Sémantique et Analyse de Programmes Parallèles

Eric Goubault
CEA/LETI/DEIN/SLA Groupe Sécurité du Logiciel

Plan

- Propriétés de programmes parallèles
- Complexité de la sémantique par entrelacements
- Complexité intrinsèque de la preuve de propriétés de programmes parallèles - exemples
- Moyens sémantiques de réduction de la complexité, dans les "bons cas" ?
- Sémantiques du "vrai parallélisme"
- "Process Graphs"
- Algorithmes de détection de points morts
- Propriétés plus fines du comportement

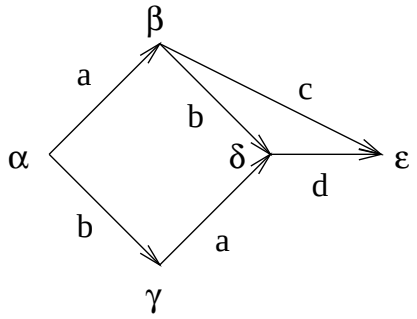
Rappels – Systèmes de transitions

Définition 1 *Un système de transitions est une structure*

$$(S, i, L, Tran)$$

- S *ensemble d'états et* i *état initial,*
- L *ensemble d'étiquettes, et*
- $Tran \subseteq S \times L \times S$ *la relation de transition*

Exemple



Catégorie

On en fait une catégorie avec pour morphismes les simulations (Glynn Winskel et al.).

Un système de transition T_1 simule un système de transitions T_0 à la condition que si T_0 peut effectuer une action a dans un certain contexte, alors T_1 peut effectuer a également dans un contexte similaire.

Un morphisme $f : T_0 \to T_1$ definit la façon dont les états et les transitions de T_0 sont reliés aux états et transitions de T_1.

Morphismes

Définition 2 *Soit* $T_0 = (S_0, i_0, L_0, Tran_0)$ *et* $T_1 = (S_1, i_1, L_1, Tran_1)$ *deux systèmes de transition. Un morphisme* $f : T_0 \to T_1$ *est un couple* $f = (\sigma, \lambda)$ *où,*

- $\sigma : S_0 \to S_1$,
- $\lambda : L_0 \to L_1$ *est tel que* $\sigma(i_0) = i_1$ *et*

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

Référence: définition différente de celle de G. Winskel et al. (pas de morphismes "partiels").

Morphismes partiels

Les morphismes partiels permettent à T_1 de ne pas effectuer d'action quand T_0 le fait.

Définition 3 *Soient* $T_0 = (S_0, i_0, L_0, Tran_0)$ *et* $T_1 = (S_1, i_1, L_1, Tran_1)$ *deux systèmes de transition. Un morphisme partiel* $f : T_0 \to T_1$ *est un couple* $f = (\sigma, \lambda)$ *où,*

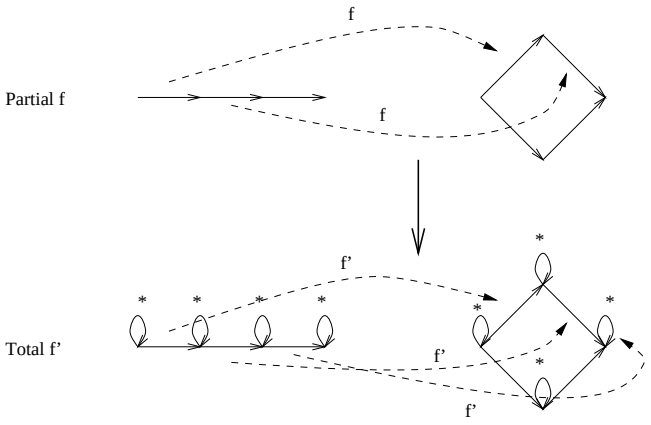
- $\sigma : S_0 \to S_1$,
- $\lambda : L_0 \to L_1$ *est une fonction* partielle. (σ, λ) *est tel que*
  - $\sigma(i_0) = i_1$,
  - $(s, a, s') \in Tran_0$ *et* $\lambda(a)$ *est défini implique* $(\sigma(s), \lambda(a), \sigma(s')) \in Tran_1$. *Sinon, si* $\lambda(a)$ *n'est pas défini alors* $\sigma(s) = \sigma(s')$.

Transitions silencieuses

On complète un système de transition $T = (S, i, L, Tran)$ en $T_* = (S_*, i_*, L_*, Tran_*)$,

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

Exemple



Produits Cartésien

Soient deux systèmes de transitions $T_0 = (S_0, i_0, L_0, Tran_0)$ et $T_1 = (S_1, i_1, L_1, Tran_1)$. Leur produit est $T_0 \times T_1 = (S, i, L, Tran)$ avec,

- $S = S_0 \times S_1$ et $i = (i_0, i_1)$. Soient $\rho_0 : S_0 \times S_1 \to S_0$ et $\rho_1 : S_0 \times S_1 \to S_1$ les projections canoniques,
- $L = L_0 \times_* L_1 = \{(a, *)/a \in L_0\} \cup \{(*, b)/b \in L_1\} \cup \{(a, b)/a \in L_0, b \in L_1\}$ avec des projections canoniques π_0 et π_1,
- $(s, a, s') \in Tran_*$ ssi $(\rho_0(s), \pi_0(a), \rho_0(s')) \in Tran_{0*}$ et $(\rho_1(s), \pi_1(a), \rho_1(s')) \in Tran_{1*}$

Autres Constructions catégoriques

Le coproduit de deux systèmes de transitions $T_0 = (S_0, i_0, L_0, Tran_0)$ et $T_1 = (S_1, i_1, L_1, Tran_1)$ est $T_0 \oplus T_1 = (S, i, L, Tran)$ avec,

- $S = (S_0 \cup S_1)/\{i_0 = i_1)$, et $i = i_0 = i_1$,
- $L = L_0 \cup L_1$,
- $Tran = Tran_1 \cup Tran_2$.

