theta_{i+1}\delta_j^0 & (i\ge j)\end{cases}$$

$$\sigma_j^0\tau_i = \begin{cases} \tau_i\sigma_j^0 & (i\le j)\\ \tau_{i-1}\sigma_j^0 & (i>j)\end{cases}$$

$$\sigma^0_j\theta_i = \begin{cases} \theta_i\sigma^0_j & (i<j) \\ \theta_{i-1}\sigma^0_j & (i\geq j) \end{cases}$$

$$\sigma^1_j\sigma^0_i = \begin{cases} \sigma^0_i\sigma^1_{j+1} & (i<j) \\ \sigma^1_j\sigma^1_j & (i=j,j+1) \\ \sigma^0_{i-1}\sigma^1_j & (i>j+1) \end{cases}$$

$$\sigma^1_j\tau_i = \begin{cases} \tau_i\sigma^1_j & (i<j) \\ \sigma^0_j & (i=j,j+1) \\ \tau_{i-1}\sigma^1_j & (i>j+1) \end{cases}$$

$$\tau_j\theta_i = \begin{cases} \tau_j & (i=j) \\ \theta_i\tau_j & (i\neq j) \end{cases}$$

$$\theta_i\tau_i = \theta_i$$

PROOF. Quite tedious. We derive some of the most interesting equations below.

We have already seen in Section 8.1.4 the commutation relations between the boundary operators.

Let us first give a few hints about the commutation relation between σ^1 and δ^0.

$$\sigma^1_i\delta^0_j(\{k\}) = \begin{cases} \{v_iu_j(k)\} & \text{if } u_j(k)\neq i \text{ and } u_j(k)\neq i+1, \\ \emptyset & \text{otherwise} \end{cases}$$

But $u_j(k)=i$ or $u_j(k)=i+1$ can happen only under two circumstances,

(1) $k<j$ and $k=i,i+1$,

(2) $k\geq j$ and $k=i-1,i$.

We can now make the following assumptions,

- if $j<i$, then case (1) is impossible, whereas case (2) can happen if $k=i-1$ or $k=i$. Then both $\sigma^1_i\delta^0_j(\{k\})$ and $\delta^0_j\sigma^1_{i-1}(\{k\})$ are void. Now, if $k\neq i-1$ and $k\neq i$ then by definition of the u_i and v_j, $v_iu_j=u_jv_{i-1}$. This entails the result.

- if $j>i+1$ then case (2) is impossible whereas case (1) can happen if $k=i$ or $k=i+1$. In that case, both $\sigma^1_i\delta^0_j(\{k\})$ and $\delta^0_j\sigma^1_{i-1}(\{k\})$ are void. If $k\neq i$ and $k\neq i+1$ then the commutation relation between u_j and v_i implies the result.

- if $j = i$ or $j = i + 1$ then it is easy to see that

$$\sigma_i^1 \delta_i^0(\{k\}) = \begin{cases} \emptyset & \text{if } k = i, \\ \{k\} & \text{otherwise} \end{cases}$$

  This is τ_i by definition.

□

These relations imply that if we set $\delta_j^1 = \tau_j \delta_j^0$ then,

Lemma 34 *(δ_i^0, σ_j^0) and (δ_i^1, σ_j^1) verify the simplicial relations and*

$$\delta_j^0 \delta_i^1 = \begin{cases} \delta_i^1 \delta_{j-1}^0 & (i < j) \\ \delta_{i+1}^1 \delta_j^0 & (i \geq j) \end{cases}$$

PROOF. Simple calculation using the commutation relations just proven previously. It is rather obvious for (δ_i^0, σ_j^0). Now for (δ_i^1, σ_j^1),

- if $i > j$ then,

$$\begin{aligned} \sigma_i^1 \delta_j^1 &= \sigma_i^1 \tau_j \delta_j^0 \\ &= \tau_j \sigma_i^1 \delta_j^0 \\ &= \tau_j \delta_j^0 \sigma_{i-1}^1 \\ &= \delta_j^1 \sigma_{i-1}^1 \end{aligned}$$

- if $j = i$ or $j = i + 1$ then,

$$\begin{aligned} \sigma_i^1 \delta_j^1 &= \sigma_i^1 \tau_j \delta_j^0 \\ &= \sigma_i^0 \delta_j^0 \\ &= Id \end{aligned}$$

- finally, if $j > i + 1$ then,

$$\begin{aligned} \sigma_i^1 \delta_j^1 &= \sigma_i^1 \tau_j \delta_j^0 \\ &= \tau_{j-1} \sigma_i^1 \delta_j^0 \\ &= \tau_{j-1} \delta_{j-1}^0 \sigma_i^1 \\ &= \delta_{j-1}^1 \sigma_i^1 \end{aligned}$$

Finally, we derive the commutation relations between the two different kinds of boundary operators,

- if $j < i$ then,

$$\begin{aligned} \delta^0_i \delta^1_j &= \delta^0_i \tau_j \delta^0_j \\ &= \tau_j \delta^0_i \delta^0_j \\ &= \delta^1_j \delta^0_{i-1} \end{aligned}$$

- otherwise, if $j \geq i$ then,

$$\begin{aligned} \delta^0_i \delta^1_j &= \delta^0_i \tau_j \delta^0_j \\ &= \tau_{j+1} \delta^0_i \delta^0_j \\ &= \delta^1_{j+1} \delta^0_i \end{aligned}$$

□

Let $\square$ be the category whose objects are the $\boxed{\text{k}}$ ($k \in \mathbb{N}$) and whose morphisms are generated by the morphisms δ^0_i, σ^0_i, τ_i, θ_i and σ^1_i.

We call any contravariant functor from the category $\square$ to some category $\mathcal{C}$ a cubical object in $\mathcal{C}$ or a cubical complex of elements of $\mathcal{C}$[1] (as for simplicial complexes, see Appendix B). The category of cubical objects in $\mathcal{C}$ is denoted $\square\mathcal{C}$. When $\mathcal{C} = R - Mod$, its elements are called combinatorial HDA.

The morphisms of combinatorial HDA are natural transformations. In particular any morphism f from F to G is such that,

$$G(\delta^0_i) \circ f = f \circ F(\delta^0_i)$$

$$G(\sigma^0_i) \circ f = f \circ F(\sigma^0_i)$$

$$G(\tau_i) \circ f = f \circ F(\tau_i)$$

$$G(\sigma^1_i) \circ f = f \circ F(\sigma^1_i)$$

The image of δ^0_i by any element F of $\square\mathcal{C}$ will be denoted by d^0_i. Similarly $F(\sigma^0_i) = s^0_i$, $F(\tau_i) = t_i$ and $F(\sigma^1_i) = s^1_i$. We write $d^1_i = d^0_i t_i$. The commutation relations we have proven show that (F, d^0_i, s^0_i) and (F, d^1_i, s^1_i) are simplicial objects in $\mathcal{C}$.

Example 50 *For $\mathcal{C} = \mathbf{Set}$, the contravariant functor,*

$$\begin{array}{rcl} \square_n : \square & \to & \mathbf{Set} \\ \boxed{k} & & Hom_{\square}(\boxed{k}, \boxed{n}) \end{array}$$

is called the standard n-cube.

[1] When $\mathcal{C} = \mathbf{Set}$ this is isomorphic to a subcategory of bipointed sets (private communication by Vaughan Pratt).

There are interesting relationships with semi-regular and general HDA.

There is an embedding functor E from $\square$ to $\boxed{\ }^{op}$ with,

- $E(\square_n) = \boxed{\text{n}}$,
- $E(d_l^k) = (\delta_l^k)^{op}$.

Therefore, there is a forgetful functor $Fg : \boxed{\ }\mathbf{Set} \rightarrow \Upsilon_{sr}$ such that $Fg(F) = F \circ E$,

Conversely, we know that all elements of Υ_{sr} are amalgamated sums of standard cubes (see Chapter 2). We can define a completion functor $Co : \Upsilon_{sr} \rightarrow \boxed{\ }\mathbf{Set}$ which, given $x \in \Upsilon_{sr}$ constructs $Co(x)$ by amalgamating the standard cubes (see Example 50) in the same way the standard cubes in Υ_{sr} were amalgamated to yield x. It can be shown that Fg is right-adjoint to Co.

Notice that the definition of acyclic combinatorial HDA is somewhat different from the one for general HDA. Degeneracies are cycles. Let $D(M) = \bigoplus_i (Im\ s_i^0 + Im\ s_i^1)$ be the module of degenerated elements of a combinatorial HDA M. Then we say that that M is acyclic if and only if $M/D(M)$ is acyclic in the usual sense, i.e. there is no x in both $M_{p,q}$ and $M_{p',q'}$, $p \neq p'$, $q \neq q'$.

We will show in Section 10.3 that a general HDA can be represented as a combinatorial HDA on $R - Mod$ and conversely.

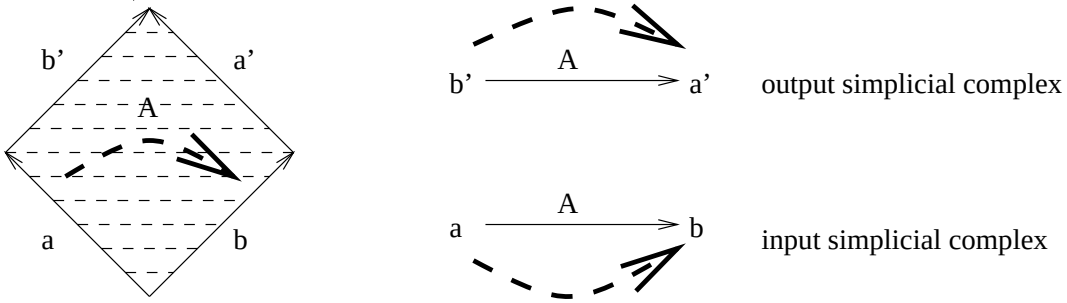
10.2 Homotopy theory

We have real simplicial sets in combinatorial HDA based on $\mathcal{C} = \mathbf{Set}$. The ones based on (δ^0, σ^0) are going to represent the input complexes in Herlihy's framework (see Chapter 7). The ones based on (δ^1, σ^1) are to represent the output complexes. What we need to do now is to adapt the homotopy theory we have developed for semi-regular HDA to the much more general combinatorial HDA. The idea is basically to have "elementary" deformations in our homotopy theory corresponding to an elementary deformation in the simplicial complex (δ^0, σ^0) and an elementary deformation in the simplicial complex (δ^1, σ^1) (see Figure 10.2).

First we recall (Appendix B) that for a simplicial complex $(K, \partial_i, \sigma_i)$ two n-simplexes x, x' are homotopic ($x \sim x'$) if and only if

- $\partial_i x = \partial_i x'$ for all $0 \leq i \leq n$,
- there exists a simplex $y \in K_{n+1}$, the "homotopy", such that,
  - $\partial_n y = x$,
  - $\partial_{n+1} y = x'$,
  - $\partial_i y = \sigma_{n-1}\partial_i x = \sigma_{n-1}\partial_i x'$ for all $0 \leq i < n$.

Figure 10.2: Homotopy in a combinatorial HDA (the curved arrows indicate the elementary deformations – left is the HDA, right is the corresponding simplicial complexes)



If K is a Kan complex, for instance if each K_n is a R-module and all the ∂_i and σ_i are R-module homomorphisms, then $\sim$ is an equivalence relation on the n-simplices of K, $n \geq 0$.

As for semi-regular HDA, we can decompose combinatorial HDA in order to have a notion of time. This is done by distinguishing among M_n, R-modules $M_{p,q}$ such that $M_n = \sum_{p+q=n} M_{p,q}$ and such that,

- $d_i^0 : M_{p,q} \to M_{p-1,q}$,
- $d_i^1 : M_{p,q} \to M_{p,q-1}$,
- $s_i^0 : M_{p,q} \to M_{p+1,q}$,
- $s_i^1 : M_{p,q} \to M_{p,q+1}$.

Let $(M, d_i^0, d_i^1, s_i^0, s_i^1)$ be a combinatorial HDA in $R - Mod$. Then we know by Lemma 34 that $(M_{j,*}, d_i^0, s_i^0) = H_j(M)$ and $(M_{*,j}, d_i^1, s_i^1) = V_j(M)$ are two simplicial R-modules, hence are two Kan complexes. Each one of them is thus equipped with a homotopy, $\sim_0$ and $\sim_1$ respectively.

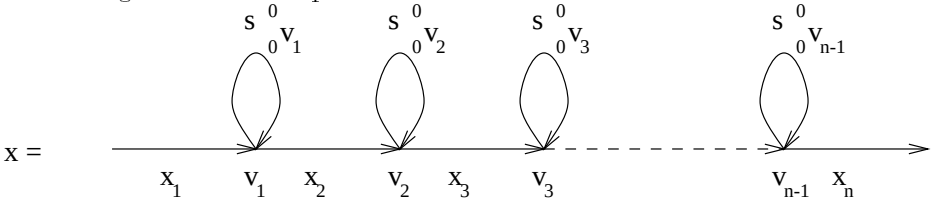
Definition and lemma 9 *Let S be the relation defined on n-cubes of M by,*

$$x = (x_1, \ldots, x_k \in M_{k,n-k}) S x' = (x'_1, \ldots, x'_k \in M_{k,n-k}) \Leftrightarrow \begin{cases} x_j = x'_j,\ j \neq i, i+1, \\ \exists y_0, H_{i-1}(x_i) \sim_0 H_{i-1}(x'_i) \\ \quad \textit{with homotopy } y_0, \\ \exists y_1, V_i(x_{i+1}) \sim_1 V_i(x'_{i+1}) \\ \quad \textit{with homotopy } y_1, \\ V_i(y_0) = H_{i-1}(y_1) \end{cases}$$

Then the transitive closure $\sim$ of S is an equivalence relation.

Let now Φ and Ψ be two states of a combinatorial HDA M. We define the set of 1-paths from Φ to Ψ to be $\tilde{M}_1^{\Phi,\Psi} = \{x \in M_1 / d_0^0(x) = \Phi, d_0^1(x) = \Psi\}$.

Figure 10.3: A 1-path in a combinatorial HDA.



i.e. $x = x_1 - s_0^0(v_1) + x_2 - \ldots - s_0^0(v_{n-1}) + x_n$

Note that this definition is a natural generalization of the definition of 1-paths on semi-regular or regular HDA in that we can now use the "idle" transitions to cancel the intermediate states between Φ and Ψ in the computation of the boundaries of a 1-path (see Figure 10.3).

Let $\pi_0(M) = \{(\Phi, \Psi) \in M_0 /$ there is a 1-path from Φ to $\Psi\}/T$ be the set of "oriented" connected components, where T is the symmetric closure of T' defined as follows,

$(\Phi_1, \Psi_1)T'(\Phi_2, \Psi_2)$ if and only if there is a 1-path from Φ_1 to Φ_2 and from Ψ_2 to Ψ_1.

Now we denote the set of 1-paths from Φ to Ψ modulo homotopy by $\pi_1^{\Phi,\Psi}(M) = \tilde{M}_1^{\Phi,\Psi}/\sim$.

We have an operation ; on paths defined as follows,

$$; : \tilde{M}_1^{\Phi,\Psi} \times \tilde{M}_1^{\Psi,\Lambda} \to \tilde{M}_1^{\Phi,\Lambda}$$

defined by $x; y = x + y - s_0^0(\Psi)$. This operation is well behaved with respect to homotopy in the sense that $x \sim x'$ and $y \sim y'$ implies $x; y \sim x'; y'$.

Then the full fundamental R-module of M is

$$\pi_1(M) = \sum_{(\Phi,\Psi)\in M_0} \pi_1^{\Phi,\Psi}(M)/\{[p];[q] = [p;q]\}$$

Let now $p = (p_i)_{0\leq i\leq k}$ be a 1-path. The set of n-loops around p is defined to be,

$$\begin{aligned} \tilde{M}_n^p &= \{x \in M_n / x = \textstyle\sum_i x_i \text{ and} \\ &\quad \forall i, j, 0 \leq i \leq k, 0 \leq j \leq n-1, d_j^0(x_i) = p_i, d_j^1(x_i) = p_{i+n-1}\} \end{aligned}$$

The nth homotopy set based at p is

$$\pi_n^p(M) = \tilde{M}_n^p/\sim$$

We say that M is n-connected if $dim\ \pi_k^p(M) = 1$ for all p and $k \leq n$, ∞-connected if n-connected for all n.

10.3 Homotopy theory of general HDA

Let $M \in \square R - Mod$. The image of δ_i^0, σ_i^0, σ_i^1, τ_i and θ_i by M are written respectively as d_i^0, s_i^0, s_i^1, t_i and T_i. We redefine in this section only $d_i^1 = d_{n-i}^0 t_{n-i} : M_{n+1} \to M_n$ $(0 \leq i \leq n)$, where M_n is a short notation for $M(\boxed{\text{n}})$. We also define $D(M)$ to be the module of degenerated elements i.e. $D(M) = \sum_i (Im\, s_i^0 + Im\, s_i^1)$.

Similarly to the standard construction for proving the equivalence of the category of simplicial modules with the category of complexes of modules [May67, GZ67], to every combinatorial automaton M we associate a general HDA $\Xi(M)$ by,

$$\Xi(M)_n = M_n \cap Ker d_0^0 \cap \ldots \cap Ker d_{n-2}^0 \cap Ker d_0^1 \cap \ldots \cap Ker d_{n-2}^1$$

as an R-module. The boundary operators ∂_0 and ∂_1 are defined as $\partial_0 : \Xi(M)_n \to \Xi(M)_{n-1}$, $\partial_0 = (-1)^{n-1} d_{n-1}^0$ and $\partial_1 : \Xi(M)_n \to \Xi(M)_{n-1}$, $\partial_1 = d_{n-1}^1$.

We have,

$$\begin{aligned} \partial_k \partial_k &= -d_{n-2}^k d_{n-1}^k \\ &= -d_{n-2}^k d_{n-2}^k \end{aligned}$$

which is 0 on $\Xi(M)_n$ by definition.

Moreover,

$$\begin{aligned} \partial_0 \partial_1 &= (-1)^{n-2} d_{n-2}^0 d_{n-1}^1 \\ &= -(-1)^{n-1} d_{n-2}^1 d_{n-1}^0 \\ &= -\partial_1 \partial_0 \end{aligned}$$

Therefore $\partial_0 \circ \partial_1 + \partial_1 \circ \partial_0 = 0$. $\Xi(M)$ is a general HDA.

Conversely, if P is a general HDA, we define

$$G(P)_{p,q} = P_{p,q} \oplus \left(\sum_{1 \leq i+j \leq p+q-2} \sum_{k=p-i, l=q-j} s_{u_k}^{\epsilon} \ldots s_{u_\alpha}^{\epsilon} s_{v_l}^{1-\epsilon} \ldots s_{v_\beta}^{1-\epsilon} s_{u_{k'}}^{\epsilon} \ldots P_{i,j} \right) / \mathcal{E}$$

where $\mathcal{E}$ is the equational theory of morphisms of $\square^{pp}$ with $p+q-1 > u_k > \ldots > u_\alpha \geq 0$, $p+q-1 > u_{k'} > \ldots \geq 0$ and $p+q-1 > v_l > \ldots > v_\beta \geq 0 \ldots$

On every $G(P)_{p,q}$ the boundary and degeneracy operators d^0, s^0 and d^1, s^1 are

$$\forall i < p+q-1,\ d_i^0 = d_i^1 = 0$$

$$\forall x \in P_{p,q},\ d_{p+q-1}^0(x) = (-1)^{p+q-1} \partial_0(x)$$

$$\forall x \in P_{p,q},\ d_{p+q-1}^1(x) = \partial_1(x)$$

and on degenerated elements, the boundaries are given by the equational theory of morphisms of $\square^{op}$.