Produits Synchronisés

Un cas particulier: produit fibré suivant.

Référence plus générale: Arnold et M.Nivat.

Soient $T_0 = (S_0, i_0, L_0, Tran_0)$ et $T_1 = (S_1, i_1, L_1, Tran_1)$ deux systèmes de transition. Leur produit synchronisé (à la CSP) est $T = (S, i, L, Tran)$ avec,

Définition

- $S = S_0 \times S_1$, $i = (i_0, i_1)$,
- $L = L_0 \times L_1$,
- $((s,t), a, (u,v)) \in Tran$ ssi
  - $a \in L_0 \cap L_1$ et $(s,a,u) \in Tran_0$ et $(t,a,v) \in Tran_1$,
  - $a \in L_0 \backslash L_1$ et $(s,a,u) \in Tran_0$ et $v = t$,
  - $a \in L_1 \backslash L_0$ et $u = s$ et $(t,a,v) \in Tran_1$.

Sémantique de processus parallèles

- Automate pour chaque processus séquentiel,
- produit synchronisé correspondant aux différentes primitives de communications utilisées.

Les états globaux du système de transition résultant P sont donc des n-uplets (n=nombre de processus communicants) $s = (s_1, \ldots, s_n)$ avec s_i état local du processus P_i.

Jusqu'à une exponentielle de plus par rapport au cas séquentiel!

Equivalence "atteignabilité", "point mort"

- Un état (global) $s \in S$ est atteignable ssi il existe un chemin de i vers s (i.e. une suite de transitions $(i, a_1, s_1) \in Tran$, $(s_1, a_2, s_2) \in Tran$, $\ldots$, $(s_{n-1}, a_n, s_n = s) \in Tran$),
- Un point mort est un état global s tel qu'il n'existe aucun $a \in L$ ni $s' \in S$ avec $(s, a, s') \in Tran$.

Equivalence

Soit un état global $s = (s_1, \ldots, s_n)$ de P. On veut déterminer s'il est atteignable.

Ajouter à chaque P_i une transition $(s_i, \delta_i, stop_i)$. Le produit synchronisé des automates modifiés est appelé P_M.

Alors $s' = (stop_1, \ldots, stop_n)$ est un point mort pour P_M ssi s est atteignable.

Equivalence "atteignabilité locale, "point mort"

- Un état local l est un sous-ensemble de $\bigcup_i S_i$, tel que $\forall 1 \leq i \leq n$, $\mid l \cap S_i \mid \leq 1$,
- l est atteignable ssi il existe un état global $s = (s_1, \ldots, s_n)$ tel que $l \subseteq \{s_1, \ldots, s_n\}$.

Equivalence

Soit P_l l'ensemble des processus qui ont un état dans l. Soit $P_{\neg l}$ son complémentaire.

Pour chaque P_i dans $P_{\neg l}$ on rajoute une transition $(s_i, \delta_i, stop_i)$ à un état s_i de chaque cycle dans le graphe de P_i.

Pour chaque $P_i \in P_l$, on rajoute des transitions $(s_i, \delta, stop_i)$ (pour $s_i \in l$) et $(stop_i, \delta_j, stop_i)$ pour tout j tel que $P_j \in P_{\neg l}$.

Equivalence

Le nouveau produit synchronisé forme l'automate $P_{M'}$.

Proposition 1 *Un état local l de P est atteignable ssi il existe un point mort s' dans $P_{M'}$ tel que $\forall i$ avec $P_i \in P_l$, $stop_i \in s'$.*

Référence: P. Godefroid et P. Wolper (CAV'90).

Equivalence "sureté", "points mort"

- Ψ peut être représenté par un automate fini A_Ψ clos par préfixes,
- Comme A_Ψ est clos par préfixes, $A_{\neg\Psi}$ est un automate avec un seul état d'acceptation (que l'on nomme X)

Equivalence

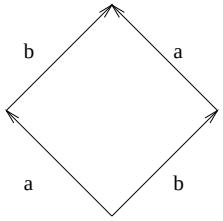
Une fois $A_{\neg\Psi}$ construit, on a,

Proposition 2 *le problème de la vérification de la propriété Ψ pour l'automate P produit des automates A_i $(1 \leq i \leq n)$ est équivalent au problème d'atteignabilité locale de l'état X du produit des A_i avec $A_{\neg\Psi}$.*

Corollaire 1 *Le problème de vérification de l'invariant Ψ est équivalent au problème de trouver un point mort*

Complexité?

Dans la sémantique des traces, ou la sémantique par entrelacements, l'exécution parallèle de deux actions a et b est représentée par,



La preuve de propriété se fait essentiellement en considérant tous les chemins possibles d'exécution (que ce soit par méthodes à la Hoare [preuves de non-interférences etc.], ou par model-checking, ou par analyse statique etc.).