Let P be a general HDA, $M = G(P)$. Then $P_n \subseteq Ker d^0_0 \cap \ldots \cap Ker d^1_{n-2}$ because, by definition of M, $d^0_0, \ldots, d^1_{n-2} = 0$ on P_n. Moreover, $x = s^\epsilon_i(y)$, for $y \in P_{n-1}$, $y \neq 0$, is certainly not in $Ker d^\epsilon_0 \cap \ldots \cap Ker d^\epsilon_{n-2}$, since $d^\epsilon_i(x) = y \neq 0$. Therefore, $\Xi(M) = \Xi \circ G(P) = P$ as a R-module. Notice that $\partial_0[\Xi \circ G(P)] = (-1)^{n-1} d^0_{n-1}[G(P)] = \partial_0[P]$ and $\partial_1[\Xi \circ G(P)] = d^1_{n-1}[G(P)] = \partial_1[P]$. This proves that $\Xi \circ G \cong Id$ in $\Upsilon_{\geq 0}$, the category of general HDA with no events (all elements are of positive dimension).

This shows that Ξ reaches all general HDA of positive dimension. We define the homotopy of a general HDA P by,

$$\pi_n(P) =_{def} \pi_n(G(P))$$

Remark: Similarly to the construction in the simplicial case (see [May67], Chapter V), we think that if M is a combinatorial HDA, $G \circ \Xi(M)$ is a composition of deformations each of which preserving one of the homotopies $\sim_0$ or $\sim_1$. Therefore $G \circ \Xi(M) \sim M$. Then $\pi_n(\Xi(M)) =_{def} \pi_n(G \circ \Xi(M)) = \pi_n(M)$. This would show that the definition of homotopy groups we have given for general HDA agrees with the definition we have given on combinatorial HDA.

10.4 Relationship with semi-regular HDA homotopy

We show here that the homotopy theory of combinatorial HDA is an extension of the homotopy theory of semi-regular HDA since we know by the previous section that it covers at least all general HDA.

Theorem 4 *For all Φ, Ψ states of some acyclic semi-regular HDA P, $\Pi_1^{\Phi,\Psi}(P)$ and $\pi_1^{\Phi,\Psi}(Co(P))$ are isomorphic R-modules. This implies that $\pi_1(Co(P)) \cong \Pi_1(P)$.*

Proof. To any 1-path p from Φ to Ψ in P, $p = (p_1, \ldots, p_k)$ we associate $\tilde{p} = p_1 + \ldots + p_k - s^0_0 d^1_0 p_1 - \ldots - s^0_0 d^1_0 p_{k-1}$ in $Co(P)$. The definition is valid since we have the cocartesian square (in Υ_{sr}),

$$\begin{array}{ccc} \coprod \square_0 & \xrightarrow{\coprod_i \dot{\sigma}_{p_i}} & (d^0_0 p_1) \coprod (d^1_0 p_1) \coprod \ldots \coprod (d^1_0 p_k) \\ \downarrow & & \downarrow \\ \coprod \square_1 & \xrightarrow{\coprod_i \sigma_{p_i}} & p \end{array}$$

hence a corresponding cocartesian diagram image by Co. This proves that $s^0_0 d^1_0 p_i \in Co(P)$ and $d^1_0(p_i) = d^0_0(p_{i+1})$ in $Co(P)$. Therefore, a simple calculation shows that $\tilde{p}$ is a 1-path from Φ to Ψ in $Co(P)$. In fact, $p \to \tilde{p}$ is an isomorphism

from the R-module of 1-paths of P to the R-module of 1-paths of $Co(P)$, $\tilde{M}_1^{\Phi,\Psi}$ as we show below.

Suppose $\Phi \in P_{0,0}$, $\Psi \in P_{k,-k}$. Let $p \in \tilde{M}_1^{\Phi,\Psi}$, $p = p_1 + \ldots + p_k$ where $p_i \in Co(P)_{i,1-i}$. We can write (for all i) $p_i = p'_i + \sigma_0^0(x_i)$, where the p'_i are in some supplementary of $Im\ \sigma_0^0$ in $Co(P)$. Then as $d_0^1 p_i = d_0^1 p'_i + x_i$, $d_0^0 p_i = d_0^0 p_i + x_i$, the fact that p is a 1-path from Φ to Ψ in $Co(P)$ implies that for all $2 \leq i \leq k-1$, $x_i = -d_0^0 p'_i = -d_0^1 p'_{i-1}$ and $d_0^0 p'_1 = \Phi$, $d_0^1 p'_k = \Psi$. Therefore $p' = (p'_1, \ldots, p'_k)$ is a 1-path in P and $p = \tilde{p'}$.

Let now p and q be two 1-paths from Φ to Ψ in P. Suppose p and q are elementary equivalent at time i, i.e. there exists $A \in P_2$ such that $p - q = (\partial_0 - \partial_1)(A)$.

In $H_{i-1}(Co(P))$, we have $\tilde{p}_i = p_i - s_0^0 d_0^0 p_i$ and $\tilde{q}_i = q_i - s_0^0 d_0^0 q_i$. Note that $d_0^0(\tilde{p}_i) = 0 = d_0^0(\tilde{q}_i)$.

Let $y_0 \in H_{i-1}(Co(P))$ be $y_0 = A - s_0^0 s_0^0 d_0^0 p_i$. Then $d_0^0(y_0) = p_i - s_0^0 d_0^0 p_i = \tilde{p}_i$ and $d_1^0(y_0) = q_i - s_0^0 d_0^0 p_i = \tilde{q}_i$. Therefore, $\tilde{p}_i$ and $\tilde{q}$ in $H_{i-1}(Co(P))$ with homotopy y_0.

In $V_i(Co(P))$, $\tilde{p}_{i+1} = p_{i+1}$ and $\tilde{q}_{i+1} = q_{i+1}$. Note that $d_0^1(\tilde{p}_{i+1}) = d_0^1(\tilde{q}_{i+1})$.

Let $y_1 = A \in V_i(Co(P))$. Then $d_0^1(y_1) = \tilde{q}_{i+1}$ and $d_1^1(A) = \tilde{p}_{i+1}$. Therefore $\tilde{q}_{i+1}$ and $\tilde{p}_{i+1}$ are homotopic in $V_i(Co(P))$ with homotopy y_1.

Finally, $V_i(y_1) = A = H_{i-1}(y_0)$ therefore $\tilde{p} S \tilde{q}$. This shows that $p \to \tilde{p}$ induces a map from $\Pi_1^{\Phi,\Psi}(P)$ to $\pi_1^{\Phi,\Psi}(Co(P))$.

We prove that it is an isomorphism by showing that $\tilde{p} S \tilde{q}$ implies p and q are elementary equivalent.

By definition of S, we have y_0 and y_1 such that,

- $d_0^0(y_0) = \tilde{p}_i$ and $d_1^0(y_0) = \tilde{q}_i$, $y_0 = A + s_0^0(x)$,
- $d_0^1(y_1) = \tilde{q}_{i+1}$ and $d_1^1(y_1) = \tilde{p}_{i+1}$, $y_1 = A + s_0^1(y)$.

Therefore, $d_0^0(y_0) = d_0^0(A) + x = p_i - s_0^0 d_0^0 p_i$. But $d_0^0(A)$ is not in $Im\ s_0^0$ therefore $x \in Im\ s_0^0$. Thus $d_0^0(y_0)\ mod\ Im\ s_0^0 = d_0^0(A) = p_i$. Similarly, $d_1^0(A) = q_i$, $d_0^1(A) = q_{i+1}$ and $d_1^1(A) = p_{i+1}$ and p and q are elementary equivalent. □

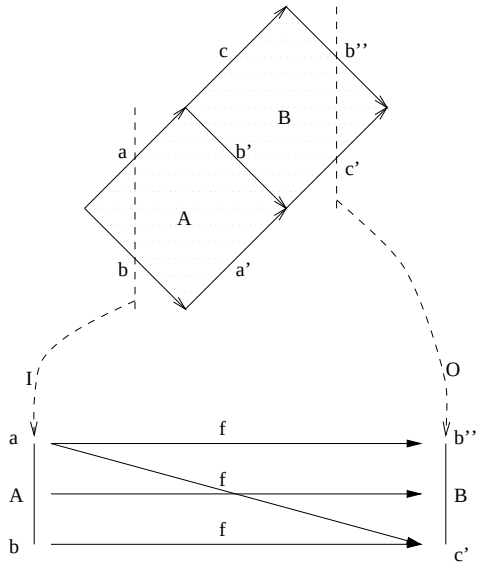
We have no result yet for higher-dimensional homotopy groups.

10.5 Construction of wait-free protocols

Let M be a combinatorial HDA, domain of possible executions of processes on a given machine , with some given inputs constrained to terminate with some prescribed output values. In other words, M is the domain specified by some decision task.

We suppose that $M_{*,0}$ is the first non-null line in M and that $M_{k,*}$ is the first non-null column of M. The input complex I of M is then the simplicial complex $(M_{*,0}, d^0, s^0)$ and the output complex O of M is the simplicial complex

Figure 10.4: A decision map read from a combinatorial HDA



$(M_{k,*}, d^1, s^1)$. Notice that points in these complexes correspond to 1-transitions in M.

We define now what is to be the decision map f from I to subcomplexes of O. Let $x \in I_0$ be a 1-transition. We set,

$$f(x) = \{ y \in O_0 / (p_1 = x, p_2, \ldots, p_{k-1}, p_k = y) \text{ 1-path in } M \}$$

This means that f sends all 1-transitions of I onto the set of 1-transitions of O which belong to the same 1-path of M.

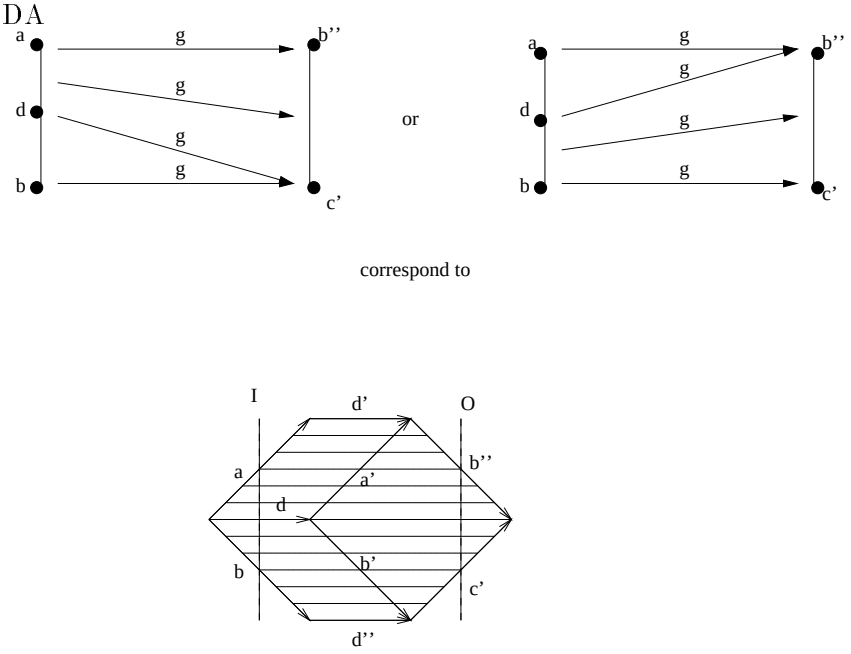
Similarly, we define for higher-dimensional transitions ($x \in I_n$, $n \geq 1$),

$$f(x) = \left\{ y \in O_n / \left\{ \begin{array}{l} \exists p, q \ n\text{-paths of } M \text{ s.t.} \\ (u_1 = x, u_2, \ldots, u_{k-n} = y) \ (n+1)\text{-path between } p \text{ and } q \end{array} \right\} \right.$$

We have pictured an example in Figure 10.4 (we have not pictured the degenerate transitions). The decision task is clearly wait-free since the HDA is 2-connected. A protocol for solving this decision task should solve the indeterminacy in the value of $f(a)$. The only way a protocol can do that is by having a hidden variable which will make the choice of taking a to b'' or a to c' from the very beginning of the execution. This means that the protocol is a map (which will actually be simplicial) from a subdivision $\sigma(I)$ to O. The new points in $\sigma(I)$ are executions of transitions of M where the value of the hidden variable differs from one point to the other. Two protocols and their corresponding maps are shown in Figure 10.5 together with the subdivided execution domain from which they can be extracted as decision maps from its input to its output complex.

Now we prove that what is exemplified in Figure 10.4 and Figure 10.5 is a general phenomenon.

Figure 10.5: Wait-free protocols corresponding to a subdivision of a combinatorial HDA



First of all, we have to define formally what subdivisions are.

(*) Definition 7 *[Spa66] A subdivision of a complex A is a complex B such that,*

(1) the vertices of B are points of $\mid A \mid$ (where $\mid . \mid$ is the geometric realization functor),

(2) if S is a simplex of B, there is a simplex $T \in A$ such that $S \subseteq \mid T \mid$,

(3) the piecewise linear map $\mid B \mid \rightarrow \mid A \mid$ mapping each vertex of B to the corresponding point of $\mid A \mid$ is a homeomorphism.

Theorem 5 *Let M be a ∞-connected combinatorial HDA, I, O and f, its input complex, its output complex and its decision map respectively. Then there exists a subdivision $\sigma(I)$ of I and a simplicial map $g : \sigma(I) \rightarrow O$ such that for all $x \in I$, $g(x) \subseteq f(x)$.*

SKETCH OF PROOF. Let x and y be elements of I_k such that $d^0_i(x) = d^0_i(y) = p_0$ for $0 \leq i \leq k-2$. M is ∞-connected implies that $x \sim_0 y$. Similarly, if x and y are elements of O_k such that $d^1_i(x) = d^1_i(y) = p_0$ for $0 \leq i \leq k-2$, $x \sim_1 y$ as M is ∞-connected.

O is therefore ∞-connected. A classical result [Mun84] asserts that for any subdivision σ if there is a simplicial map $\Phi : \sigma(\dot{\Delta}[j]) \rightarrow O$ (where $\Delta[j]$ is the standard simplex of dimension j) then there exists a subdivision τ and a simplicial map $\Psi : \tau(\Delta[j]) \rightarrow O$ such that,

- $\tau(\dot{\Delta}[j]) = \sigma(\dot{\Delta}[j])$,
- $\Phi_{|\sigma(\dot{\Delta}[j])} = \Psi_{|\tau(\dot{\Delta}[j])}$.

Therefore we can subdivide I by induction on the dimension. The important case is then the base case where we add more points to I.

Let $x \in I_0$. We know that $f(x) = \{y_1, \ldots, y_{k_x}\}$. We add new points to I, $z_2, \ldots, z_{k_x}$, defining a subdivision σ of I_0. We choose to define g by setting $g(x) = y_1$. This is done for all $x \in I_0$. g is defined on vertices so g is a simplicial map. The remark above gives us the rest of the subdivision. □

This result enables us to construct a protocol for solving our decision task since it is shown in [HS94] that there is an algorithm, the participating set algorithm, that solves the simplex agreement task, whose decision map is precisely the map g defined above. This corresponds to the generalization of the main result of [HS94] to a distributed machine with forking abilities for instance (since we are not compelled to have a constant dimension in all time slices of the HDA M) and refines the result of Section 8.2.3.

Summary We refined the semi-regular and regular HDA models by adding some degeneracy operators to the boundary operators describing the way n-transitions are part of $(n+1)$-transitions. The degenerate transitions can be seen as the higher-dimensional analogues to the idle transitions of ordinary transition systems (see Chapter 1). They can also be seen as the only construction that enables us to have a simplicial complex corresponding to each boundary operator. Then a homotopy theory of oriented paths has been constructed using the homotopy theory of each of the simplicial complexes. It was then shown that this allows us to have a homotopy theory for all general HDA, compatible with the one we have just defined. It was shown to coincide with the homotopy theory (at least when it comes to the fundamental groups) of Chapter 7 when restricted to semi-regular HDA. Finally, we gave an application by recasting Herlihy's method for building wait-free protocols from their decision maps and input and output complexes in an entirely semantic framework. This gives good promise for future generalizations of problems for complex distributed systems like systems with changing topologies and varying number of processes.

Chapter 11

Timed Higher-Dimensional Automata

11.1 Introduction

In this chapter, we are proposing an extension of HDA to deal with real-time systems.

Most existing models do not give a natural view on real-time systems (we refer the reader to Chapter 1 for a brief account of models of real-time systems). "Deadlock-freeness" is a property induced by the non-naturality of the base semantic model. Many technical details make the intuition about time disappear. One example is "time-determinism": all models should make the passage of time deterministic. The main problem in process algebras is the choice operator, as reckoned in [HR91],

> "If two processes are just idling before the environment requests one of them the choice between them will not be made by the passage of time alone. That is to say, + is not decided by the action σ. This is necessary to ensure that the passage of time is deterministic."

There is no problem as soon as time is measured on actions. Here we follow [Jos89] and use the ability of HDA to represent scheduling properties as a basis for a model of real-time systems where actions take time. A transition should really **take time** in the sense that it corresponds to an abstraction of some computation. As a matter of fact, we asked for refinability so we cannot assume actions to be only "elementary" – almost instantaneous – ones.

Unfortunately models for real-time concurrent systems having transitions bearing time changes can no longer be based on ordinary transition systems since interleaving of two actions a, b will result in having an execution time equal to the sum of the times a and b take. This obviously ruins all future reasoning and explains why this natural idea has never been formalized up to now (except in some restricted way in [CZ91]). A solution is to follow a **truly concurrent** operational approach. As more generally scheduling policies of processes onto

processors have a direct impact on the measure of time[1], it appears that we need more than that.

We need to be able to describe the **level of parallelism**, i.e. the number of busy processors at a given time. The main idea is to conceive executions as **geometric shapes**. Ordinary transition systems can already be thought of as one-dimensional **trajectories**. Then the asynchronous execution of n actions is a trajectory (or transition) of dimension n. We carry on by realizing these shapes in some euclidean space $\mathbb{R}^n$ (Section 11.1.2) as a basic step towards having execution time of transitions measured by their **length**. This situation is abstracted in Section 11.2 where the length depends on a norm associated with every transition. We construct a category of models (timed HDA) by defining morphisms to be "simulations" (as in recent work in concurrency, [WN94]).

A correctness criterion with respect to untimed semantics is obtained by forgetting the geometry and the norms (Section 11.2.2). Fairness (Section 11.2.1) is also discussed. In Section 11.2.3, Zeno behaviours are shown to be of a topological nature. Similarly to fairness properties, we propose to give a choice between allowing or not these behaviours.

Finally in Section 11.3, the model is shown to be **natural** in the sense that parallel composition, non-determistic choice, etc. with suitable timing laws are categorical combinators in the category of timed HDA. Moreover the model covers **different paradigms** since synchronized product and function spaces are again natural constructions. The category of timed HDA is actually a model for non-commutative intuitionistic **linear logic**. It has then enough categorical properties for being used for **denotational (or categorical) semantics**. A **SOS**-like metalanguage is defined for the operational semanticians and put to work on a toy language.

11.1.1 From the untimed to the timed world

In the formalization of semi-regular, regular and general HDA, we have forgotten the geometry. Let us have it back in the manner of the geometric realization functor (Chapter 2).

As a matter of fact, in order to introduce time in the model we already have, we are going to represent transitions as real continuous geometric objects. Continuous geometry is good for measuring time: the principle here is to have time measured by the **length** of transitions (or paths). Traces are then real **trajectories** as in mechanics. This is close to intuition, contrarily to most approaches **transitions take time**. Being interested in program analysis, where transitions are in fact abstractions of some complex process, this approach is very natural. In particular refinement comes then for free (see Figure 11.4 for an easy example).

We recapitulate the construction of the geometric realization functor.

[1] In most work on analysis of real-time languages, a "maximal parallelism" assumption is assumed. This clearly is too rigid when it comes to real machines, and leads to complicated discussions when one wants to change the scheduling policy in a semantics.

We associate with every n-transition x a unit cube of dimension n in $\mathbb{R}^n$,

$$\square_n = \{(t_0, \ldots, t_n)/\forall i, 0 \leq t_i \leq 1\}$$

Then, similarly to the glueing process for semi-regular HDA (Chapter 2), we glue these cubes together according to the values of the boundary functions. In order to do this, we need to define functions characterizing the boundaries of these unit cubes in $\mathbb{R}^n$. Let δ_i^k, $0 \leq i \leq n$, be the continuous functions ($n > 0$) from $\square_{n-1}$ to $\square_n$ with

$$\delta_i^k(t_0, \ldots, t_{n-1}) = (t_0, \ldots, t_{i-1}, k, t_i, \ldots, t_{n-1})$$

They describe how the boundaries of a cube can be included into it. Then ($i \leq j$),

$$\delta_i^k \circ \delta_j^l = \delta_{j+1}^l \circ \delta_i^k$$

Consider now, for a semi-regular HDA M, the set

$$\mathcal{R}(M) = \bigcup_{n, x \in M_n} (x, \square_n)$$

Each $(x, \square_n)$ inherits a topology given by the standard one on $\mathbf{R}^{n+1}$, thus $\mathcal{R}(M)$ is a topological space with the disjoint sum topology. Let $\equiv$ be the equivalence relation (the "glueing" relation) induced by the identities: $\forall k, i$, $x \in M_{n+1}$, $t \in \square_n$, $n \geq 0$, $(d_i^k(x), t) \equiv (x, \delta_i^k(t))$. Let

$$\mid M \mid = \mathcal{R}(M)/\equiv$$

It has a structure of topological space induced by $R(M)$. $\mid M \mid$ is called the **geometric realization** of M. It is easy to make this construction into a functor from Υ_{sr} to **Top**, the category of topological spaces with continuous maps. As observed in [GZ67], we can actually work in Ke the full subcategory of *Kelley spaces* (i.e. compactly generated topological spaces, [AM93]) instead of the entire category Top. The geometric realization functor has then fairly nice properties. When ranging over Ke it commutes (similarly to [GZ67]) with finite inverse limits and all colimits. All this gives us a hint about how to define timed higher-dimensional automata. A first step towards a general definition is given in next section.

11.1.2 Timing a semi-regular HDA

Let M be a semi-regular HDA. The standard way in mathematics to measure the length (time) of transitions in $\mid M \mid$ is to have a norm $\| \cdot \|_x$ on the tangent space at every $x \in M$ of the shapes we have.

Then the length of a transition a is the integral of the speed $\| \frac{d\gamma(t)}{dt} \|_{\gamma(t)}$ for a parametrization γ of a (it does not depend on the parametrization chosen).

$\mid M \mid$ has a well known differential structure. On every transition of dimension n, we put the norm $\|u_1, \ldots, u_n\|_{x_1, \ldots, x_n} = max\{\mid u_1 \mid, \ldots, \mid u_n \mid\}$. The norm chosen corresponds to giving all 1-transitions the unit duration and to have

Figure 11.1: Some paths in a Timed HDA

that when n processes run asynchronously, the time to complete them is the maximum of the times necessary to complete each of them. This corresponds to our view of independent processes running asynchronously.

For instance, in Figure 11.1, the geometric realization of the path is of length 2. The fully synchronous execution in the automaton at the right-hand side (the diagonal of the square from the starting point to the end) is of length 1.

This view to timed HDA, if encouraging, is not yet satisfactory. We have a very **rigid** notion of time in the sense that the norm has to be chosen uniformly for all transitions. We only have to abstract away from a so concrete representation in order to get what we need.

Look for instance at the classical billiard example (Figure 11.4): the representation of transitions has been chosen in order to fit to the trajectory of the ball; states are then the coordinates of the point representing them. The picture will become more general in next section.

11.2 Basic definitions

First of all, we need a geometric shape X to define a timed HDA, i.e. we need a topological space. There are many kinds of topological spaces. We have seen that timing semi-regular HDA only requires Kelley spaces. Actually, Kelley spaces seem to be a good choice. They have very good algebraic properties: they form a complete and cocomplete cartesian closed subcategory of **Top** [AM93].

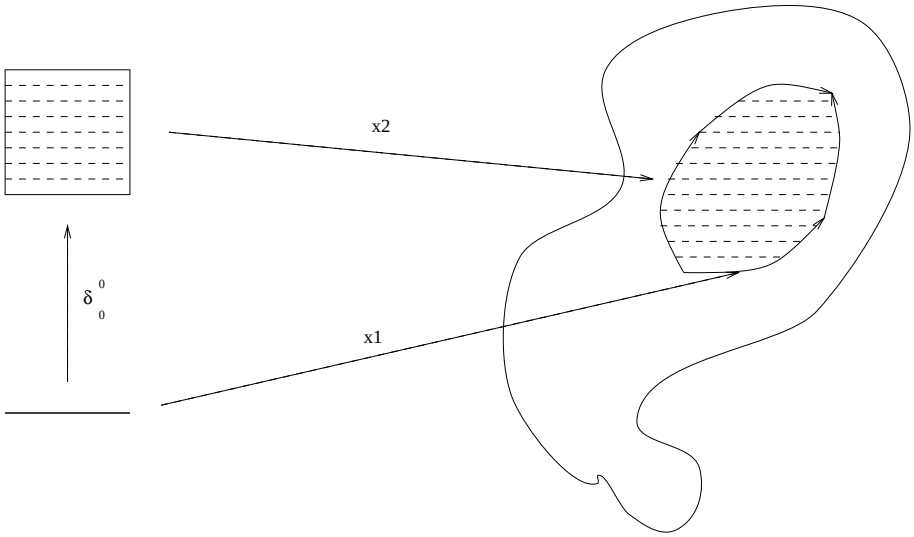
Then we have to give a differential structure on X to be able to measure time. It is difficult to do so in full generality. In particular, when it comes to algebraic properties, differential manifolds are difficult to handle[2]. We therefore choose to present here a very particular mathematical object, in which the differential structure is given by the transitions.

Thus we have to look at transitions now. Intuitively they should be sort of **deformed cubes** (see Figure 11.2). This leads us to define them as almost inclusion functions, i.e. as continuous functions $x : \square_n \to X$ (called singular cubes[3]). They are required to be continuously deformed cubes only in their interior since we may want to identify some of their boundaries to get cyclic

[2]Quotients, function spaces are hard work (they need submersion theorems and infinite dimensional differential geometry respectively).

[3]By analogy with singular simplices, [May67].

Figure 11.2: Deformed cubes



shapes. This is formalized by saying that all singular cubes $x : \square_n \to X$ induce homeomorphisms from[4] $\overset{\circ}{\square}_n$ to their images[5].

Moreover, we want X to be covered by all its transitions, i.e. we impose $\{x(\overset{\circ}{\square}_n)/n \in \mathbb{N}, x \in X_n\}$ to partition X, i.e. X is the disjoint union $\bigcup_{n\in\mathbb{N},x\in X_n}$ $x(\overset{\circ}{\square}_n)$. We should be able to take boundaries, i.e. the collection of singular cubes should be stable by composition with δ^ϵ_j (by Section 11.1.1).

Finally, on every tangent space $T_xX =_{def} T_xu(\overset{\circ}{\square}_n)$ (where $x \in u(\overset{\circ}{\square}_n)$, $u \in X_n$) of X at $x \in X$ we have a norm $\|\cdot\|_x$ such that $F(x, \dot{x}) = \|\dot{x}\|_x$ is a continuous function[6]. The norm can be seen as an **infinitesimal cost** for the computation at some point. To sum up things,

Definition and lemma 10 *A (unlabeled) timed HDA is a compactly generated Hausdorff topological space X together with a presentation of X by singular cubes, i.e. a sub-HDA of $\mathcal{S}(X)$ (a combinatorial cell complex in the terminology of [LW69]). This means that we have sets X_n containing singular cubes $x : \square_n \to X$ stable by composition with δ^ϵ_j. Moreover we impose the following conditions on X,*

- *$\{x(\overset{\circ}{\square}_n)/n \in \mathbb{N}, x \in X_n\}$ partition X, i.e. X is the disjoint union $\bigcup_{n\in\mathbb{N},x\in X_n}$ $x(\overset{\circ}{\square}_n)$,*
- *all singular cubes $x : \square_n \to X$ induce homeomorphisms from $\overset{\circ}{\square}_n$ to its*

[4] $\overset{\circ}{\square}_n$ denotes the topological interior of $\square_n$ i.e. $\overset{\circ}{\square}_n = \{0 < t_i < 1\}$, $n \geq 1$ and $\overset{\circ}{\square}_0 = \{0\}$.

[5] Therefore the singular cubes give a (trivial!) structure of manifold to all the $x(\overset{\circ}{\square}_n)$.

[6] If F is at least C^3 then this defines a Finsler space [Run59].

Recall that a norm F verifies the properties, $\forall k \in \mathbb{R}$, $F(x, k\dot{x}) = \mid k \mid F(x, \dot{x})$, $F(x, \dot{x}) \geq 0$ and $F(x, \dot{x}) = 0$ if and only if $\dot{x} = 0$, and $\forall x$, $\dot{x}$ and $\dot{x}'$, $F(x, \dot{x} + \dot{x}') \leq F(x, \dot{x}) + F(x, \dot{x}')$.

Figure 11.3: Delay transitions (left) and timeout HDA (right).

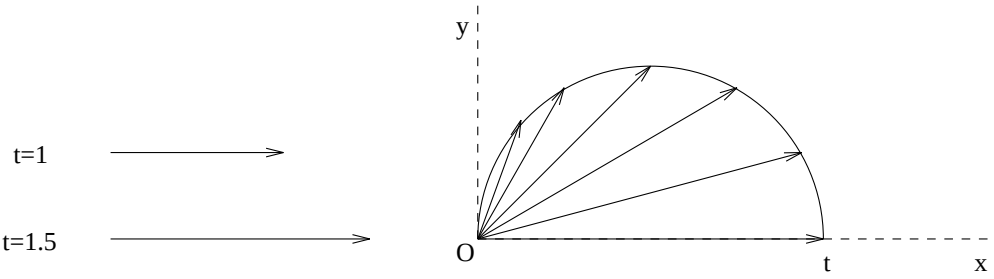


image. Therefore the singular cubes give a structure of manifold to all the $x(\overset{\circ}{\square}_n)$.

- *X is given a family of norms $\|.\|_x$ on every tangent space $T_xX =_{def} T_xu(\overset{\circ}{\square}_n)$ (where $x \in u(\overset{\circ}{\square}_n)$) of X at $x \in X$ such that,*
  - *$F(x,\dot{x}) = \|\dot{x}\|_x$ is a continuous function,*
  - *for all $k \in \mathbb{R}$, $F(x,k\dot{x}) =\mid k \mid F(x,\dot{x})$,*
  - *$F(x,\dot{x}) \geq 0$ and $F(x,\dot{x}) = 0$ if and only if $\dot{x} = 0$,*
  - *for all x, $\dot{x}$ and $\dot{x}'$, $F(x,\dot{x}+\dot{x}') \leq F(x,\dot{x}) + F(x,\dot{x}')$.*

PROOF. Things are particularly easy in our case. Define the atlas (see Appendix C) A_x of $x(\overset{\circ}{\square}_n)$ to be composed of the unique chart $(\overset{\circ}{\square}_n, x^{-1})$. It gives $x(\overset{\circ}{\square}_n)$ the structure of a differentiable manifold since $\overset{\circ}{\square}_n$ is an open subset of $\mathbb{R}^n$ and x^{-1} is an homeomorphism.
The local coordinates in the atlas A_x are denoted $(x_1,\ldots,x_n)$. □

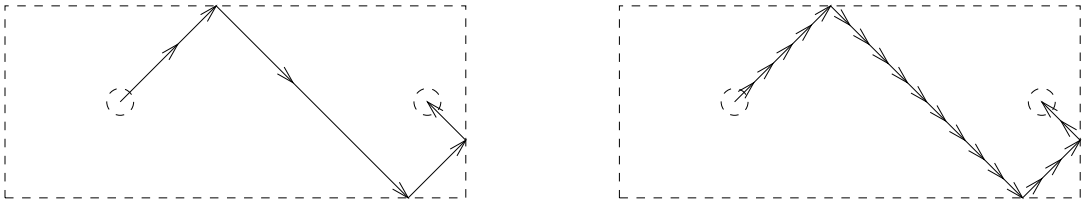
Example 51 *(see Figure 11.3)*

- *Let X_t ($t \in \mathbb{R}$) be the timed HDA generated by the unique 1-transition $\lambda x.tx : \square_1 \to t\square_1 = \{0 \leq x_1 \leq t\}$. $t\square_1$ is equipped with the norm $\|\dot{x}\|_x =\mid \dot{x} \mid$. We will see that it is a delay transition of duration t (similar to the δ_t operator of timed CCS).*

- *Define T_t to be the upper half circle of diameter t centered at coordinates $(\frac{t}{2},0)$ in the plane $\mathbb{R}^2$ (with its standard basis). It is given the structure of a timed HDA with the norm induced by the euclidean one in $\mathbb{R}^2$, and with the covering of 1-transitions (for $\theta \in [0,\pi/2]$) $x_\theta : \square_1 \to T_t$, $x_\theta(u) = (tucos^2(\theta), tusin(\theta)cos(\theta))$. We will see that it allows us to represent a timeout operator (t is the maximum waiting time).*

When X is a timed HDA, it is easy to see that the collection of sets X_n defines a semi-regular HDA. We define in a similar manner morphisms of timed HDA,

Figure 11.4: A timed HDA representing a billiard ball trajectory (i), and a refined version (ii).



(i) (ii)

Definition 52 *Let X and Y be timed HDA. A continuous function $f : X \to Y$ is a morphism of timed HDA if and only if,*

(i) *for all n-transition $x \in X_n$, there exists a n-transition $y \in Y_n$ such that*

$$\begin{array}{ccc} \square_n & \xrightarrow{x} & X \\ & \searrow^{y} & \downarrow f \\ & & Y \end{array}$$

commutes (which reads $y = f(x)$).

(ii) *f defines a differentiable function from all the $x(\overset{\circ}{\square}_n)$ to $y(\overset{\circ}{\square}_n)$, where x is a n-transition of X and y is its image by f (an n-transition of Y),*

(iii) *f commutes with all the boundary operators.*

Actually, since we are in a very special case, (i) implies (ii) since f is then the identity function in the local coordinates, thus a C^∞ diffeomorphism.

We write $T\Upsilon$ for the category of timed HDA.

Notice that no requirement has been made on the way morphims behave with respect to time. Choices are not so easy for "computer-scientific" reasons as well as for "technical reasons"[7]. Nevertheless, we will consider two subcategories of $T\Upsilon$, $T\Upsilon_=$ whose objects are timed HDA and whose morphisms $f : X \to Y$ preserve time (are *isometries*), i.e.

$$\|df(u).\dot{u}\|^Y_{f(u)} = \|\dot{u}\|^X_u$$

(where df is the differential of f), and $T\Upsilon_\leq$ whose objects are timed HDA and whose morphisms f contract time[8], i.e.

$$\|df(u).\dot{u}\|^Y_{f(u)} \leq \|\dot{u}\|^X_u$$

[7] Categories of metric spaces do not have very good algebraic properties in general. One must be careful when defining morphisms!

[8] Called conservative functions or non-expansive maps in categories of generalized metric spaces.

Remark as well that v being a n-transition of X is equivalent to v being a morphism from $\square_n$ (whose structure as a semi-regular HDA is generated by the only n-transition Id) to X.

Timed HDA are in particular semi-regular HDA. As such we know what is a path in it (we may add in particular initial and final states to timed HDA). But it is not clear however how to decide how much time a transition may take. To answer this question we define "virtual paths" in a timed HDA X as being particular curves on X of which paths are abstractions of, in some way.

Definition 53 *A virtual path γ in a timed HDA X is a continuous function $\gamma : [0, \infty[\to X$ such that,*

(i) *there exist open intervals $I_k =]\alpha_k, \alpha_{k+1}[$, 1-transitions x^k, $k = 0, \ldots, m-1$ such that $\alpha_0 = 0$, $\alpha_m = \infty$ and $\gamma_{|_{I_k}} : I_k \to x^k(\overset{\circ}{\square}_1)$ (then γ is of length k),*

(ii) *$\gamma_{|_{I_k}}$ is a differentiable function (I_k has the standard differentiable structure of $\mathbb{R}$),*

(iii) *$x_i^k \circ \gamma_{|_{I_k}}$ are increasing maps.*

The set of virtual paths from a point u to a point v is denoted by $\mathcal{V}i(u, v)$.

To determine the time that a path takes from its initial to its final point we use the metric generated by the norm on X[9].

Definition 54 *(see [Run59]) The distance inf between two points u and v in X is defined to be (with value in $\mathbb{R} \cup \infty$)*[10],

$$d_i^X(u, v) = inf_{\gamma \in \mathcal{V}i(u,v)} \int_0^\infty \| \tfrac{d\gamma}{dt}(t) \|_{\gamma(t)} dt$$

We have also the distance sup between two points (with value in $\mathbb{R} \cup \infty$),

$$d_s^X(u, v) = sup_{\gamma \in \mathcal{V}i(u,v)} \int_0^\infty \| \tfrac{d\gamma}{dt}(t) \|_{\gamma(t)} dt$$

d_i^X defines a distance function thus a metric on X (generalized in the sense that it may take infinite values).

In $T\Upsilon_=$, automata are simulated exactly in the same time (i.e. all virtual paths and their images have the same length).

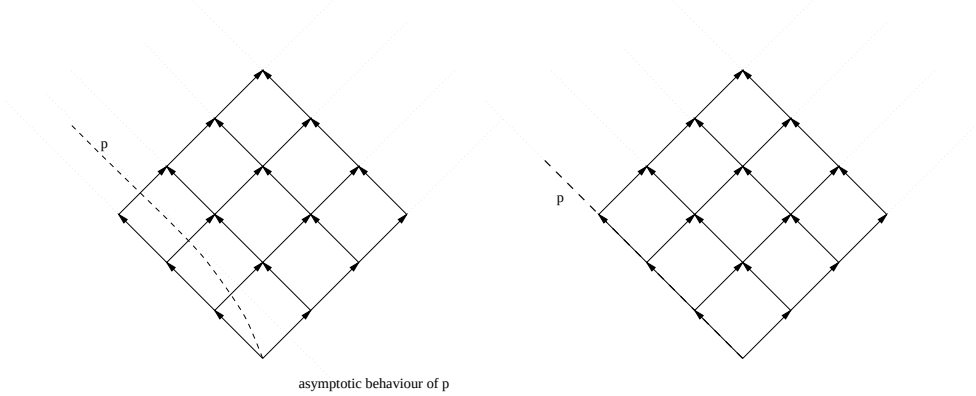
In $T\Upsilon_\leq$, we allow to simulate by **faster automata**. This is a sensible notion of simulation since programs can only be safely implemented on faster machines than needed[11].