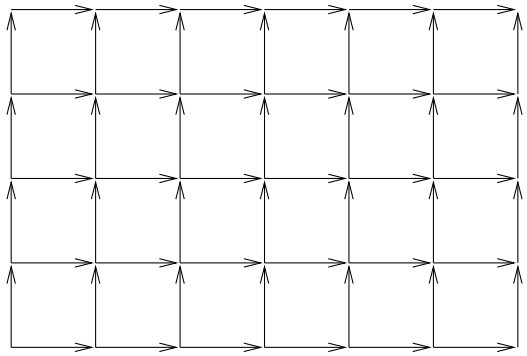
Détection de points morts classiques

On calcule les ascendants de l'état initial par itération sur le système de transition.

On ne conserve que les états (globaux) qui ne sont pas descendants (immédiats) des états parcourus.

On arrête quand on n'a plus aucune transition à exécuter.

Exemple



Nombre de chemins exponentiel en le nombre d'états des processus.

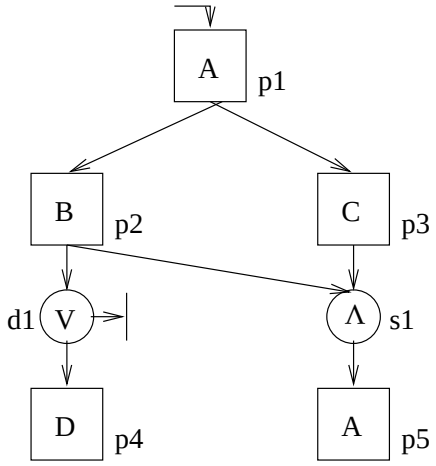
Que faire? Cette complexité est-elle intrinsèque au parallélisme?

Coordination de Processus

Petit modèle intuitif: une spécification de contrôle de processus W est,

- X, un ensemble d'objets de contrôle de processus:
  - S: ensemble de synchronisation ("$\wedge$"),
  - P: ensemble de processus ("petites boîtes"),
  - D: ensemble de décisions ("$\vee$"), comprenant les décisions de terminaison (ensemble D_t).
- $Trig \subseteq X \times X$: ensemble de déclencheurs ("$\rightarrow$"),
- $Sup : X \rightharpoonup V$: fonction de décomposition ("boîtes"). Soit Q l'image de la fonction partielle Sup (l'ensemble de tous les noms de groupes) et $A = V \backslash Q$ l'ensemble des actions atomiques,
- $I \subseteq X$ contient les objets initiaux.

Exemple

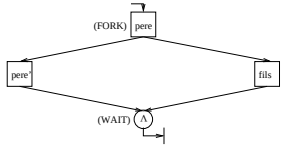


Exemple réél

FORK/WAIT

- processus père et fils sont les mêmes syntaxiquement - déterminent qui ils sont à l'exécution,
- synchronisation en fin de calcul par WAIT

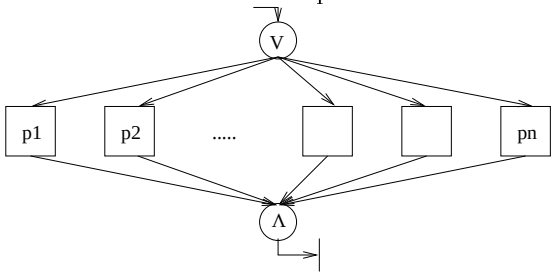
```
main(argc,argv)
{
   ...
   if ((pid=fork())==0) then { fils }
   else { pere;pid est process id fils }
   ...
}
```



Exemple réél

COBEGIN/COEND

- mise en parallèle de n processus
- synchronisation en fin de calcul par COEND



Exemple – La machine PVM

La machine PVM est une collection de machines réelles sur un même réseau sur lesquelles un démon pvmd3 tourne:

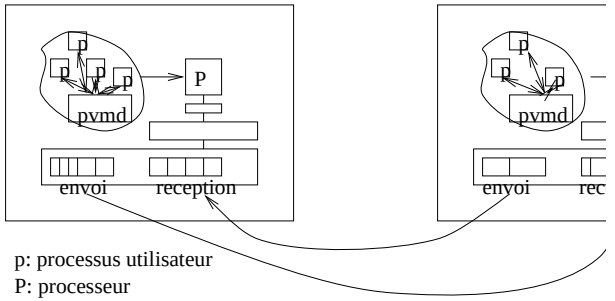
- pvmd3 est un processus UNIX qui contrôle les processus utilisateurs au sein d'une application PVM et qui coordonne les communications entre les machines qui composent la machine PVM
- un démon pvmd3 par ordinateur et par utilisateur de la machine PVM qui maintient la configuration globale et locale de la machine PVM
- chaque machine (et architecture) doit avoir son propre pvmd3

La machine PVM

La librairie PVM permet de communiquer avec ces démons pvmd3 pour,

- créer et détruire des processus,
- gérer des tampons de communication, envoyer, recevoir (point à point ou par broadcast) des messages (de façon synchrone ou asynchrone),
- synchroniser des processus (par barrières de synchronisation)
- connaître l'état de la configuration de la machine PVM et pour la modifier (dynamiquement)

Parallel Virtual Machine



pvm_spawn

Appel PVM simplifié,

```
int numt,etid[n];
numt=pvm_spawn(``programme'',NULL,PvmTaskDefault,NULL,
                                                n,&etid[0]);
```

Permet de lancer `n` tâches PVM exécutant le code ``` ``programme'' ```

Le `tid` de chacune des tâches est rangé dans une entrée du tableau `etid`

Renvoie `numt <= n` le nombre de tâche(s) effectivement lancée(s)

Les groupes de processus

Un groupe est un ensemble de processus,

- auquel on peut ajouter ou enlever dynamiquement de nouveaux membres (`pvm_joingroup`, `pvm_lvgroup`),
- dans lequel on peut gérer des processus individuels (`pvm_getinst`, `pvm_gettid`),
- dont on peut synchroniser tous les membres (`pvm_barrier`),
- auquels on peut "broadcaster" un message (`pvm_bcast`) ou répartir un ensemble de données (`pvm_scatter` etc.)
- qui peuvent exécuter une même opération sur leurs données locales (`pvm_reduce`).