[9] This is very close to the intuition behind the metric spaces models for real-time of [RR87].

[10] Where the integral is in fact the sum of the integrals on the open intervals I_k.

[11] There exist important properties that are not preserved when going on a faster machine (see [CZ91]), but this goes beyond the scope of this thesis.

Figure 11.5: A weakly fair path whose untimed version (right) is not fair



Example 52 *A simple computation shows now that X_t (Example 51) has length t, i.e. has execution time t.*

For T_t, the 1-transitions x_θ have execution time from 0 to t. The transition x_0 leads to the escape sequence, all the other ones lead to the normal ending of the program.

Finally, a hypercube of dimension n timed as in Section 11.1.2 has maximal execution time n (all interleavings) and minimal execution time 1 (synchronous execution of the n 1-transitions, i.e. the diagonal of the hypercube). We postpone the proof to Section 11.4.1.

Similarly to the untimed case, we can define **labeled** timed HDA to be unlabeled timed HDA plus a labelling morphism in $T\Upsilon$. **Timed higher-dimensional transition systems** are labeled timed HDA together with an initial state.

11.2.1 Fairness

Notice that we can easily define a time **local to a processor**. We can take for granted that in $\mid M \mid$ the length of the projection of a path γ on the ith coordinate is the cpu time of the ith processor on γ. More generally, we suppose that $\dot{x}_i(\frac{d\gamma}{dt})$[12] is the infinitesimal cost of computation on processor i.

We propose two notions of fairness.

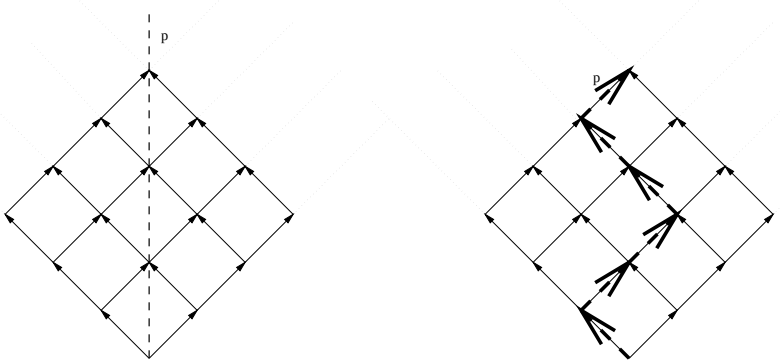
Quantitative weak fairness is expressed as a property of the norm: all processors must be used for some time on every (fair) paths, i.e.

$$\|(0,\ldots,\dot{x}_i(\tfrac{d\gamma}{dt}),0,\ldots,0)\|$$

should be a strictly positive function of time. This does not say anything about completing actions on some processor. This only enforces that all processors are used. This does not correspond to any good fairness property in the untimed case (see Figure 11.5).

[12]Where $\dot{x}_i$ denotes the ith coordinate in the tangent space.

Figure 11.6: A strongly fair path and its untimed fair version (right)



Quantitative strong fairness is a stronger property on the norm: whenever the global time diverges, the local times of every processor must diverge as well. One can show (see Figure 11.6) that this corresponds to strong fairness in the untimed case (when concurrency is modeled by interleaving of actions).

11.2.2 Correctness Timed/Untimed

Similarly to the work done in program analysis, we can define a way to go from the timed to the untimed world and then back to the timed one which has special properties. It is done in general [CC77] by means of Galois connections which ensure that an analysis (or a non-standard semantics) is **correct** with respect to a semantics.

Being in a completely categorical framework, the right mathematical tool is then pairs of adjoint functors. We actually have here a right-adjoint to the functor $|\cdot|$, $\mathcal{F}t : T\Upsilon \to \Upsilon_{sr}$ defined by $\mathcal{F}t(X) = (X_n)_n$ ($\mathcal{F}t$ forgets time).

Obviously, for all $M \in \Upsilon_{sr}$, $\mathcal{F}t(\mid M \mid_t) \cong M$. Let $\eta_X = Id : X \to \mathcal{F}t(\mid X \mid_t)$.

Now, $\mid \mathcal{F}t(X) \mid_{t_n} = \{(x, \square_n)_\sim / x \in X_n\}$ where $\sim$ is the equivalence relation generated by, $\forall t \in \square_{n-1}$, $(d_i^k(x), t) \sim (x, \delta_i^k(t))$. For $(x, \square_n) \in \mid \mathcal{F}t(X) \mid_{t_n}$ we have a morphism $x \circ pr_2 : (x, \square_n) \to X$. To have them extended to a morphism $\mid \mathcal{F}t(X) \mid_t \to X$ we have to verify that they are well-behaved with respect to $\sim$.

Let $t \in \square_{n-1}$ and $x \in X_n$. Then $(d_i^k(x), t) \sim (x, \delta_i^k(t))$. The morphism constructed from the right side is $x \circ \delta_i^k \circ pr_2 = d_i^k(x) \circ pr_2$ which is the morphism constructed from the left side. This concludes the construction of $\epsilon_X :\mid \mathcal{F}t(X) \mid_t \to X$.

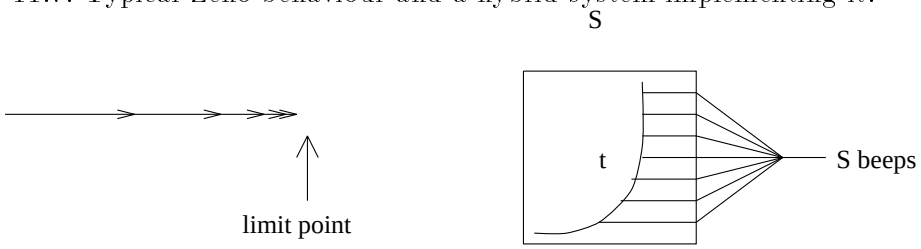
ϵ_X defines a natural transformation since ϵ_X is natural in X. As a matter of fact, let $f : X \to Y$ be a morphism in $T\Upsilon$ and $(x : \square_n \to X, t \in \square_n) \in \mid \mathcal{F}t(X) \mid_t$. $\mid \mathcal{F}t(f) \mid_t (x, t) = (f(x), t)$ and $\epsilon_Y(f(x), t) = f(x)(t) = f(x(t)) = f(\epsilon_X(x, t))$, hence the naturality in X.

We prove now that,

$$(1) : \mathcal{F}t(X) \xrightarrow{\eta_{\mathcal{F}t(X)}} F(\mid \mathcal{F}t(X) \mid_t) \xrightarrow{\mathcal{F}t(\epsilon_X)} \mathcal{F}t(X) = \quad Id$$

$$(2) : \mid M \mid_t \xrightarrow{\mid \eta_M \mid_t} \mid \mathcal{F}t(\mid M \mid_t) \mid \xrightarrow{\epsilon \mid M \mid_t} \mid M \mid_t = \quad Id$$

Figure 11.7: Typical Zeno behaviour and a hybrid system implementing it.



A timed HDA that could represent this behaviour is X with,

- $X = [0, 1]$,
- $X_0 = \{x_i / i \in \mathbb{N}, x_i = 1 - \frac{1}{2^i}\}$,
- $X_1 = \{[x_i, x_{i+1}] / i \in \mathbb{N}\}$ and the obvious boundary operators,
- the norm is induced by the euclidean norm on $[0, 1]$.

We only look at (1) since (2) is similar. For $x \in \mathcal{F}t(X)$ (i.e. x is in some X_n), $\mathcal{F}t(\epsilon_X) \circ \eta_{\mathcal{F}t(X)}(x) = \mathcal{F}t(\epsilon_X)(x, \square_n) = x$. Therefore $\mathcal{F}t(\epsilon_X) \circ \eta_{\mathcal{F}t(X)} = Id$. This completes the proof that $\mathcal{F}t$ is right adjoint to $|\cdot|_t$.

Notice that η_X and $\mathcal{F}t(f)$ (where f is a morphism of semi-regular HDA) are isometries. This entails that this adjunction restricts to adjunctions between $T\Upsilon_=$ and Υ_{sr}, and $T\Upsilon_\leq$ and Υ_{sr} respectively. Having simulations as morphisms in these categories, this shows that **simulation properties** (and bisimulation ones in particular) in the timed world are **correct** with respect to the corresponding ones in the untimed world.

11.2.3 Zeno behaviours

Let γ be an infinite virtual path. If $\gamma([0, \infty[)$ is compact, then there is a limit point a in the sequence $(\gamma(\alpha_k))_k$. Therefore, even if time always increases by strictly positive steps, there may be a (sub)path in which time "**slows down**" up to some point.

This is exemplified by a Zeno kind of paradox (which can be explicitly given a timed HDA representation, Figure 11.7) in which a door is seen to be closed through observations of the type "it is closed half way from the end". The time it needs to be closed is finite, the number of allowed obervations (transitions) is infinite. No lower bound whatsoever is imposed on the time of transitions. This precisely creates the paradox.

There are easy ways to prevent Zeno paradoxes to occur in a timed HDA X. As they happen when there exist some limit points, it suffices to prevent them to crop up.

A sufficient condition is to have a **lower bound** on the time transitions take. A better condition is the "finite variability" property: there should only be a finite number of actions that can be fired in a finite time interval.

Why not put this condition in the model from the very beginning ?

We argue that for hybrid systems (or even just ordinary real-time systems like in [MT90]), it may be interesting to consider Zeno paradoxes as well.

Suppose we have a system S in which the temperature t diverges in finite time (grows at an exponential rate in practise). Suppose also that S is equipped with a measuring apparatus which beeps every time the temperature grows by one degree Celsius. We model S by a timed HDA in which the states represent the number of times S has beeped (i.e. the temperature of S minus its initial value) and the 1-transitions are the delay transitions from one state to the next.

Then it implements a Zeno behaviour: no strictly positive lower bound can be given to the time of execution of any transition. As we do not know the **precision** at which time can be measured, we cannot eliminate this Zeno behaviour when studying the system S. Notice that Timed Transition Systems (Chapter 1) verify the finite variability condition and thus cannot express Zeno behaviours.

11.2.4 Complexity for constrained parallel machines

An easy computation shows that in the geometric realization of a semi-regular HDA $D_{[n]}$ generated by one n-transition there are minimal and maximal length virtual paths from $(0,0,\ldots,0)$ to $(1,1,\ldots,1)$ (see Section 11.4.1 for details).

There is one minimal length path γ_0 (the synchronous execution of n actions) up to parametrization given by (for all t), $x_1(\gamma_0(t)) = x_2(\gamma_0(t)) = \ldots = x_n(\gamma_0(t))$.

There are infinitely many maximal length paths γ_1 (the interleavings of parts of n actions) given by, $\forall t, \exists i, \forall j \neq i, \dot{x}_j(\gamma'(t)) = 0$. This entails that in $\mid D_n \mid$, there are $n!$ maximal paths, all of dimension one e.g. the interleavings of n actions, there is one minimal path, the path of maximal dimension.

If we measure **worst case** complexity by the maximum execution time, we see that truncating the behaviour of a program by T_n, i.e. executing the program on a machine with at most n processors, this worst case complexity gets bigger. This seems to provide a useful framework in which we can assess the complexity of algorithms on constrained parallel machines.

11.3 THDA as denotational and operational models

In this section, we show that $T\Upsilon$ is a complete and co-complete cartesian closed, monoidal closed category similarly to Υ_{sr}. Some constructions will be exemplified in both categories. $T\Upsilon_{\leq}$ is shown to be a complete and co-complete monoidal category. $T\Upsilon_{=}$ has only filtered limits and colimits and a tensor product. As customary since [WN94], categorical combinators will be recognized to be **timed-process-algebra sort of combinators** (as those of [MT90]).

In order to see this, we introduce a **SOS-like metalanguage** generalizing the one used for semi-regular HDA in Chapter 2 which gives an operational view of

the constructions. It is actually very close to the Timed Transition Diagrams formalism [HMP93] used to represent Timed Transition Systems in that it adds to ordinary transition systems upper and lower bounds within which actions are executed.

The idea is to write n-transitions a of some timed HDA X as arrows $s \xrightarrow[[t_1\,t_2]]{a} s'$ where s and s' are the beginning state (i.e. the beginning state of a beginning 1-transition of etc. a beginning $(n-1)$-transition of a) and end state of a respectively. t_1 is the minimal execution time, t_2 the maximum execution time of a (t_2 may be ∞ as we are working in $\mathbb{R} \cup \{\infty\}$. More formally, we define an entailment relation $\models$ to relate X to its transitions, and we write,

$$X \models s \xrightarrow[[t_1,t_2]]{a} s' \Leftrightarrow \begin{cases} d^0_0 d^0_1 \ldots d^0_{n-1} x &= s, \\ d^1_0 d^1_1 \ldots d^1_{n-1} x &= s', \\ T_i^{x(\square_n)}(s,s') &= t_1, \\ T_s^{x(\square_n)}(s,s') &= t_2 \end{cases}$$

Sometimes we specify the dimension n of the n-transition a by writing $dim\ a = n$.

11.3.1 Limits

Definition and lemma 11 *Let X and Y be two timed HDA. Then their cartesian product is the timed HDA Z given by,*

- $Z_n = \{z : \square_n \xrightarrow{\Delta} \square_n \times \square_n \xrightarrow{x \times y} X \times Y / x \in X_n, y \in Y_n\}$ *where Δ is the diagonal $\Delta(x) = (x,x)$,*
- $Z = \bigcup_{n \in \mathbb{N}, z \in Z_n} z(\square_n) \subseteq X \times Y$,
- $\|\dot{x}, \dot{y}\|_{(x,y)} = max(\|\dot{x}\|_x, \|\dot{y}\|_y)$.

PROOF. $X \times Y$ is a topological space with the product topology. We actually endow $X \times Y$ with the finer topology defined by,

"F is closed in $X \times Y$ (defining the new topology) if and only if for all compact subsets C of $X \times Y$ (under the old product topology) $C \cap F$ is closed (again under the old product topology)".

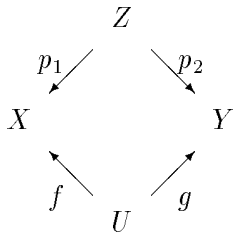
We write $K(X \times Y)$ for the topological space which has the same points as $X \times Y$ and whose topology is defined by the process above. $K(X \times Y)$ is a Kelley space and it is called the "Kelleyfication" of $X \times Y$ [ML71].

Now, $\square_n$ is compact. Thus $x(\square_n)$, $y(\square_n)$ are compact topological spaces since x and y are continuous. This entails that $z(\square_n) = (x \times y) \circ \Delta(\square_n)$ is compact. Therefore $z(\square_n)$ is Kelley as a closed set of the Kelley space $X \times Y$. The colimit $Z = \bigcup_{n, z \in Z_n} z(\square_n)$ of Kelley spaces is Kelley.

$\Delta : \overset{\circ}{\square}_n \to \Delta(\overset{\circ}{\square}_n)$ and $x \times y : \overset{\circ}{\square}_n \times \overset{\circ}{\square}_n \to x(\overset{\circ}{\square}_n) \times y(\overset{\circ}{\square}_n)$ are homeomorphisms therefore their composition is a homeomorphism as well. They partition Z.

The family of norms is well defined and is continuous on Z.

Let U be a timed HDA and f, g be two morphisms of timed HDA such that we have the following diagram,



where p_1 and p_2 are the first and second projections respectively. p_1 and p_2 are continuous and preserve the n-transitions. They induce the identity function on the local coordinates. Therefore they are morphisms of timed HDA.

Define then $h : U \to X \times Y$ by $h(u) = (f(u), g(u))$ for all u in U. We show that $h(U) \subseteq Z$. For all u in U, there exists n and $\alpha : \square_n \to U$ such that $u \in \alpha(\square_n)$. Then $f(\alpha)$ and $g(\alpha)$ are n-transitions of X and Y respectively. Therefore,

$$z : \square_n \xrightarrow{\Delta} \square_n \times \square_n \xrightarrow{f(\alpha) \times g(\alpha)} X \times Y$$

is a n-transition of Z. Finally $h(u) \in z(\square_n) \subseteq Z$.

This entails that h factorises f and g through p_1 and p_2 respectively. The fact it is the unique such map is obvious. □

It is described operationally by the rule,

$$\frac{X \models u \xrightarrow[[\alpha_1, \alpha_2]]{t} v \qquad\qquad X' \models u' \xrightarrow[[\alpha'_1, \alpha'_2]]{t'} v'}{X \times X' \models (u, u') \xrightarrow[[max(\alpha_1, \alpha'_1), max(\alpha_2, \alpha'_2)]]{(t, t')} (v, v')}$$

and $dim\ t = dim\ t' = dim\ (t, t')$

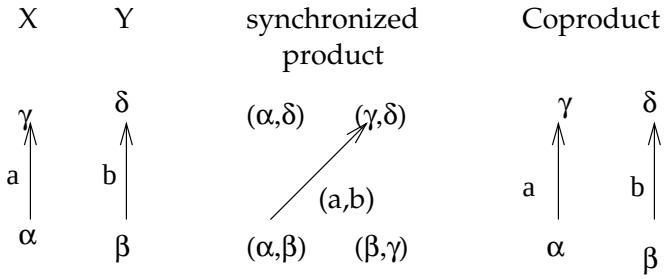
This shows that this is really a **synchronized product** (see Figure 11.8) of the two automata X and Y.

The projections here are not isometries in general, but they are contracting maps, i.e. $d_i^X(p_1(u), p_1(v)) \leq d_i^Z(u, v)$.

Lemma 35 *Let X and Y be two timed HDA and $f, g : X \to Y$ be morphisms of timed HDA. Then the timed HDA Z,*

- $Z = \{x \in X / f(x) = g(x)\}$,
- $Z_n = \{x \in X_n / f(x) = g(x)\}$,
- *the family of norms on TZ is induced by the family of norms in TX.*

Figure 11.8: Synchronized product (middle) and coproduct (right) of two transitions (left)



together with the inclusion map $Z \subseteq X$ is the equalizer of f and g.

PROOF. $z(\square_n)$ is a sub-Kelley space of X, therefore Z is a sub-Kelley space of X.

Let $y \in X$ such that $f(y) = g(y)$. There exists x, a n-transition of X, such that $y \in x(\overset{\circ}{\square}_n)$: $y = x(t)$ with $t \in \overset{\circ}{\square}_n$. $u = f(x)$ and $v = g(x)$ are n transitions of Y, but $u(t) = f(y) = g(y) = v(t)$. Therefore $u(\overset{\circ}{\square}_n) \cap v(\overset{\circ}{\square}_n) \neq \emptyset$ and then $u = v$. This shows that $z(\overset{\circ}{\square}_n)$, $z \in Z_n$ partition Z.

Finally, let U be a timed automaton and $h : U \to X$ a morphism such that $f \circ h = g \circ h$. Then for all x, $f(h(x)) = g(h(x))$ and then $h(x) \in Z$. This shows that $(Z, \subseteq)$ is the equalizer $equ(f, g)$. □

Note that the inclusion function is an isometry.

Therefore $T\Upsilon$ is finitely complete. The following lemma implies it is actually (small) complete.

Lemma 36 *Infinite products exist in $T\Upsilon$.*

PROOF. Let X^i be objects of $T\Upsilon$. We define X^∞ to be the timed automaton determined by the following data,

- as a topological space it is $K(\coprod_i X^i)$,
- as a semi-regular automaton it is the product of all the X^i,
- the norm is defined to be the supremum of all norms on all components (which may be infinite).

□

Again, the canonical projections are contracting maps.

11.3.2 Colimits

Definition and lemma 12 *We define the union of two timed HDA X and Y to be the timed HDA Z with,*

- $Z = X \cup Y$,
- $Z_n = X_n \cup Y_n$,
- *for all $u \in Z$, $u \in X$ and then $\|\dot{u}\|_u$ is the corresponding norm in X or $u \in Y$ and then $\|\dot{u}\|_u$ is the corresponding norm in Y.*

Z is then the coproduct of X and Y in $T\Upsilon$.

PROOF. Z is a topological space with the disjoint sum topology. With this topology, it is a Kelley space. Then

$$\bigcup_{n,z\in Z_n} z(\overset{\circ}{\square}_n) = \bigcup_{n,x\in X_n} x(\overset{\circ}{\square}_n) \cup \bigcup_{n,y\in Y_n} y(\overset{\circ}{\square}_n) = X \cup Y$$

all of these being disjoint. Moreover, all singular cubes of Z induce homeomorphisms from $\overset{\circ}{\square}_n$ to its image and the family of norms is well defined and continuous.

Define $in_l : X \to Z$ and $in_r : Y \to Z$ to be the inclusion functions. They are continuous functions preserving the n-transitions and the boundary operators. The local charts on Z are the union of the local charts of X and Y. Therefore in_l and in_r induce the identity function on the local coordinates. Hence they are C^∞ and are morphisms of $T\Upsilon$.

Now, if we have T a timed HDA and $f : X \to T$ and $g : Y \to T$ two morphisms of timed HDA then $h : Z \to T$ defined by $h(in_l(x)) = f(x)$, $h(in_r(x)) = g(x)$ is a morphism of timed HDA and is the unique morphism factorizing f and g through in_l and in_r respectively. The union defines the coproduct in $T\Upsilon_1$ as well. □

Notice that

$$\|din_l(u).\dot{u}\|^Z_{in_l(u)} = \|\dot{u}\|^X_u$$

$$\|din_r(u).\dot{u}\|^Z_{in_r(u)} = \|\dot{u}\|^X_u$$

which is actually equivalent to,

$$d^Z_i(in_l(u), in_l(v)) = d^X_i(u, v)$$

$$d^Z_i(in_r(u), in_r(v)) = d^Y_i(u, v)$$

i.e. in_l and in_r are isometries for the metric d_i.

The union (or coproduct) of two timed HDA X and Y is described operationally by the two rules,

$$\frac{X \models \quad u \quad \xrightarrow[[\alpha_1,\alpha_2]]{t} \quad v}{X \cup X' \models \quad u \xrightarrow[[\alpha_1,\alpha_2]]{t} v} \qquad \frac{X' \models \quad u' \quad \xrightarrow[[\alpha'_1,\alpha'_2]]{t'} \quad v'}{X \cup X' \models \quad u' \xrightarrow[[\alpha'_1,\alpha'_2]]{t'} v'}$$

We recognize a rule for **non-deterministic choice** (see Figure 11.8) more natural than the operators + in most of the existing process algebras.

Lemma 37 *Let X, Y be timed HDA and f, g be morphisms of timed HDA from X to Y. Then the timed HDA Z defined by,*

- $Z = Y/\{f(x) = g(x)\}$,
- $Z_n = Y_n/\{f(x) = g(x), x \in X_n\}$,
- $\|\cdot\|^Z = \|\cdot\|^Y$.

together with the projection from Y to Z is the coequalizer of f and g.

PROOF. Z is $Y/f(equ(f,g))$. Z is given the Kelley Hausdorff quotient topology and Z_n is the quotient in Υ_{sr}. The Z_n partition X. □

Note that the projection is an isometry for d_i.

Lemma 38 *$T\Upsilon$ has infinite coproducts.*

PROOF. Let X^i be objects in $T\Upsilon$. Define X^∞ to be the topological colimit of the X^i. It is a Kelley Hausdorff space. Let X^∞_n be the colimit of the X^i_n in Υ_{sr}. The X^∞_n partition X^∞. The norm is taken to be equal to all local norms on the X^i. □

Note that the canonical injections into this coproduct are isometries.

Therefore, $T\Upsilon$ is a (small) co-complete category.

11.3.3 Function space

Proposition 16 *$T\Upsilon$ is a cartesian closed category.*

PROOF. We define, for X and Y two timed HDA, $Z = X \Rightarrow Y$ to be,

- $Z_n = \{z : \square_n \to (X \to Y)/z' : \square_n \times X \to Y, z'(u,x) = z(u)(x) \text{ morphism}\}$, where $\square_n$ is considered as the timed HDA with the unique n-transition $Id : \square_n \to \square_n$,
- $Z = \bigcup_{n \in \mathbb{N}, z \in Z_n} z(\square_n) \subseteq (X \to Y)$,

- for $f \in Z$, $\|\dot{f}\|_f = sup_{x \in X, \|\dot{x}\|_x^X = 1} \|\dot{f}(x)\dot{x}\|_{f(x)}^Y$.

We give $X \Rightarrow Y \subseteq Hom(X,Y)$ the topology induced by the topology of $Hom(X,Y)$.

Let $z' : \square_n \times X \to Y$ be a morphism. Then $z = curry(z') : \square_n \to (X \to Y)$ is continuous. Therefore $z(\square_n)$ is compact hence a Kelley subspace of $Hom(X,Y)$. Then Z is Kelley as a colimit of Kelley spaces.

We prove that $X \Rightarrow Y$ is the exponent of Y by X in $T\Upsilon$.

Let $f : U \times X \to Y$ be a morphism in $T\Upsilon$. We define the function $g : U \to (X \to Y)$ as $g(u)(x) = f(u,x)$. We first prove that $Im\ g \subseteq X \Rightarrow Y$.

Let $u \in U$, $u \in v(\square_n)$ for some n and some n-transition $v : \square_n \to U$ of U. Then $u = v(\alpha)$ for some $\alpha \in \square_n$. Now, $f \circ (v \times Id)$ is a morphism as a composition of morphisms (v is a n-transition, thus a morphism). This means that the function $z : \square_n \to (X \to Y)$ with $z(b)(x) = f \circ (v \times Id)(b,x)$ is in $(X \Rightarrow Y)_n$. Notice that $z(\alpha) = g(u)$. Therefore, $g(u) \in z(\square_n) \subseteq (X \Rightarrow Y)$.

We then prove that g is a morphism of timed HDA. Let v be a n-transition of U, i.e. $v : \square_n \to U$. $g \circ v : \square_n \to (X \to Y)$ defines $g' : \square_n \times X \to$ by curryfication, $g'(u,x) = g(v(u))(x) = f(v(u),x)$. This entails that $g' = f \circ (v \times Id)$ which is a morphism of $T\Upsilon$. Therefore, $g \circ v \in (X \Rightarrow)_n$. g maps n-transitions onto n-transitions. Now, g is continuous is a standard result of topology [AM93], since we have chosen the compact-open topology on $X \Rightarrow Y$.

We end up by proving that the evaluation function $eval : X \times (X \Rightarrow Y) \to Y$ defined by $eval(x,f) = f(x)$ is a morphism of $T\Upsilon$. Standard results of topology show that $eval$ is continuous. We just have to prove now that $eval$ maps n-transitions onto n-transitions.

Let v be an n-transition of $X \times (X \Rightarrow Y)$. v is given by two n-transitions α of X and β of $(X \Rightarrow Y)$, $v : \square_n \stackrel{\Delta}{\to} \square_n \times \square_n \stackrel{\alpha \times \beta}{\to} X \times (X \Rightarrow Y)$. We have to prove that $eval \circ v$ is a n-transition of Y. β is a n-transition of $X \Rightarrow Y$ therefore, its uncurryfied version $\beta' : \square_n \times X \to Y$ is a morphism. Notice that $eval \circ v = \beta' \circ (Id \times \alpha)$ and thus is a morphism. By the characteristic property of n-transitions this entails that $eval \circ v$ is a n-transition. $\square$

In $T\Upsilon_{\leq}$ we have to restrict to HDA with "ultrametric" norms, i.e. such that

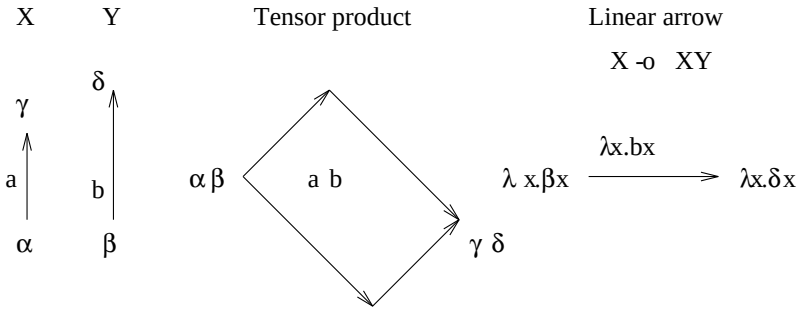
$$\|u + v\| \leq max(\|u\|, \|v\|)$$

11.3.4 Tensor product

Definition and lemma 13 *The tensor product of two timed HDA X and Y is the timed HDA $Z = X \otimes Y$ defined by,*

- $Z = X \times Y$,
- $Z_n = \{z : \square_n \cong \square_k \times \square_{n-k} \stackrel{x \times y}{\longrightarrow} Z / x \in X_k, y \in Y_{n-k}\}$
- $\|\dot{x}, \dot{y}\|_{(x,y)} = max(\|\dot{x}\|_x, \|\dot{y}\|_y)$.

Figure 11.9: Parallel composition (middle) of two transitions (left) and linear function space (right).



PROOF. Z is a Kelley space with the Kelleyfication of the product topology. The homeomorphism $\square_n \cong \square_k \times \square_{n-k}$ induces a homeomorphism $\overset{\circ}{\square}_n \cong \overset{\circ}{\square}_k \times \overset{\circ}{\square}_{n-k}$. Therefore, the singular cubes of Z_n define homeomorphisms from $\overset{\circ}{\square}_n$ to their image.

Finally, the family of norms is well defined and continuous. □

Notice that the semi-regular HDA given by the Z_n is the tensor product of the semi-regular HDA given by the X_n and Y_n.

The **parallel composition with no interference** can be defined operationally by the rule (see Figure 11.9)

$$\frac{X \models u \xrightarrow[[\alpha_1,\alpha_2]]{t} v \qquad X' \models u' \xrightarrow[[\alpha'_1,\alpha'_2]]{t'} v'}{X \otimes X' \models u \otimes u' \xrightarrow[[max(\alpha_1,\alpha'_1),\alpha_2+\alpha'_2]]{t \otimes t'} v \otimes v'}$$

and $dim\ t \otimes t' = dim\ t + dim\ t'$

Proposition 17 *$T\Upsilon$ is a monoidal closed category.*

PROOF. There are obvious isomorphisms between $(X \otimes Y) \otimes Z$ and $X \otimes (Y \otimes Z)$ and $X \otimes \square_0$ and X. This shows that $T\Upsilon$ is monoidal.

We now define, for X and Y two timed HDA, $Z = X \multimap Y$ to be,

- $Z_n = \{z : \square_n \to (X \to Y) / z' : \square_n \otimes X \to Y, z'(u,x) = z(u)(x)$ is a morphism$\}$, where $\square_n$ is considered as the timed HDA with the unique n-transition $Id : \square_n \to \square_n$,
- $Z = \bigcup_{n \in \mathbb{N}, z \in Z_n} z(\square_n) \subseteq (X \to Y)$ endowed with the compact-open topology,
- for $f \in Z$, $\|\dot{f}\|_f = sup_{x \in X, \|\dot{x}\|_x^X = 1} \|\dot{f}(x)\dot{x}\|_{f(x)}^Y$.

We prove that $T\Upsilon(X \otimes Y, Z) \cong T\Upsilon(X, Y \multimap Z)$ in a similar manner as for $X \Rightarrow Y$. $\square$

We conjecture that operationally[13],

$$\frac{X \models u \xrightarrow[[\alpha_1,\alpha_2]]{t} v \qquad X \multimap X' \models u' \xrightarrow[[\alpha'_1,\alpha'_2]]{t'} v'}{X' \models u'(u) \xrightarrow[[max(\alpha_1,\alpha'_1),\alpha_2+\alpha'_2]]{t'(t)} v'(v)}$$

In $X \multimap X'$ we have functions which **fork** new actions (dynamically) as $\lambda x.b \otimes x$ in Figure 11.9. The argument of these functions may be computed **in parallel** with the body of the function.

11.3.5 Labeled timed HDA and THTS

Definition 55 *A labeled timed HDA is a pair $(X, l : X \to L)$ of a timed HDA X and a morphism of timed HDA l. L is called the labelling automaton.*

Notice that all unlabeled timed HDA X can be considered as labeled timed HDA $(X, Id : X \to X)$.

Example 53 *In Figure 11.10 we have pictured the labeled timed HDA $(X, l : X \to L)$ defined as follows,*

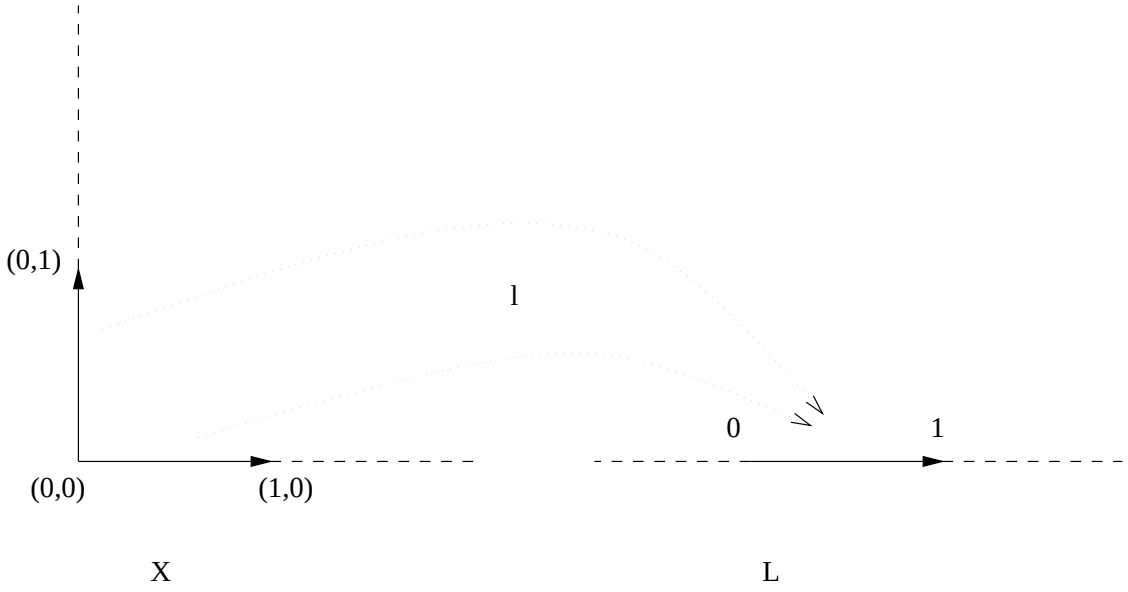
- $X = \{(x,0)/0 \leq x \leq 1\} \cup \{(0,y)/0 \leq y \leq 1\}$ *considered as a sub-topological space of $\mathbb{R}^2$ with the ordinary topology,*
- *the norms on TX are induced by the norm $\|(x,y,\dot{x},\dot{y})\| = max(|\dot{x}|, |\dot{y}|)$ on $T\mathbb{R}^2 \cong \mathbb{R}^4$,*
- $X_1 = \{a_1, a_2\}$ *with $a_1, a_2 : \square_1 \to X$ defined by,*
  - $a_1(x) = (x, 0)$,
  - $a_2(y) = (0, y)$.
- $L = \square_1$ *equipped with the norm $\|(x,\dot{x})\| = |\dot{x}|$,*
- $L_1 = \{a\}$ *with $a : \square_1 \to L$ being the identity function,*
- $l(0,y) = y$ *and $l(x,0) = x$ for all $0 \leq x \leq 1$ and $0 \leq y \leq 1$.*

Notice that if we ask the morphism l to be an isometry (as in Example 53) then the labels of actions prescribe the exact time they should take in any context. If we ask the morphism l to be non-expansive then, we are only prescribing a lower bound on the time actions with a given label may take.

Definition 56 *A timed higher-dimensional transition system (THTS) is a pair $((X, l : X \to L), s)$ where $(X, l : X \to L)$ is a labeled timed HDA and s is a state of X, called initial state of X. Morphisms of THTS are morphisms of labeled timed HDA which preserve the initial state.*

[13] The untimed part is easy to verify though.

Figure 11.10: A labeled timed HDA



11.4 Examples

11.4.1 Semi-regular HDA as THDA

Proposition 18 *Let M be a semi-regular HDA. Then $\mid M \mid$ is a timed HDA with the family of norms $\|u_1, \ldots, u_n\|_{x_1,\ldots,x_n} = max\{u_1, \ldots, u_n\}$.*

PROOF. Straightforward. □

The norm chosen corresponds to giving to all 1-transitions the unity duration and to give the rule that when n processes run asynchronously the time to complete them is the maximum of the times necessary to complete each of them. This corresponds to our view of independent processes running asynchronously.

The study of geodesics in $\mid M \mid$ reveals the following properties.

Proposition 19 *In $\mid D_{[n]} \mid$ (considered as a timed HDA) there are minimal and maximal length virtual paths from $(0, 0, \ldots, 0)$ to $(1, 1, \ldots, 1)$.*

- *There is one minimal length path γ_0 (the synchronous execution of n actions) up to parametrization given by (for all t)*

$$x_1(\gamma_0(t)) = x_2(\gamma_0(t)) = \ldots = x_n(\gamma_0(t))$$

- *There are infinitely many maximal length paths γ_1 (the interleavings of parts of n actions) given by,*

$$\forall t, \exists i, \forall j \neq i, \dot{x}_j(\gamma'(t)) = 0$$

PROOF. Let γ be a virtual path from $(0,0,\ldots,0)$ to $(1,1,\ldots,1)$ in $\mid D_{[n]} \mid$. Then, if $I_i = \{t \in [0,1]/\dot{x}_1(\gamma'(t)) = \gamma'_i(t) \geq \gamma'_j(t) \forall j \neq i\}$,

$$\begin{aligned} \int_0^1 F(\gamma'(t))dt &= \sum_i \int_{I_i} \gamma'_i(t)dt \\ &= \int_I \gamma'_k(t)dt + \sum_{i \neq k} \int_{I_i} (\gamma'_i(t) - \gamma'_k(t)) \\ &\geq 1 \end{aligned}$$

Moreover, a direct computation shows that the length of γ_0 is equal to one. Finally, if there exists k, $t \in I_k$ and j such that greater than 1.