PVM_JOINGROUP

```
int inum;
inum=pvm_joingroup(char *group);
```

- Permet d'ajouter la tâche qui a appelé `pvm_joingroup` au groupe qui a pour nom la chaîne de caractères `group`.
- Renvoie son numéro "d'instance" dans le groupe. C'est un numéro d'ordre entier commençant à 0 et s'incrémentant à chaque nouveau membre.
- Erreur si `inum < 0`.

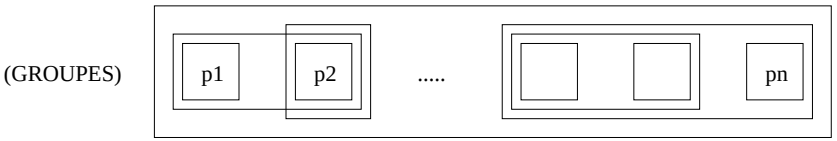
PVM_LVGROUP

```
int info;
info=pvm_lvgroup(char *group);
```

- Retire le processus appelant du groupe `group`.
- Erreur si `info < 0`.

Spécification du contrôle

Par une représentation de tous les groupes possibles,

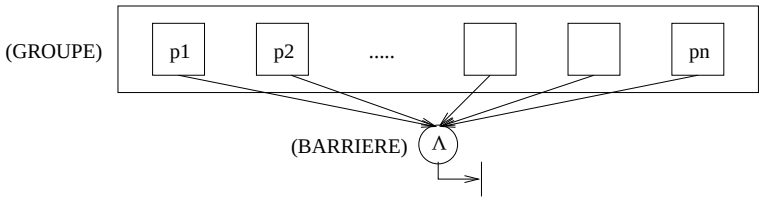


PVM_BARRIER

```
int info;
info=pvm_barrier(char *group,int count);
```

- Bloque le processus exécutant `pvm_barrier` jusqu'à ce que `count` membres du groupe `group` se soient également bloqués.
- Permet de synchroniser tout un groupe. `count` permet de gérer le fait que d'autres processus ont pu s'ajouter au groupe pendant l'attente.
- Erreur si `info < 0`.

SPÉCIFICATION DU CONTRÔLE



PROBLÈME DU DÉMARRAGE D'UN PROCESSUS

Y a t-il une suite d'événements qui conduit à l'exécution d'un processus p?

Théorème 1 *Le problème du démarrage d'un processus pour les spécifications de contrôle de processus sans décomposition (domaine de définition de Sup est vide) est* **NP-complet**.

Référence: "On the Complexity of Some Verification Problems in Process Control Specifications", A. H. M. ter Hofstede et M. E. Orlowska, Technical Report, Faculty of Information Technology, Queensland University of Technology, Number FIT-TR-97-02, 29 avril 1997.

PREUVE

Par transformation en temps polynomial du problème SAT en le problème du démarrage d'un processus.

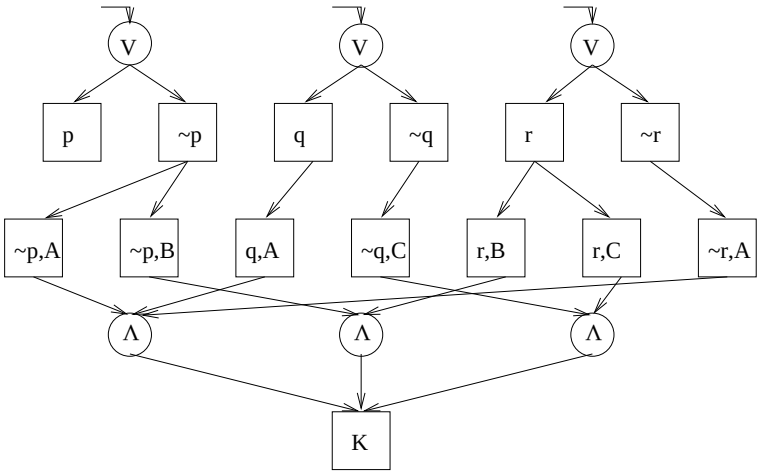
Soit K une instance du problème SAT. Soit u une variable booléenne apparaissant dans K.
A u on associe les objets $X = \{D_u, u, \tilde{}u\}$, où D_u est une décision initiale, et les déclencheurs $Trig = \{(D_u, u), (D_u, \tilde{}u)\}$.
A chaque literal x, on associe un processus de nom x.

Preuve

On construit les déclencheurs de telle façon que pour toute clause C de K telle que $\check{}x \in C$, un déclencheur existe de x au processus de nom $< \check{}x, C >$.

Les synchronisateurs sont construits de telle façon que pour toute clause C de K, on ait un synchronisateur S_C avec comme entrée tous les processus avec un nom de la forme $< x, C >$. Enfin tout synchronisateur S_C a un déclencheur de sortie à un processus de nom K.

$$K = A \wedge B \wedge C = (p \vee \neg q \vee r) \wedge (p \vee \neg r) \wedge (q \vee \neg r)$$



Fin de la démonstration

Alors, "K" peut être démarré ssi K n'est pas satisfiable.

La transformation effectuée est polynomiale. Donc le problème du démarrage est au moins NP-difficile.

D'autre part, le problème n'est pas dans NP car la vérification d'un scénario d'éxécution peut être faite en temps polynomial. A remarquer que la décomposition n'est jamais utilisée.

Problème de Terminaison

C'est déterminer si il existe une exécution partant d'un objet initial, qui arrive sur une décision de terminaison.

Théorème 2 *Le problème de la terminaison est DSPACE(exp)-difficile*

Preuve

Par réduction au problème d'atteignabilité dans les réseaux de Pétri généraux.

Corollaire: Le problème de terminaison généralisé: depuis tout état atteignable, est-il possible d'atteindre un état terminal? est DSPACE(exp)-difficile.

Points Morts

Théorème 3 *Le problème de savoir si une spécification de contrôle de processus sans décomposition n'a pas de point mort est DSPACE(exp)-difficile*

Par transformation au problème d'atteignabilité dans les réseaux de Pétri: trouver un point mort c'est déterminer si un des marquages atteignables est un point mort, c'est donc un cas particulier du problème d'atteignabilité.

Corollaires

Corollaire 1: Le problème de savoir si un objet x peut être démarré et que ce processus peut se retrouver en un point mort est DSPACE(exp)-difficile

Corollaire 2: Le problème du "livelock": c'est à dire qu'il existe un état, ni point-mort, ni terminal dans lequel aucune action atomique ne peut être effectuée, est DSPACE(exp)-difficile

Faire mieux?

Par des sémantiques du "vrai parallélisme", peut on arriver à des calculs simples au moins dans le cas de processus très asynchrones?

En effet on a tendance à identifier la complexité d'un problème d'analyse avec la complexité des coordinations (synchronisations) et non avec celle de l'asynchronie.

Les systèmes de transitions cumulent les difficultés

Evidemment, cas le pire inchangé! Mais éventuellement bon comportement dans des sous-cas intéressants.

Les traces de Mazurkiewicz généralisées

Définition 4 *Un language de traces généralisé est un triplet* (M, I, L) *où,*

- L *ensemble de symboles,*
- $M \subseteq L^*$,
- $I : M \to 2^{L \times L}$ *est une fonction qui associe à chaque* $s \in M$ *une relation* $I_s \subseteq L \times L$.

Suite

tel que, si on définit $\cong$ comme la plus petite relation d'équivalence sur L^* telle que $sabu \cong sbau$ si aI_sb, et,

- pour tous $s \in M$, I_s est symétrique et irréflexive,
- (I est cohérente) $s \cong s'$ implique $I_s = I_{s'}$,
- (M est I-fermé) aI_sb implique $sab \in M$,
- (I est cohérente)
  (i) aI_sb et $aI_{sb}c$ et $cI_{sa}b$ implique $aI_{sc}b$,
  (ii) aI_sc et cI_sb implique (aI_sb ssi $aI_{sc}b$).

Morphismes

Définition 5 *Soient* (M, I, L) *et* (M', I', L') *deux langages de trace généralisés. Une fonction partielle* $\lambda : L \to L'$ *est un morphisme de* (M, I, L) *vers* (M', I', L') *ssi,*

- λ *preserve les mots:* $s \in M$ *implique* $\lambda^*(s) \in M'$,
- λ *respecte la relation d'indépendance:*
  aI_sb *et* $\lambda(a)$, $\lambda(b)$ *definis implique* $\lambda(a)I'_{\lambda^*(s)}\lambda(b)$ *où* λ^* *est une extension de* λ *sur les mots définie comme suit,*
  - $\lambda^*(\epsilon) = \epsilon$,
  - $\lambda^*(sa) = \begin{cases} \lambda^*(s)\lambda(a) & si\ \lambda(a)\ est\ defini \\ \lambda^*(s) & sinon \end{cases}$

Détection de Points Morts induite

Soit $t = (s, a, s') \in Tran$ avec $s = (s_1, \ldots, s_n)$ et $s' = (s'_1, \ldots, s'_n)$ une transition du système de transition produit synchronisé vu précédemment. On définit,

- son pré-ensemble $-t = \{s_i \in s / (s_i, a, s'_i) \in Tran_i\}$,
- son post-ensemble $t- = \{s'_i \in s' / (s_i, a, s'_i) \in Tran_i\}$,
- sa proximité $-t- = -t \cup t-$.

Equivalence de transitions

Deux transitions globales $t_1 = (s_1, a_1, s_1')$ et $t_2 = (s_2, a_2, s_2')$ sont équivalentes ($\cong$) ssi

$$-t_1 = -t_2 \wedge t_1- = t_2 - \wedge a_1 = a_2$$

Soit Tr=Tran/$\cong$.

Relation de dépendance

On définit la relation de dépendance entre transitions D par

$$(t_1, t_2) \in D \Leftrightarrow -t_1 - \cap - t_2 - \neq \emptyset$$

Cela permet de définir une relation d'indépendance I et un langage de traces de Mazurkiewicz pour le système parallèle.