Now, consider the case of maximal length virtual paths.

$$\begin{aligned} \int_I F(\gamma'(t))dt &= \sum_i \int_{I_i} \gamma'_i(t)dt \\ &\geq \sum_i \int_I \gamma'_i(t)dt \\ &\geq n \end{aligned}$$

Notice that $l(\gamma_1) = \bigcup_i \int_I \gamma'_i(t)dt = n$. □

This entails that in $\mid D_n \mid$,

- there are $n!$ maximal paths, all of dimension one e.g. the interleavings of n actions,
- there is one minimal path, the path of maximal dimension.

11.4.2 Semantics of a toy language

We consider the following language (a subset of RTCCS, [Kri91]),

$$\begin{array}{lllllll} P & ::= & a;P & \mid & P \mid P & \mid & \textbf{nil} \\ \mid & P+P & \mid & P \| P & \mid & (t)P & \mid \quad \textbf{rec}\ x.P[x] \end{array}$$

The atomic actions a are supposed to take unit time. $(t)P$ can behave like P after time t.

Semantic domains

As in Chapter 5 we want to give to terms of the language denotations which are higher-dimensional transition systems. To this end, we want to define a huge THDA D (called *domain* as in [Gou93]) which will contain all possible operational behaviours of terms of the language. Elements of the domain, and thus denotations, will be sub$THTA$ of D (i.e. inclusion morphisms into D) as in the untimed case.

All this is most conveniently done by recursive domain definitions (see [Plo84]). As a matter of fact, we generally want a domain to contain a few specified

actions and to be closed under such constructs as the parallel composition (the tensor product). $\coprod$ (and then amalgamated sums – i.e. pushouts – $+$), $\times$ and $\otimes$ are covariant functors commuting with colimits, therefore standard results [AL91b] guarantee the existence of solutions to recursive equations like $D \cong U + D \otimes D$ (U is a given $THTA$) which is precisely a $THTA$ closed under parallel composition. More complex constructions can be done (similar to the homotopical constructions of [Gou93]), for instance to give domains for imperative languages where states are mappings from variables to actual values but we will not need them here. We will not consider here domains for some functional parallel languages (like CML [Rep92] which would involve equations like $D \cong U + D \otimes D + D \multimap D$).

Denotational semantics

We first construct the domain we need. Here we give a denotational semantics where denotations are operational behaviours[14]. We recall that for CCS we had in Chapter 5 semi-regular HDA (a_i^j), $(\overline{a}_i^j)$, $(\tau_{i,j}^k)$ $(i,j \in \mathbb{N},\ k \in K)$, (a^j), $(\overline{a}^j)$ and (τ) be the following HDA (informally or geometrically),

$$(a_i^j): \quad 1 \xrightarrow{a_i^j} \alpha_i \qquad\qquad (a^j): \ 1 \overset{a^j}{\circlearrowright}$$

$$(\overline{a}_i^j): \quad 1 \xrightarrow{\overline{a}_i^j} (\overline{\alpha}_i^j) \qquad\qquad (\overline{a}^j): \ 1 \overset{\overline{a^j}}{\circlearrowright}$$

$$(\tau_{i,j}^k): \quad 1 \xrightarrow{\tau_{i,j}^k} \alpha_i^k \otimes \overline{\alpha}_j^k \qquad\qquad (\tau): \ 1 \overset{\tau}{\circlearrowright}$$

Let P and K be the domains given by the recursive equations,

$$P \cong (|\ a_i^j\ |)_{i,j} + (|\ \overline{a}_i^j\ |)_{i,j} + (|\ \tau_{i,j}^k\ |)_{i,j,k} + P \otimes P$$

$$K \cong (|\ a^j\ |) + (|\ \overline{a}^j\ |) + (|\ \tau\ |) + K \otimes K$$

They are the timed versions of the domains we had for CCS,

$$D \cong (a_i^j)_{i,j} + (\overline{a}_i^j)_{i,j} + (\tau_{i,j}^k)_{i,j,k} + D \otimes D$$

$$L \cong (a^j) + (\overline{a}^j) + (\tau) + L \otimes L$$

We had also $l : D \to L$ the morphism of HDA defined by,

- $\forall i \in \mathbb{N},\ l(a_i^j) = a^j$
- $\forall i \in \mathbb{N},\ l(\overline{a}_i^j) = \overline{a}^j$

[14] We could have given one in a more classical form like input-ouput relations or history of communications.

- $\forall i,j \in \mathbb{N}$, $l(\tau^k_{i,j}) = \tau$
- $\forall x, y \in P$, $l(x \otimes y) = l(x) \otimes l(y)$

It lifts easily to $k =\mid l \mid : P \to K$

We had introduced as well an operator $\otimes_c$ for dealing with synchronization. $\otimes_c$ lifts to timed HDA.

The domain of HDA in which we give the semantics of the language is $k : P \to K$. We can actually give it in $D = P$, and recover the full definition by applying the labelling l. Then,

- $[\![\mathbf{nil}]\!] = (1)$
- $[\![a^j; P]\!] = (\mid a^j_i \mid) \coprod \left(\alpha^j_i \otimes [\![P]\!]\right)$ for some fresh i
- $[\![\overline{a}^j; P]\!] = (\mid \overline{a}^j_i \mid) \coprod \left(\overline{\alpha}^j_i \otimes [\![P]\!]\right)$ for some fresh i
- $[\![p + q]\!] = [\![p]\!] \coprod [\![q]\!]$ ($\coprod$ is the coproduct in $T\Upsilon/K$, it corresponds to an amalgamated sum in $T\Upsilon$)
- $[\![p \| q]\!] = [\![p]\!] \otimes [\![q]\!]$
- $[\![p \mid q]\!] = [\![p]\!] \otimes_c [\![q]\!]$
- $[\![(t).P]\!] = X_t \coprod (t \otimes [\![P]\!]$, where X_t is defined in Example 51
- $[\![\mathbf{rec}\ x.p[x]]\!] = \overrightarrow{lim}\ [\![p^i[\mathbf{nil}]]\!]$ where the direct limit is taken on the full subcategory of $T\Upsilon/K$ whose objects are the $[\![p^i[\mathbf{nil}]]\!]$

Operational semantics

Its operational semantics is then (by results of Section 11.3),

$$\frac{}{\mathbf{nil} \models 1 \xrightarrow[[0,0]]{1} 1}$$

$$\frac{}{a^j; P \models 1 \xrightarrow[[1,1]]{a^j} \alpha^j_i} \qquad \frac{}{\overline{a}^j; P \models 1 \xrightarrow[[1,1]]{\overline{a}^j} \overline{\alpha}^j_i}$$

$$\frac{P \models s \xrightarrow[[\alpha_1,\alpha_2]]{u} s'}{a^j; P \models \alpha^j_i \otimes s \xrightarrow[[\alpha_1,\alpha_2]]{u} \alpha^j_i \otimes s'}$$

$$\frac{P \models s \xrightarrow[[\alpha_1,\alpha_2]]{u} s'}{\overline{a}; P \models \overline{\alpha}_i \otimes s \xrightarrow[[\alpha_1,\alpha_2]]{u} \overline{\alpha}_i \otimes s'}$$

$$\frac{Q \models s \xrightarrow[[\alpha_1,\alpha_2]]{a} \quad t}{Q + Q' \models s \xrightarrow[[\alpha_1,\alpha_2]]{a} t}$$

$$\frac{Q' \models s' \xrightarrow[[\alpha'_1,\alpha'_2]]{a'} \quad t'}{Q + Q' \models s' \xrightarrow[[\alpha'_1,\alpha'_2]]{a'} t'}$$

$$\frac{Q \models s \xrightarrow[[\alpha_1,\alpha_2]]{a} \quad t \qquad\qquad Q' \models s' \xrightarrow[[\alpha'_1,\alpha'_2]]{a'} t'}{Q \| Q' \models s \otimes s' \xrightarrow[[max(\alpha_1,\alpha'_1),\alpha_2+\alpha'_2]]{a \otimes a'} t \otimes t'}$$

$$\frac{Q \models s \xrightarrow[[\alpha_1,\alpha_2]]{a} \quad t \qquad\qquad Q' \models s' \xrightarrow[[\alpha'_1,\alpha'_2]]{a'} t' \quad a \neq \overline{a'}}{Q \mid Q' \models s \otimes s' \xrightarrow[[max(\alpha_1,\alpha'_1),\alpha_2+\alpha'_2]]{a \otimes a'} t \otimes t'}$$

$$\frac{Q \models s \xrightarrow[[\alpha_1,\alpha_2]]{x \otimes a \otimes y} \quad t \qquad\qquad Q' \models s' \xrightarrow[[\alpha'_1,\alpha'_2]]{z \otimes \overline{a} \otimes t} t'}{Q \mid Q' \models s \otimes s' \xrightarrow[[?,?]]{x \otimes z \otimes \tau \otimes y \otimes t} t \otimes t'}$$

$$\frac{Q[\mathbf{rec}\ x.Q[x]] \models s \xrightarrow[[\alpha_1,\alpha_2]]{u} \quad t}{\mathbf{rec}\ x.Q[x] \models s \xrightarrow[[\alpha_1,\alpha_2]]{u} t}$$

The rule for synchronization has no timing laws since it has to be defined by $\otimes_c$. The last rule expresses that $[\![\mathbf{rec}\ x.Q[x]]\!]$ forms a co-cone with the diagram $([\![Q^i[\mathbf{nil}]]\!])_i$.

11.5 Homology and Homotopy

We end this presentation of timed HDA by the comparison between the "continuous geometry" available in this model and the discrete one we have been extensively studying on (discrete) HDA.

11.5.1 Cubical versus simplicial homology

We recall the elements of comparison in Serre's thesis [Ser51] between the singular cube homology theory and the classical singular one based on simplexes. Let L_n be the standard simplex of dimension n in $\mathbb{R}^{n+1}$, seen here as the set of points $(y_0, \ldots, y_n)$ with $0 \leq y_i \leq 1$ and $\sum_{i=0,\ldots,n} y_i = 1$. Define an application $\theta_n : \square_n \to L_n$ by $\theta_n(x_1, \ldots, x_n) = (y_0, \ldots, y_n)$ and

$$\begin{cases} y_0 &= 1 - x_1 \\ y_1 &= x_1(1 - x_2) \\ \ldots & \\ y_{n-1} &= x_1 x_2 \ldots x_{n-1}(1 - x_n) \\ y_n &= x_1 x_2 \ldots x_{n-1} x_n \end{cases}$$

Now we have the following lemma,

Lemma 39 *Let X be a topological space. The θ_n induce a morphism θ from the R-module of singular chains (classical ones) $C(X)$ to the R-module of singular cubic chains $CC(X)$. θ commutes with the total boundary operator.*

Proposition 20 *$\theta^* : H_*(X) \to H'_*(Tot(X))$ is an isomorphism.*

Therefore, cubical homology (for the total complex) and classical singular homology are equivalent. In particular, the homology of the total complex derived from the semi-regular HDA underlying the structure of a timed HDA is isomorphic to the singular homology of the topological space underlying it.

11.5.2 Homotopy of oriented paths

In this section, we formalize the continuous counterpart of serializability. We restrict our dicussion to the fundamental group.

The fundamental group of oriented paths

Let X be a timed HDA and α, β be points in X. An oriented path from α to β is a continuous function

$$f : I = [0, 1] \to X$$

such that,

(1) $f(0) = \alpha$,

(2) $f(1) = \beta$,

(3) $x_i \circ f$, where x_i is the ith local coordinate in Y, are increasing functions.

Condition (3) means that a path should always execute more as time goes: the local coordinates measure the amount of time spent on each processor and these must increase as the global time goes.

The set of paths from α to β is denoted by $\mathcal{P}_1^{\alpha,\beta}(X)$.

We have a natural operation on paths, that is, the concatenation of paths.

Let $f \in \mathcal{P}_1^{\alpha,\beta}(X)$ and $g \in \mathcal{P}_1^{\beta,\gamma}(X)$ then we define $f.g : I \to X$ by

$$f \cdot g(x) = \begin{cases} f(2x) & \text{if } 0 \leq x \leq 1/2 \\ g(2x-1) & \text{if } 1/2 \leq x \leq 1 \end{cases}$$

It is easy to see that $f \cdot g \in \mathcal{P}_1^{\alpha,\gamma}(X)$. $(f,g) \to f \cdot g$ is not associative nor commutative. It has a neutral element, the constant path $1_\alpha \in P_1^{\alpha,\alpha}(X)$ defined by $1_\alpha(x) = \alpha$ for all $\alpha \in I$.

The homotopy relation will then be defined first for paths with end points fixed and extended in order to behave well with respect to concatenation.

Let $f, g \in \mathcal{P}_1^{\alpha,\beta}(X)$ be two oriented paths in X. A continous function $h : I \times I \to X$ is a homotopy from f to g if and only if,

- for all $y \in I$, $h(0,y) = f(y)$,
- for all $y \in I$, $h(1,y) = g(y)$,
- for all $x \in I$, $y \to h(x,y)$ is an oriented path from α to β in X.

This defines an equivalence relation on $\mathcal{P}_1^{\alpha,\beta}(X)$. We write $f \sim_{\alpha,\beta} g$ if f and g are two homotopic oriented paths in X from α to β.

PROOF. $h(x,y) = f(y)$ defines a homotopy between f and f.

If h is a homotopy from f to g, $h'(x,y) = h(1-x,y)$ defines a homotopy from g to f.

Finally, if h_1 is a homotopy from e to f and h_2 is a homotopy from f to g then,

$$h(x,y) = \begin{cases} h_1(2x,y) & \text{if } 0 \leq x \leq 1/2 \\ h_2(2x-1,y) & \text{if } 1/2 \leq 1 \end{cases}$$

defines a homotopy from e to g. □

Let $\Pi_1^{\alpha,\beta}(X)$ be the free R-module generated by the equivalence classes of oriented paths from α to β.

The homotopy relation behaves well with respect to concatenation. In particular, if $f, f' \in \mathcal{P}_1^{\alpha,\beta}(X)$, $g, g' \in \mathcal{P}_1^{\beta,\gamma}(X)$ such that $f \sim_{\alpha,\beta} f'$ and $g \sim_{\beta,\gamma} g'$ then $f.g \sim_{\alpha,\gamma} f'.g'$. Therefore $(f,g) \to f.g$ defines an operation, still written $(f,g) \to f.g$ on equivalence classes of oriented paths.

Now $. : \mathcal{P}_1^{\alpha,\beta}(X)/\sim_{\alpha,\beta} \times \mathcal{P}_1^{\beta,\gamma}(X)/\sim_{\beta,\gamma} \to \mathcal{P}_1^{\alpha,\gamma}(X)/\sim_{\alpha,\gamma}$ is associative.

(*) PROOF. (see [Mas91]) Let $f \in \mathcal{P}_1^{\alpha,\beta}(X)$, $g \in \mathcal{P}_1^{\beta,\gamma}(X)$ and $h \in \mathcal{P}_1^{\gamma,\delta}(X)$. Let $F : I \times I \to X$ defined by,

$$F(s,t) = \begin{cases} f\left(\frac{4t}{1+s}\right) & \text{if } 0 \leq t \leq \frac{s+1}{4} \\ g(4t-1-s) & \text{if } \frac{s+1}{4} \leq t \leq \frac{s+2}{4} \\ h\left(1 - \frac{4(1-t)}{2-s}\right) & \text{if } \frac{s+2}{4} \leq t \leq 1 \end{cases}$$

then F is continuous (because f, g and h are) and,

- for each s, $t \to F(s,t)$ is an oriented path in X,
- for all $t \in I$, $F(0,t) = [(f.g).h](t)$,
- for all $t \in I$, $F(1,t) = [f.(g.h)](t)$.

thus F is a homotopy between $(f.g).h$ and $f.(g.h)$ and $([f].[g]).[h] = [f].([g].[h])$. □

$(f,g) \to f \cdot g$ extends naturally to $\cdot : \Pi_1^{\alpha,\beta}(X) \times \Pi_1^{\beta,\gamma}(X) \to \Pi_1^{\alpha,\gamma}(X)$ by the distributive law,

$$(\lambda[x] + \mu[y]).[z] = \lambda[x].[z] + \mu[y].[z]$$

therefore $(f,g) \to f.g$ is bilinear.

We are now ready define the homotopy group $\Pi_1(X)$, the group of oriented paths (from anywhere to anywhere) modulo homotopy. The monoid operation on paths is only partially defined since two paths have to be composable in order to have their concatenation defined. A way to extend this operation is to think as the concatenation of two non-composable paths ($[p] \in \Pi_1^{\alpha,\beta}(X)$ and $[p'] \in \Pi_1^{\alpha',\beta'}(X)$ with $\beta \neq \alpha'$) as their union, i.e. their sum in the free modules $\Pi_1^{\alpha,\beta}(X) \oplus \Pi_1^{\alpha',\beta'}(X)$. The addition of two composable paths should also be equated to their concatenation.

This means that we want to make the identification

$$[x] + [y] \cong [x] \cdot [y]$$

whenever x and y are composable.

Therefore, we set,

$$\Pi_1(X) = \left(\bigoplus_{\alpha,\beta \in X} \Pi_1^{\alpha,\beta}(X) \right) / \cong$$

Relationship with the standard fundamental group

The standard fundamental group is defined in a similar manner, but the group structure, induced by the concatenation of closed paths (or loops) is more natural in some way. We recall the construction below.

Let $x \in X$. Then the closed paths (or loops) based at x are the continuous functions $f : I \to X$ such that $f(0) = f(1) = x$. A homotopy between two loops f and g is a continuous function $H : I \times I \to X$ such that,

- for all $y \in I$, $H(0,y) = f(y)$,
- for all $y \in I$, $H(1,y) = g(y)$,
- for all $x \in I$, $H(x,0) = H(x,1) = x$.

Two remarks here.

First, the concatenation of paths is well defined for all loops based at x, which is a simpler case than for non-closed paths which were not all composable. Therefore, modulo homotopy we have a natural monoid structure.

Secondly, we impose no "orientation of time" on paths. Therefore the inverse of a path f, defined as $f' : I \to X$ with for all x, $f'(x) = f(1-x)$, is also a path. This implies that we have a natural group structure which we had to artificially construct in the oriented path case.

The fundamental group $\Pi(X,x)$ is then defined to be the group of all loops modulo homotopy, with concatenation as the group law. It can be shown [Spa66] that if $x \in X$ and $y \in X$ are in the same connected component, then $\Pi(X,x)$ and $\Pi(X,y)$ are isomorphic groups. In the following, we will suppose that X is connected, therefore we can write $\Pi(X)$ for any of the $\Pi(X,x)$. We suppose also $R = \mathbb{Z}$, then,

Proposition 21 *The map* $u : \Pi_1^{\alpha,\beta}(X) \times \Pi_1^{\alpha,\beta}(X) \to \Pi(X)$ *defined by*

$$u([x],[y]) = [x - y]$$

is a monomorphism of groups.

Relationship with the untimed case

Claim 3 *Let M be a timed HDA, $\alpha, \beta \in X_0$. Then,*

$$\Pi_1^{\alpha,\beta}(M) \cong \Pi_1^{\alpha,\beta}(\mathcal{F}t(M))$$

Summary In this chapter we have used the natural geometric representation of semi-regular HDA as a basis for a truly-concurrent operational model for real-time, where time is measured as the length of paths. Basically, a timed HDA was defined as a shape together with an observational structure given by a semi-regular HDA realized on this shape, and a family of norms defining the infinitesimal cost of computation in all directions.