La théorie des langages de trace assure que la considération d'un représentant par classe d'équivalence de traces modulo I suffit à trouver les points morts.

Principe de l'algorithme

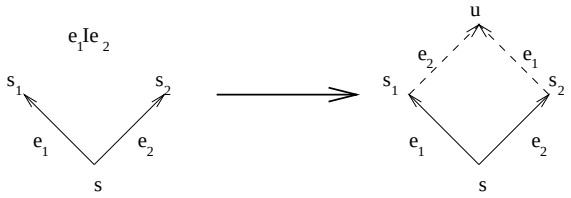
- On parcourt toujours les états globaux mais en ne suivant que certains des chemins. Le principe de sélection est,
- Quand plusieurs transitions indépendantes peuvent être exécutées à partir d'un état, on n'exécute qu'une d'entre elles (au hasard)
- Quand les transitions que l'on peut exécuter à partir d'un état sont dépendantes, leurs exécutions peuvent amener à des traces différentes. Pour gérer cela on maintient un "sleep set" qui est un ensemble de transitions associé à chaque état que l'on a atteint durant le parcours des états. C'est l'ensemble des transitions que l'on pouvait exécuter depuis un état mais que l'on n'a pas choisi d'exécuter durant le parcours. On les exécutera plus tard

Systèmes de transition asynchrone

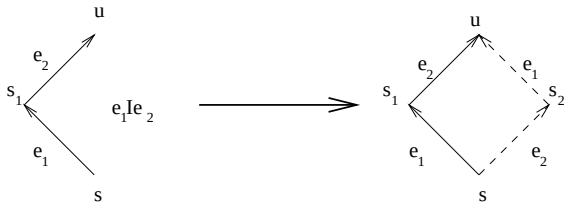
Définition 6 *Un système de transition asynchrone est un quintuplet (S,i,E,I,Tran),*

- *(S,i,E,Tran) est un système de transition,*
- *$I \subseteq E \times E$ est une relation symétrique irréflexive (la relation d'"indépendance") telle que,*
  - **(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)** $e_1 I e_2 \wedge (s, e_1, s_1) \in Tran \wedge (s, e_2, s_2) \in Tran \Rightarrow \exists u, (s_1, e_2, u) \; Tran \wedge (s_2, e_1, u) \in Tran$
  - **(4)** $e_1 I e_2 \wedge (s, e_1, s_1) \in Tran \wedge (s_1, e_2, u) \in Tran \Rightarrow \exists s_2, (s, e_2, s_2) \; Tran \wedge (s_2, e_1, u) \in Tran$

Condition (3)



Condition (4)



Morphismes

Ce sont les morphismes de systèmes de transition f préservant la relation d'indépendance:

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

Automates de trace

Définition 7 *Un automate de trace est un triplet $A = (E, Q, T)$,*

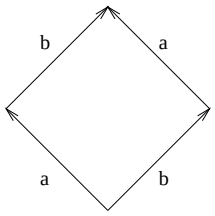
- *E est un alphabet "parallèle", c'est à dire un ensemble d'événements sur lequel on a une relation binaire symétrique, irreflexive $\|_E$ appelée relation de parallélisme*
- *Q ensemble d'états,*
- *$T \subseteq Q \times (E \cup \{\epsilon\}) \times Q$ ensemble de transitions.*

Automates de Trace

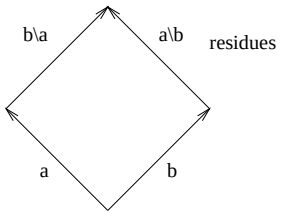
Tout cela est soumis aux conditions suivantes,

- $q \xrightarrow{\epsilon} r$ ssi $q = r$,
- si $q \xrightarrow{a} r$ et $q \xrightarrow{a} r'$ alors $r = r'$ (similaire à la condition (2) des systèmes de transition asynchrones),
- pour tous les états q et événements a, b, si $a \|_E b$, $q \xrightarrow{a} r$ et $q \xrightarrow{b} s$ alors pour un état p il existe des transitions $s \xrightarrow{a} p$ et $r \xrightarrow{b} p$ (similaire à la condition (3) des systèmes de transition asynchrones).

Equivalence par permutation et résidus



Permutation



Strong confluence

Idée similaire!

Système de transition concurrent

Définition 8 *Un système de transition concurrent (CTS) est une structure* $(G, \uparrow)$,

- $G = (O, A, dom, cod, id)$ *est un graphe "avec identités" c.a.d.,*
  - O *ensemble d'états,*
  - A *ensemble de transitions,*
  - $dom : A \to O$ *relie les transitions à leurs états initiaux,*
  - $cod : A \to O$ *relie les transitions à leurs états finaux,*
  - $id : O \to A$ *relie chaque* $s \in O$ *à une transition spéciale (les transitions "*" des systèmes de transition standards)* id_s *telle que* $dom(id_s) = s$ *et* $cod(id_s) = s$.

Suite

- $\uparrow$: $Coin(G^\#) \to A^\#$ est *l'opération résidu* qui vérifie,
  - $G^\#$ est le graphe augmenté $(O^\#, A^\#, dom, cod, id)$ avec,
    - $O^\# = O \cup \{\Omega\}$ (Ω n'appartient pas à O),
    - $A^\# = A \cup \{\omega_q / q \in O^\#\}$, $dom(\omega_q) = q$, $cod(\omega_q) = \Omega$.
  - $Coin(X)$ (où X est un graphe) est l'ensemble des transitions *coinitiales*, c.a.d. l'ensemble des couples (t, u) de transitions t, u de X qui ont les mêmes états de départ.

soumis aux conditions suivantes,

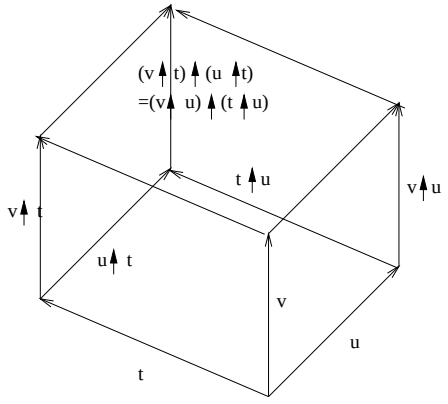
(1) pour tout $t \in A^{\#}$ et $u \in A^{\#}$,
   (a) $dom(t \uparrow u) = cod(u)$,
   (b) $cod(t \uparrow u) = cod(u \uparrow t)$.

(2) pour tout $t : q \to r \in A^{\#}$,
   (a) $id_q \uparrow t = id_r$,
   (b) $t \uparrow id_q = t$,
   (c) $t \uparrow t = id_r$.

(3) pour toutes transitions coinitiales t, u, v dans $A^{\#}$, $(v \uparrow t) \uparrow (u \uparrow t) = (v \uparrow u) \uparrow (t \uparrow u)$ (l'"axiome du cube")

(4) pour toutes transitions coinitiales t, u dans $A^{\#}$, si $t \uparrow u$ et $u \uparrow t$ sont toutes les deux des identités alors $t = u$.

L'axiome du cube



Morphismes

Définition 9 *Un morphisme de systèmes de transition concurrent est une paire de fonctions* $\rho = (\rho_O, \rho_A) : (O, A, dom, cod, id, \uparrow) \to (O', A', dom', cod', id', \uparrow')$ *telle que,*

- $\rho_O : O \to O'$ *et* $\rho_A : A \to A'$ *sont des fonctions avec* $dom' \circ \rho_A = \rho_O \circ dom$, $cod' \circ \rho_A = \rho_O \circ cod$ *et* $\rho_A \circ id = id' \circ \rho_O$
- *Si* t, u *sont des transitions propres chérentes de* C *alors* $\rho(t \uparrow u) = \rho(t) \uparrow' \rho(u)$.

Systèmes de transitions avec indépendance

Définition 10 *Un système de transition avec independance est une structure* $(S, s^I, L, Tran, I)$ *avec* $(S, s^I, L, Tran)$ *un système de transition et* $I \subseteq Tran^2$ *une relation irreflexive et symétrique telle que*

(i) $(s, a, s') \sim (s, a, s'') \Rightarrow s = s''$ *(condition (2) des systèmes de transition asynchrones),*

(ii) $(s, a, s')I(s, b, s'') \Rightarrow \exists u\, (s, a, s')I(s', b, u) \wedge (s, b, s'')I(s'', a, u)$ *(condition (3) des systèmes de transition asynchrones),*

(iii) $(s, a, s')I(s', b, u) \Rightarrow \exists s''(s, a, s')I(s, b, s'') \wedge (s, b, s'')I(s'', a, u)$ *(condition (4) des systèmes de transition asynchrones),*

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

Suite

où $\sim$ est la plus petite equivalence sur les transitions contenant la relation R définie par,

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

Morphismes

Ce sont des paires de fonctions (σ, λ) qui forment des morphismes de systèmes de transition plus,

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

Réseaux de Pétri

Définition 11 *Un réseau de Pétri $N = (P, T, pre, post)$ est composé de,*

- *P ensemble de places,*
- *T ensemble de transitions,*
- *$pre : T \to \tilde{P}$ est la fonction de pré-condition, où $\tilde{P}$ est l'ensemble des multi-ensembles de P,*
- *$post : T \to \tilde{P}$ est la fonction de post-condition.*

On appelle marquage tout multi-ensemble de places

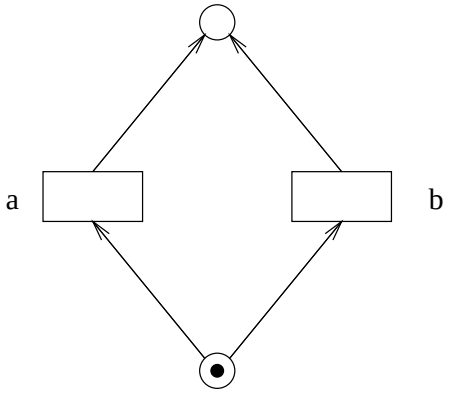
Relation de transition induite

La fonction de pré-condition décrit comment les transitions "consomment" les ressources et la fonction de post-condition montre comment les transitions "créent" des nouvelles resources.
Cela permet de définit une relation de transition entre les marquages.