We have shown that Zeno behaviours could be expressed without having an incoherent model, that we could express some fairness properties and relate (by abstract interpretation) the discrete observational structure (untimed) to the timed HDA. This proved that timed HDA is an extension of our previous models indeed. Finally we proved that some categorical constructs correspond to process algebraic operators with nice timing laws. In particular, we had a tensor product describing the parallel composition with no interference for which the time taken by a system of two processes is the maximum of the times taken by each of the processes.

We ended by giving a few hints about how to extend the homotopy theory we developed in the untimed case. We do not know yet what the "continuous" homotopy theory for oriented paths looks like.

Future work

We have presented in this thesis a geometric theory of concurrent machines. It raises some mathematical problems: some algebraic problems about "weak" bicomplexes, a homotopy theory, etc. We have only presented a fomal basis for solving these. Some work remains to be done. In particular:

- Higher-order homotopy groups are not fully described yet. In the more general context of combinatorial HDA we do not know yet how they relate to the ones defined for free general HDA generated by acyclic semi-regular HDA.
- We do not yet have any good "continuous" counter-part to this discrete homotopy theory, which would be useful for scheduling problems of real-time systems. As a matter of fact, we would like to see a generalization of timed HDA so that at least we could relax the condition on $\{x(\overset{\circ}{\square}_n)\}$ to partition a timed HDA X. The condition: "the set $E = \{x(\overset{\circ}{\square}_n)\}$ verifies the property $\forall x, y \in E$, $x \cap y = \emptyset$ or $x \subseteq y$ or $y \subseteq x$" would allow definitions of tangent spaces, norms, etc. and would authorize better definitions of the timeout operator (as all partial executions of a transition of execution time t would be geometrically included in it).

On the more computer-scientific side we could think of a number of problems, for instance:

- We have not gone too deep into the study of infinite paths. This could be tackled by using an extension of SOS similar to G^{∞}SOS.
- We would like to carry on some work on verifying "real" protocols for distributed systems. This might need a good implementation of a version of the verification algorithm we gave in Chapter 9. We have recently begun an implementation in **C**.

We have already carried out some work on logics for HDA (from the categorical structure, Hoare-like ones, or even from an interaction categories point of view) and on a probabilistic extension of HDA. There should be future work to complete them.

Appendix A

Mathematical background - Rings, modules and complexes

Let R be a commutative ring (integral and unitary).

We recall the following definitions, taken from [Lan93a].

A left-module (or simply a module) M over R is an abelian group (written additively) together with an operation of R on M (multiplication), such that, $\forall a, b \in R, \forall x, y \in M$, $(a+b)x = ax + bx$ and $a(x+y) = ax + ay$.
In many cases, when the context makes it clear, we will say module for module over R, or R-module.
Module homomorphisms (or linear functions) are functions which preserve operations plus and multiplication by an element of R, and element 0 (neutral for $+$).
Let M be a module over R, and S a subset of M. Then S is a basis of M is S is not empty, if all elements of M are obtained as linear combinations of elements of S (S generates M), and if 0 cannot be obtained as a non-null linear combination of elements of S (S is free).
M is a free module if and only if it has a basis. We write (a) for the free module generated by a. We write $Mod(S)$ or $R - Mod(S)$ for the module generated by a set S.
Note that if R is a field, a R-module is a vector space, thus is free in the sense that all vector spaces admit a base.
If M_1 and M_2 are two free modules, M_1 and M_2 generated respectively by S and T, then we write $M_1 + M_2$ or $\sum_i M_i$ for the free module M generated by $S \cup T$. When $S \cap T = \emptyset$ we write $M = M_1 \oplus M_2 = \bigoplus_i M_i$.
We list below some properties of interest about freeness of modules which are used implicitly in most of the proofs of, for instance Section 6.4 and Chapter 8. We refer the reader to [Lan93a] for proofs of these facts.

First of all, for any ring R,

- any R-module is isomorphic to a quotient of a free R-module (this fact leads to the notion of free resolutions, used in Chapter 8),
- if A' is a sub-module of A such that A'/A is free, then A is isomorphic to $A' \oplus (A/A')$.

When R is a principal ideal domain, i.e. when R is an integral domain in which every ideal is principal (like $\mathbb{Z}$, all ideals are of the form $n.\mathbb{Z}$, $n \in \mathbb{N}$), we can be much more precise about the structure of R-modules in general. In this case, any submodule of a free R-module is free. The only part of a module which needs be classified is therefore the non-free part. This is done through the notion of torsion module. If A is a R-module, its torsion sub-module is $TorA = \{a \in A / ra = 0 \text{ for some } r \in R,\ r \neq 0\}$. Then a finitely generated R-module A is free if and only if it is torsion free (i.e. $TorA = 0$). It is easy to see that $A/TorA$ is free. A cyclic module corresponding to some element $r \in R$ is a R-module A such that r generates the ideal of all elements z annihilating every element of A, i.e. $\forall a \in A$, $za = 0$ implies $\exists y \in R$ such that $z = yr$. This leads to the following theorem, known as the structure theorem for finitely generated modules.

Theorem 6 *Any R-module A is isomorphic to the direct sum of a free R-module and cyclic modules $A_1, \ldots, A_q$ whose corresponding elements $r_1, \ldots, r_q \in R$ are such that $r_i \mid r_{i+1}$ for all $1 \leq i \leq q-1$. The elements $r_1, \ldots, r_q$ are unique up to multiplication by invertible elements of R and together with $rank(A) = dim\,(A/TorA)$ characterize the module up to isomorphism.*

In all this text we note (a) for the free R-module generated by a and $(a)_r$ for the cyclic module generated by a with corresponding element $r \in R$.

Other types of modules than free modules are of interest as well.

A module M is a projective module if given an epimorphism $\sigma : B \longrightarrow C$, each map $\gamma : M \longrightarrow C$ can be lifted to a $\beta : M \longrightarrow B$ such that $\sigma\beta = \gamma$. Note that every free module is projective, and that every projective $\mathbb{Z}$-module is free.
Dually, a module M is an injective module if for each $\alpha : A \longrightarrow M$ and for any monomorphism $\kappa : A \longrightarrow B$, there exists $\beta : B \longrightarrow M$ such that $\beta\kappa = \alpha$. Note that if R is a field, then any R-module (i.e. vector space) is injective. A $\mathbb{Z}$-module is injective if and only if it is divisible, i.e. for each integer $m \neq 0$ and each element d, there is a solution of the equation $mx = d$.
A differential module (M, d) (or simply M) is a module M together with a grading $M = \bigoplus_i M_i$, a function $d : M \longrightarrow M$ such that for all i, $d(M_{i+1}) \subseteq M_i$[1] and such that $d \circ d = 0$.
A complex of modules is a sequence of modules M_i together with a function $d : M \longrightarrow M$ such that for all i $d(M_{i+1}) \subseteq M_i$ and such that $d \circ d = 0$.

[1] It is of homology type.

Appendix B

Mathematical Background - Some basic properties of simplicial complexes

For full details on the definitions and properties, we refer the reader to [May67] and [GZ67].

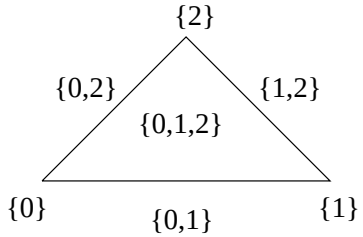
A simplicial complex is, geometrically, a union of points, segments, triangles, tetrahedra etc. Formally, a simplicial complex K is a set of simplices, that are finite subsets of a given set $\overline{K}$ of points, subject to the condition that every non-empty subset of an element of K is itself an element of K. Look at Figure B.1 for an example: the triangle is the simplex of dimension 2 $\{0,1,2\}$. The condition on subsets of simplices to be simplices as well enforces all segments $\{0,1\}$, $\{0,2\}$ and $\{1,2\}$ to be in the simplicial complex, as well as all points $\{0\}$, $\{1\}$ and $\{2\}$.

Ordering vertices, we get an equivalent definition of simplicial complexes as follows. A n-simplex (or simplex of dimension n, forming the set K_n) is a sequence $(a_0,\ldots,a_n)$ of elements of $\overline{K}$ such that $\{a_0,\ldots,a_n\}$ is a m-simplex of K for some $m \leq n$. If m is strictly less than n then we say that the n-simplex $(a_0,\ldots,a_n)$ is degenerate.

Being a face of some n-simplex $(a_0,\ldots,a_n)$ means forgetting one of the components of $(a_0,\ldots,a_n)$. This defines face operators $(0 \leq i \leq n)$,

$$\partial_i(a_0,\ldots,a_n) = (a_0,\ldots,a_{i-1},a_{i+1},\ldots,a_n)$$

Figure B.1: A filled-in triangle seen as a simplicial complex



The fact that we can consider $\{a_0, \ldots, a_n\}$ as degenerate m-simplexes for $m > n$ is described by degeneracy operators ($0 \leq i \leq n$),

$$s_i(a_0, \ldots, a_n) = (a_0, \ldots, a_i, a_i, a_{i+1}, \ldots, a_n)$$

These operators can be shown to verify the following commutation rules,

$$\begin{cases} \partial_i \partial_j = \partial_{j-1} \partial_i & \text{if } i < j \\ s_i s_j = s_{j+1} s_i & \text{if } i \leq j \\ \partial_i s_j = s_{j-1} \partial_i & \text{if } i < j \\ \partial_j s_j = Id = \partial_{j+1} s_j & \\ \partial_i s_j = s_j \partial_{i-1} & \text{if } i > j+1 \end{cases}$$

Now a morphism from a simplicial complex K to a simplicial complex L is a map carrying each vertex of K to a vertex of L. This induces a map from each simplex of K to a simplex of L (as unions of the image of each vertex in the simplex).

With the other formulation we have, morphisms are just functions $f = (f_n)_n$ of graded sets, $f_n : K_n \to L_n$ such that,

$$f_n \partial_i = \partial_i f_{n+1}$$

$$f_n s_i = s_i f_{n-1}$$

In fact we could have defined the category of simplicial complexes (or simplicial sets) as contravariant functors from Δ, the category whose objects are the linear orders $\Delta_n = \{0 < 1 < \ldots < n\}$ and whose morphisms are monotonic maps to **Set**. As a matter of fact, the dual commutation relations that the ∂_i and s_j verify precisely generate all morphisms in Δ. The reader can check that the natural transformations in Δ^{op}**Set** coincide with what we called morphisms of simplicial complexes.

With this formulation, the standard n-simplexes, i.e. the n-triangles (triangle for $n = 2$, tetrahedron for $n = 3$ etc.) are the representable functors $Hom(\cdot, \Delta_n)$.

This view of simplicial sets enables us to generalize furthermore and define simplicial objects in any category $\mathcal{C}$ as the category of contravariant functors from Δ to $\mathcal{C}$.

An interesting case is simplicial objects in $R - Mod$. They are called simplicial modules. It can be shown that there is an equivalence of categories between the category of simplicial modules and the category of complexes of modules [May67, GZ67]. An interesting property of simplicial modules is also that they are Kan complexes and then a homotopy theory can be defined in easy ways.

Appendix C

Mathematical Background - Some basic concepts of differential geometry

Manifolds are natural generalizations of curves and surfaces in that they can describe geometric objects of higher dimension and support a differential calculus, i.e. a calculus on tangent vectors.

A manifold[1] is given by the following data (taken from [Die74]).

An atlas A of some topological space X is a covering of X by opens U_i $(i \in I)$ such that,

- with each U_i comes a homeomorphism $\phi_i : U_i \to V_i$ where V_i is an open of the topological space $\mathbb{R}^n$ (with its standard topology),
- $\phi_j \circ \phi_i^{-1} : \phi_i(U_i \cap U_j) \to \phi_j(U_i \cap U_j)$ is a C^∞ diffeomorphism (standard notion in $\mathbb{R}^n$)

(U_i, ϕ_i) is called a chart of the atlas.

Then a differentiable manifold is a topological space X together with such an atlas.

The maps $x^k = pr_k \circ \phi_i : U_i \to \mathbb{R}$, where $pr_k : \mathbb{R}^n \to \mathbb{R}$ is the kth projection, are called the local coordinates in U_i (for the chart (U_i, ϕ_i)). The dimension of X at $x \in U_i$ is the least integer n such that $\phi_i : U_i \to \mathbb{R}^n$. It is constant in all connected components of X. Locally (i.e. in some U_i), the manifold "looks like" an open subset of $\mathbb{R}^n$ (look at Figure C.1).

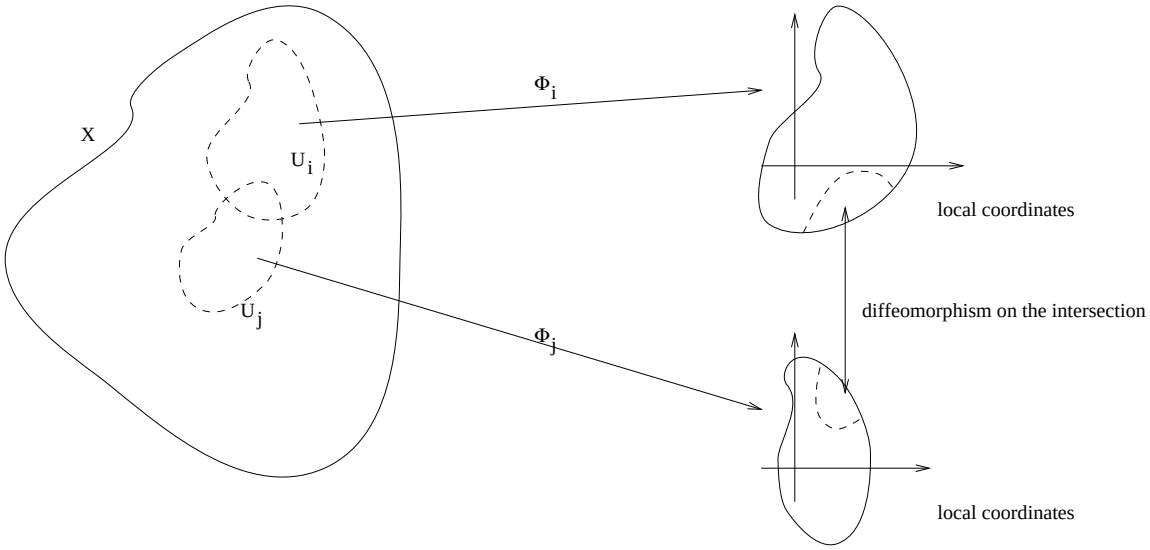
The local coordinates can be used as ordinary coordinates in Euclidean geometry. The usual notions (on surfaces for instance), differentiable functions, tangent space, sub-manifold, etc. can then be defined easily.

Let X and Y be two manifolds. A continuous function $f : X \to Y$ is differentiable if for all charts (U, ϕ) in X and (V, ψ) in Y such that $f(U) \subseteq V$,

$$F = \psi \circ f_{|U} \circ \phi^{-1} : \phi(U) \to \psi(V)$$

[1] Here we only consider C^∞ manifolds and C^∞ differentiable functions.

Figure C.1: A manifold and its atlas



is differentiable[2]. The differential of f is denoted by df.

Now, let $x \in X$ and f_1, f_2 be two differentiable functions defined in a neighbourhood W of x and with value in Y. We say that f_1 and f_2 are tangent at x if,

- $f_1(x) = f_2(x)$,
- for a chart (U, ϕ) in X with $U \subseteq W$ and a chart (V, ψ) in Y such that $f_1(U) \subseteq V$ and $f_2(U) \subseteq V$ the functions $\psi \circ f_{1|U} \circ \phi^{-1}$ and $\psi \circ f_{2|U} \circ \phi^{-1}$ have the same derivative at point $\phi(x)$.

It defines an equivalence relation on differentiable functions from X to Y.

Particularizing this to $X = \mathbb{R}$, differentiable functions from X to Y modulo the tangency relation are called the tangent vectors of Y at point y. They actually form a real vector space of dimension n (the same dimension as the manifold) called $T_y(Y)$ as follows. We define a bijection $\theta_\phi : T_y(Y) \to \mathbb{R}^n$ which associates the vector $d(\phi \circ f)(0)$ with an equivalence class of a differentiable function $f : W \to Y$, where W is an open neighbourhood of 0 in $\mathbb{R}$, such that $f(0) = y$.

Let $(e_1, \ldots, e_n)$ be the standard basis of $\mathbb{R}^n$, i.e. $e_1 = (1, 0, \ldots, 0)$,...,$e_n = (0, \ldots, 1)$. $(\dot{x}_1 = \theta_\phi^{-1}(e_1), \ldots, \dot{x}_n = \theta_\phi^{-1}(e_n))$ is the basis of the $\mathbb{R}$-vector space $T_y(Y)$ associated with the chart (U, ϕ) (with local coordinates $(x_1, \ldots, x_n)$).

Let X and Y are two manifolds and $f : X \to Y$ is a differentiable function. If $x \in X$, $y = f(x) \in Y$ and (U, ϕ) is a chart of X at x (respectively (V, ψ) is a chart of Y at y, the linear function,

$$T_x(f) = \theta_\psi^{-1} \circ F'(\phi(x)) \circ \theta_\phi : T_x(X) \to T_y(Y)$$

[2]This notion is the usual one in $\mathbb{R}^n$.

where

$$F = \psi \circ f_{|U} \circ \phi^{-1}$$

is the local expression of f in charts (U, ϕ) and (V, ψ) is the differential of f at x. In general we write $df(x).\dot{x}$ or $df(x, \dot{x})$ for $T_x(f)(\dot{x}_1, \ldots, \dot{x}_n)$.

The last notation is justified by the fact that the T_xX actually define by amalgamation a manifold T_X called the tangent manifold whose local coordinates are of the form $(x_1, \ldots, x_n, \dot{x}_1, \ldots, \dot{x}_n)$ if $(x_1, \ldots, x_n)$ are the local coordinates in the chart (U, ϕ) of X and $(\dot{x}_1, \ldots, \dot{x}_n)$ are the corresponding local coordinates of T_xX. TX is a fiber bundle over X.

Now the usual laws of differential calculus that one has on Euclidean spaces $\mathbb{R}^n$ hold on general manifolds.

Bibliography

[ABND+90] H. Attiya, A. Bar-Noy, D. Dolev, D. Peleg, and R. Reischuk. Renaming in an asynchronous environment. *Journal of the ACM*, 37(3):524–548, July 1990.

[Abr93a] S. Abramsky. Interaction categories. Technical report, Imperial College, London, March 1993.

[Abr93b] S. Abramsky. Interaction categories 2. Technical report, Imperial College, London, August 1993.

[AD91] R. Alur and D. Dill. The theory of timed automata. In *Proceedings of the REX Workshop, Real-Time: Theory in Practice*, LNCS. Springer-Verlag, 1991.

[AF92] H. Attiya and R. Friedman. A correctness condition for high-performance multiprocessors. In *Proc. of the 24th STOC*. ACM Press, 1992.

[AL91a] M. Abadi and L. Lamport. An old-fashioned recipe for real-time systems. Technical report, DEC, 1991.

[AL91b] A. Asperti and G. Longo. *Categories, types and structures*. The MIT Press, second edition, 1991.

[AM93] S. Abramsky and T. Maibaum. *Handbook of Logic in Computer Science*, volume 1. Oxford Press, 1993.

[Bar84] H.P. Barendregt. The lambda calculus. In *Studies in Logic and the Foundations of Mathematics*, volume 103. North-Holland, 1984.

[Bar85] M. Barr. **-autonomous categories*. Springer-Verlag, 1985.

[Bau89] H. J. Baues. Algebraic homotopy. In *Cambridge Studies in Advanced Mathematics*, volume 15. Cambridge University Press, 1989.

[BB90] J. C. M. Baeten and J. A. Bergstra. Real time process algebra. Technical Report CS-R9053, Centre for Mathematics and Computer Science, 1990.

[BC82] G. Berry and P.-L. Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science*, 1982.

[BC85] G. Berry and L. Cosserat. The ESTEREL synchronous programming language and its mathematical semantics. In *Proc. CMU Seminar on Concurrency*, number 197 in LNCS. Springer-Verlag, 1985.

[BDK94] F. Bracho, M. Droste, and D. Kuske. Representation of computations in concurrent automata by dependence orders. Technical report, Technische Universitat Dresden, 1994.

[Bed88] M. A. Bednarczyk. *Categories of asynchronous systems.* PhD thesis, University of Sussex, 1988.

[Ber79] G. Berry. *Modèles complètement adéquats et stables des lambda-calculs typés.* PhD thesis, Université Paris VII, 1979.

[BG93] E. Borowsky and E. Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *Proc. of the 25th STOC.* ACM Press, 1993.

[BH81a] R. Brown and P. J. Higgins. Colimit theorems for relative homotopy groups. *Journal of Pure and Applied Algebra*, (22):11–41, 1981.

[BH81b] R. Brown and P. J. Higgins. On the algebra of cubes. *Journal of Pure and Applied Algebra*, (21):233–260, 1981.

[BHR84] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 7:560–599, 1984.

[BK84] J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60:109–137, 1984.

[BKP86] H. Barringer, R. Kuiper, and A. Pnueli. A really abstract concurrent model and its temporal logic. In *Proc. of the 13th POPL*, pages 173–183. ACM Press, 1986.

[BL91] T. Bolognesi and F. Lucidi. LOTOS-like process algebra with urgent or timed interactions. In *Proc. of REX Workshop, Real-time: Theory in Practice*, LNCS. Springer-Verlag, 1991.

[BMZ88] O. Biran, S. Moran, and S. Zaks. A combinatorial characterization of the distributed tasks which are solvable in the presence of one faulty processor. In *Proc. 7th Annual ACM Symposium on Principles of Distributed Computing*, pages 263–275. ACM Press, August 1988.

[BRG87] W. Brauer, W. Reisig, and Rozenberg G. *Petri Nets. Part I: central models and their properties*. Number 254 in LNCS. Springer-Verlag, 1987.

[Car86] A. Carboni. Bicategories of partial maps. *Cahiers Top. Geom. Diff.*, 1986.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximations of fixed points. *Principles of Programming Languages 4*, pages 238–252, 1977.

[CC92a] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, August 1992.

[CC92b] P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretation. In *Conference Record of the 19th ACM Symposium on Principles of Programming Languages*, pages 83–94, Albuquerque, New Mexico, 1992. ACM Press, New York, U.S.A.

[CC94] P. Cousot and R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and PER analysis of functional languages). In *Proceedings of the 1994 International Conference on Computer Languages*, pages 95–112. IEEE Computer Society Press, May 1994.

[CE56] H. Cartan and S. Eilenberg. *Homological Algebra*. Princeton University Press, 1956.

[CG93] R. Cridlig and E. Goubault. Semantics and analyses of Linda-based languages. In *Proc. of WSA'93*, number 724 in LNCS. Springer-Verlag, 1993.

[Cha90] S. Chaudhuri. Agreement is harder than consensus: set consensus problems in totally asynchronous systems. In *Proc. of the 9th Annual ACM Symposium on Principles of Distributed Computing*, pages 311–334. ACM Press, August 1990.

[CHPP87] P. Caspi, N. Halbwachs, D. Pilaud, and J. Plaice. LUSTRE: a declarative language for programming synchronous systems. In *Proc. of POPL'87*. ACM Press, 1987.

[Cri95] R. Cridlig. Semantic analysis of shared-memory concurrent languages using abstract model-checking. In *Proc. of PEPM'95*, La Jolla, June 1995. ACM Press.

[CRJ87] S. D. Carson and P. F. Reynolds Jr. The geometry of semaphore programs. *ACM Transactions on Programming Languages and Systems*, 9(1):25–53, January 1987.

[Cur86] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming*. Pitman, 1986.

[CZ91] R. Cleaveland and A. Zwarico. A theory of testing for real-time. In *Proceedings of LICS*. IEEE Press, 1991.

[Dav58] M. Davis. *Computability and unsolvability*. McGraw-Hill, 1958.

[DCDMPS83] F. De Cindio, G. De Michelis, L. Pomello, and C. Simone. Milner's communicating systems and Petri nets. In *Selected Papers from the 3rd European Workshop on Applications and Theory of Petri Nets*. Informatik-Fachberichte 66, Springer Verlag, 1983.

[DDNM85] P. Degano, R. De Nicola, and U. Montanari. Partial ordering derivations for CCS. In *Proc. of the 5th FOCS*. LNCS 199, Springer Verlag, 1985.

[DDNM88] P. Degano, R. De Nicola, and U. Montanari. A distributed operational semantics for CCS based on condition/event systems. *Acta Informatica*, 26(1/2):59–91, 1988.

[Die74] J. Dieudonné. *Eléments d'analyse 3*. Gauthier-Villars, 1974.

[Dij68] E.W. Dijkstra. *Cooperating Sequential Processes*. Academic Press, 1968.

[DM87] P. Degano and U. Montanari. Concurrent histories: a basis for observing distributed systems. *Computer systems science*, 34:422–461, 1987.

[DNH83] P. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1983.

[DS89] J. Davies and S. Schneider. An introduction to Timed CSP. Technical Report PRG-75, Oxford University Computing Laboratory, August 1989.

[FLP85] M. Fisher, N. A. Lynch, and M. S. Paterson. Impossibility of distributed commit with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[FS90] P. J. Freyd and A. Scedrov. Categories, allegories. In *North-Holland Mathematical Library*, volume 39. North-Holland, 1990.

[Gir87] J.Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[GJ92] E. Goubault and T. P. Jensen. Homology of higher-dimensional automata. In *Proc. of CONCUR'92*, Stonybrook, New York, August 1992. Springer-Verlag.

[GL92] E. Goubault and C. Lebris. Groupes Ext^1 et déformations de représentations de certaines algèbres de Lie nilpotentes. *Bulletin des Sciences Mathématiques*, 116(4):465–486, 1992.

[GM84] U. Goltz and A. Mycroft. On the relationship of CCS and Petri nets. In *Proc. of the 11th ICALP*. LNCS 172, Springer Verlag, 1984.

[Gou93] E. Goubault. Domains of higher-dimensional automata. In *Proc. of CONCUR'93*, Hildesheim, August 1993. Springer-Verlag.

[Gou95] E. Goubault. Schedulers as abstract interpretations of HDA. In *Proc. of PEPM'95*, La Jolla, June 1995. ACM Press.

[Gra90] P. Granger. *Analyse de conguences*. PhD thesis, Ecole Polytechnique, 1990.

[Gro91] J. R. J. Groves. Rewriting systems and homology of groups. In L. G. Kovacs, editor, *Groups – Canberra 1989*, number 1456, pages 114–141. Lecture notes in Mathematics, Springer-Verlag, 1991.

[GS90] C.A. Gunther and D.S. Scott. Semantic domains. In *Handbook of Theoretical Computer Science*. Elsevier, 1990.

[Gun92] J. Gunawardena. Causal automata. *Theoretical Computer Science*, (101):265–288, 1992.

[Gun94] J. Gunawardena. Homotopy and concurrency. In *Bulletin of the EATCS*, number 54, pages 184–193, October 1994.

[Gup94] V. Gupta. *Chu spaces*. PhD thesis, Stanford University, 1994.

[GZ67] P. Gabriel and M. Zisman. Calculus of fractions and homotopy theory. In *Ergebnisse der Mathematik und ihrer Grenzgebiete*, volume 35. Springer Verlag, 1967.

[Har87] D. Harel. StateCharts: a visual approach to complex systems. *Science of Computer Programming*, 8(3):231–275, 1987.

[Hav94] K. Havelund. *The Fork Calculus*. PhD thesis, Aarhus University, 1994.

[Hen91] T.A. Henzinger. *The Temporal Specification and Verification of Real-time Systems*. PhD thesis, Stanford University, 1991.

[Her94] M. Herlihy. A tutorial on algebraic topology and distributed computation. Technical report, presented at UCLA, 1994.

[HLS72] J.R. Hindley, B. Lercher, and J.P. Seldin. *Introduction to Combinatory Logic*. Cambridge University Press, 1972.

[HMC94] E. Harcourt, J. Mauney, and T. Cook. From processor timing specifications to static instruction scheduling. In *Proc. of the Static Analysis Symposium'94*, LNCS. Springer-Verlag, 1994.

[HMP93] T.A. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In *Proc. of the REX Workshop, Real-Time: Theory in Practice*, LNCS. Springer-Verlag, 1993.

[Hoa81] C.A.R. Hoare. A model for communicating sequential processes. Technical Report PRG-22, Programming Research Group, University of Oxford Computing Laboratory, 1981.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[HR91] M. Hennessy and T. Regan. A process algebra for timed systems. Technical Report 5/91, University of Sussex, 1991.

[HR94] M. Herlihy and S. Rajsbaum. Set consensus using arbitrary objects. In *Proc. of the 13th Annual ACM Symposium on Principles of Distributed Computing.* ACM Press, August 1994.

[HR95] M. Herlihy and S. Rajsbaum. Algebraic topology and distributed computing, a primer. Technical report, Brown University, 1995.

[HS93] M. Herlihy and N. Shavit. The asynchronous computability theorem for t-resilient tasks. In *Proc. of the 25th STOC.* ACM Press, 1993.

[HS94] M. Herlihy and N. Shavit. A simple constructive computability theorem for wait-free computation. In *Proceedings of STOC'94.* ACM Press, 1994.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison–Wesley, 1979.

[HW60] P.J. Hilton and S. Wylie. *Homology theory: an introduction to algebraic topology.* Cambridge University Press, 1960.

[Jos89] M. Joseph. Time and real-time in programs. In *Proc. of FST-TCS'89*, number 405 in LNCS. Springer-Verlag, 1989.

[Kah74] G. Kahn. The semantics of a simple language for parallel programming. *Information Processing*, (74), 1974.

[Kel76] R.M. Keller. Formal verification of parallel programs. *Communications of the ACM*, 7(19):371–384, 1976.

[KM77] G. Kahn and D. B. MacQueen. Coroutines and networks of parallel processes. *Information Processing*, (77), 1977.

[Kri91] P. Krishnan. A model for real-time systems. In *Proc. of Mathematical Foundations of Computer Science*, number 520 in LNCS. Springer-Verlag, 1991.

[L78] J.-J Lévy. *Réductions Correctes et Optimales dans le Lambda-Calcul.* PhD thesis, Université Paris VII, 1978.

[Lan93a] S. Lang. *Algebra.* Addison-Wesley, third edition, 1993.

[Lan93b] E. Lanzmann. Automates d'ordre supérieur. Master's thesis, Université d'Orsay, 1993.

[LP90] Y. Lafont and A. Prouté. Church-Rosser property and homology of monoids. Technical report, Ecole Normale Supérieure, 1990.

[LS86] J. Lambek and P.J. Scott. Introduction to higher-order categorical logic. In *Cambridge studies in advanced mathematics*, volume 7. Cambridge University Press, 1986.

[LW69] A.T. Lundell and S. Weingram. *The Topology of CW-Complexes.* Van Nostrand Reinhold Company, 1969.

[LZ93] S.T. Leung and J. Zahorjan. Improving the performance of runtime parallelization. In *ACM Sigplan Symposium on Principles and Practice of Parallel Programming*, May 1993.

[Mas69] J. L. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, IT-15(1), January 1969.

[Mas78] W. S. Massey. Homology and cohomology theory. Number 46 in Monographs and Textbooks in Pure and Applied Mathematics. Marcel DEKKER, INC., 1978.

[Mas91] W. S. Massey. A basic course in algebraic topology. In *Graduate Texts in Mathematics*, volume 127. Springer-Verlag, 1991.

[May67] J. P. May. *Simplicial objects in algebraic topology.* D. van Nostrand Company, inc, 1967.

[Maz88] A. Mazurkiewicz. Basic notions of trace theory. In *Lecture notes for the REX summer school in temporal logic.* Springer-Verlag, 1988.

[MC85] J. Mc Cleary. *User's guide to spectral sequences.* Publish or perish, Mathematics lecture series 12, 1985.

[Mil80] R. Milner. *Calculus of Communicating System.* Number 92 in LNCS. Springer-Verlag, 1980.

[Mil83] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, (25):267–310, 1983.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[Mil91] R. Milner. The polyadic π-calculus: a tutorial. In *Proc. of the International Summer School on Logic and Algebra of Specification.* Springer-Verlag, 1991.

[ML63] S. Mac Lane. Homology. In *Die Grundlehren der Mathematischen Wissenschaften in Einzeldarstellungen*, volume 114. Springer Verlag, 1963.

[ML71] S. Mac Lane. *Categories for the working mathematician.* Springer-Verlag, 1971.

[MM92] S. Maclane and I. Moerdijk. *Sheaves in Geometry and Logic.* Springer-Verlag, 1992.

[Mog86] E. Moggi. Categories of partial maps and λ_p-calculus. In *Category Theory and Computer Programming*, pages 242–251. Springer-Verlag, 1986.

[MT90] F. Moller and C. Tofts. A temporal calculus of communicating systems. In *Proceedings of CONCUR'90*, number 458 in LNCS. Springer-Verlag, 1990.

[Mun84] J. R. Munkres. *Elements of Algebraic Topology.* Addison-Wesley, 1984.

[NRSV90] X. Nicollin, J.-L. Richier, J. Sifakis, and J. Voiron. ATP: an algebra for timed processes. In *Proc. of the IFIP TC 2 Working Conference on Programming Concepts and Methods*, 1990.

[NS92] X. Nicollin and J. Sifakis. An overview and synthesis of timed process algebras. In *Proc. of CAV'92*, number 575 in LNCS, pages 376–398. Springer-Verlag, 1992.

[PF94] T. A. Proebsting and C. W. Fraser. Detecting pipeline structural hazards quickly. In *Proc. of the Symposium on Principles of Programming Languages.* ACM Press, 1994.

[Plo81] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus, 1981.

[Plo84] G. Plotkin. Domains. Technical report, Computer Science Department, Edinbourgh, 1984.

[Plo85] G. Plotkin. Denotational semantics with partial functions. Technical report, Stanford University, Lectures at the C.S.L.I. Summer School, 1985.

[PP92] P. Peterson and D. Padura. Dynamic dependence analysis: a novel method for data dependence evaluation. In *Proceedings of the fifth Workshop on Languages and Compilers for Parallel Computing*, number 757 in LNCS. Springer-Verlag, August 1992.

[Pra86] V. Pratt. Modeling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33–71, 1986.

[Pra91a] V. Pratt. Event spaces and their linear logic. Technical report, Stanford University, 1991.

[Pra91b] V. Pratt. Modeling concurrency with geometry. In *Proc. of the 18th ACM Symposium on Principles of Programming Languages*. ACM Press, 1991.

[Pra92] V. Pratt. The duality of time and information. In *Proc. of CONCUR'92*, number 630 in LNCS, Stonybrook, New York, August 1992. Springer-Verlag.

[Pra94a] V. Pratt. Chu spaces: Automata with quantum aspects. Technical report, Stanford University, 1994.

[Pra94b] V. Pratt. Chu spaces: Complementarity and uncertainty in rational mechanics. Technical report, Stanford University, 1994.

[Ren93] A. Rensink. *Refinement of actions*. PhD thesis, CWI, Amsterdam, 1993.

[Rep92] J.H. Reppy. *Higher-order concurrency*. PhD thesis, Department of Computer Science, Cornell University, 1992.

[RR87] G. M. Reed and A. W. Roscoe. Metric spaces as models for real-time concurrency. In *Proc. of Mathematical Foundations of Programming Languages and Semantics*, number 298 in LNCS, pages 331–343. Springer-Verlag, 1987.

[RR88a] G. M. Reed and A. W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, (58):249–261, 1988.

[RR88b] E. Robinson and G. Rosolini. Categories of partial maps. *Information and Computation*, (79):95–130, 1988.

[Run59] H. Rund. *The Differential Geometry of Finsler Spaces*. Springer-Verlag, 1959.

[Sch86] D. A. Schmidt. *Denotational Semantics, a methodology for language developpement*. Wm. C. Brown Publishers, 1986.

[Sch91] S. Schneider. An operational semantics for timed CSP. Technical report, Programming Research Group, Oxford University, February 1991.

[Ser51] J.P. Serre. *Homologie Singulière des Espaces Fibrés. Applications*. PhD thesis, Ecole Normale Supérieure, 1951.

[Shi85] M.W. Shields. Concurrent machines. *Computer Journal*, 28, 1985.

[SMC91] J. Salz, R. Mirchandaney, and K. Crowley. Run-time parallelization and scheduling of loops. *IEEE Transactions on Computer*, 40(5), May 1991.

[SNW94] V. Sassone, M. Nielsen, and G. Winskel. Relationships between models of concurrency. In *Proceedings of the Rex'93 school and symposium*, 1994.

[SOK94] C. C. Squier, F. Otto, and Y. Kobayashi. A finiteness condition for rewriting systems. *Theoretical Computer Science*, 131:271–294, 1994.

[Spa66] E. J. Spanier. *Algebraic Topology*. McGraw Hill, 1966.

[Sta89] A. Stark. Concurrent transition systems. *Theoretical Computer Science*, 64:221–269, 1989.

[SZ93] M. Saks and F. Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. In *Proc. of the 25th STOC*. ACM Press, 1993.

[Ten91] R. D. Tennent. *Semantics of Programming Languages*. Prentice Hall, 1991.

[Tho89] B. Thomsen. CHOCS: a calculus of higher-order communicating systems. In *Proc. of the 16th Symposium on Principles of Programming Languages*. ACM Press, 1989.

[Ull82] J. D. Ullman. *Principle of Database Systems*. Pitman, 1982.

[vEB90] P. van EMde Boas. Machine models and simulations. In *Handbook of Theoretical Computer Science*, volume A. Elsevier and The MIT Press, 1990.

[vG90] R. van Glabbeek. *Comparative concurrency semantics and refinement of actions*. PhD thesis, Centrum voor Wiskunder en Informatica, 1990.

[vG91] R. van Glabbeek. Bisimulation semantics for higher dimensional automata. Technical report, Stanford University, 1991.

[vGG89] R. van Glabbeek and U. Goltz. Partial order semantics for refinement of actions. *Bulletin of the EATCS*, (34), 1989.

[Win88] G. Winskel. An introduction to event structures. Number 354 in LNCS. Springer-Verlag, 1988.

[Win93] G. Winskel. *The formal semantics of programming languages.* The MIT Press, 1993.

[WN94] G. Winskel and M. Nielsen. *Models for concurrency*, volume 3 of Handbook of Logic in Computer Science, pages 100–200. Oxford University Press, 1994.

[Yi90] W. Yi. Real-time behavior of asynchronous agents. In *Lecture Notes in Computer Science*, volume 458, pages 502–520. Springer-Verlag, 1990.