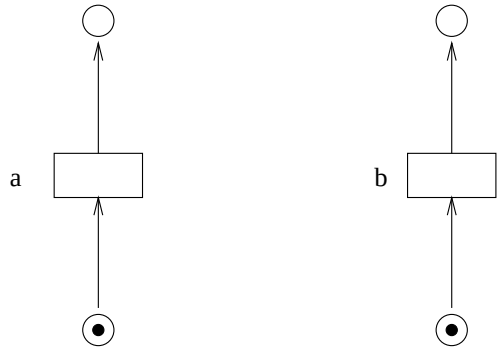
Si M et M' sont deux marquages d'un réseau N, et t est une transition de N on écrit $M[t\rangle M'$ pour "t fait une transition de M à M'" ssi

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

Exemple: Exclusion Mutuelle entre a et b



Exemple: Exécution parallèle de a et de b



Calculs de Points morts induits

Référence: A. Valmari (A stubborn attack on state explosion). Ou présentation traduite en termes de systèmes de transitions (en particulier par P. Godefroid).

Idée:

Définition 12 *Un sous-ensemble T de l'ensemble des transitions exécutables à partir d'un état s est "persistant" ssi toutes les transitions qui ne sont pas dans T et qui sont exécutables à partir de s ou atteignables à partir de s par des transitions qui ne sont pas dans T, sont indépendantes avec toutes les transitions de T.*

Calculs induits

On peut alors faire une recherche d'états menant aux points morts comme suit: On explore itérativement à partir d'un état les transitions qui sont dans un ensemble T avec,

- T est un ensemble persistant,
- $T \neq \emptyset$ dès qu'il existe au moins une transition à exécuter à partir de s.

Critères le plus souvent syntaxiques pour trouver les ensembles persistants (en mémoire partagée par exemple, on regarde les variables que peuvent modifier, ou dont dépend chaque processus).

Autres calculs induits

Référence: S. Melzer et S. Romer (Deadlock Checking using Net Unfoldings). Beaucoup d'autres...

Structures d'événements

Définition 13 *Une structure d'événements première est une structure* $(E, \leq, \#)$ *où* E *est un ensemble d'*événements *partiellement ordonné par* $\leq$ *appelé* relation de dépendance causale *et où* $\# \subseteq E \times E$ *est une relation symétrique irreflexive, la* relation de conflit *satisfaisant,*

- $\{e'/e' \leq e\}$ *est fini (axiome des "causes finies"),*
- $e\#e'$ *and* $e' \leq e''$ *implique* $e\#e''$ *(le conflit est héréditaire).*

Morphismes

Définition 14 *Soit* $S = \{E, \leq, \#\}$ *et* $S' = \{E', \leq', \#'\}$ *des structures d'événements premières. Un morphisme de structures d'événements de* S *vers* S' *est une fonction partielle* $f : E \to E'$ *telle que,*

- *si* $f(e)$ *est définie alors* $\{e'/e' \leq f(e)\} \subseteq f(\{e''/e'' \leq e\}$,
- *si* $f(e_0)$ *et* $f(e_1)$ *sont tous les deux définis* $f(e_0)\#f(e_1)$ *ou* $f(e_0) = f(e_1)$ *implique* $e_0\#e_1$ *ou* $e_0 = e_1$.

Ensemble de configurations

Soit $(E, \leq, \#)$ une structure d'événements. $\mathcal{D}(E, \leq, \#)$ est l'ensemble des configurations de E, sous-ensembles $x \subseteq E$ de E qui sont,

- sans conflit: pour tous $e, e' \in x$, e n'est pas en conflit avec e',
- fermés vers le bas: pour tous e, e', $e \in x$ et $e' \leq e$ implique $e' \in x$

Relation avec les systèmes de transition

Soit $e \in E$ et $c \in \mathcal{D}(E, \leq, \#)$ alors e est permis en la configuration c, ce que l'on écrit par $c \vdash e$ si,

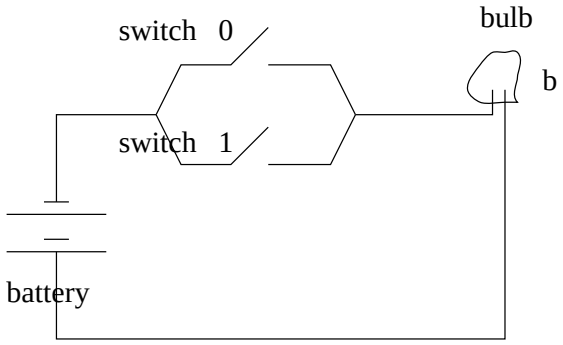
(i) $e \notin c$,

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

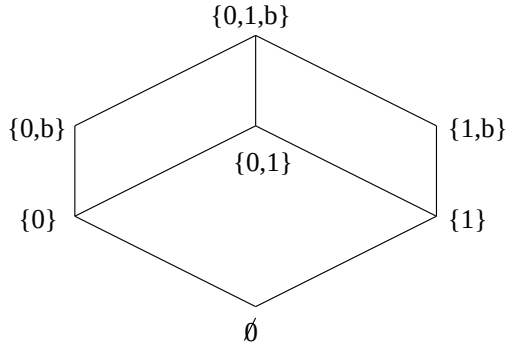
(iii) $e' \in E$ and $e' \# e$ implique $e' \notin c$

Les configurations finies sont des traces quand on ordonne ses éléments avec la dépendance causale. $\{e_1 < e_2 < \ldots < e_n\}$ est une *garantie* pour c ssi $\{e_1, \ldots, e_{i-1}\} \vdash e_i$ pour $i = 1, \ldots, n$. On écrit aussi les garanties comme des chaînes $e_1 \ldots e_n$.

Exemple: un interrupteur parallèle



Exemple: les configurations correspondantes

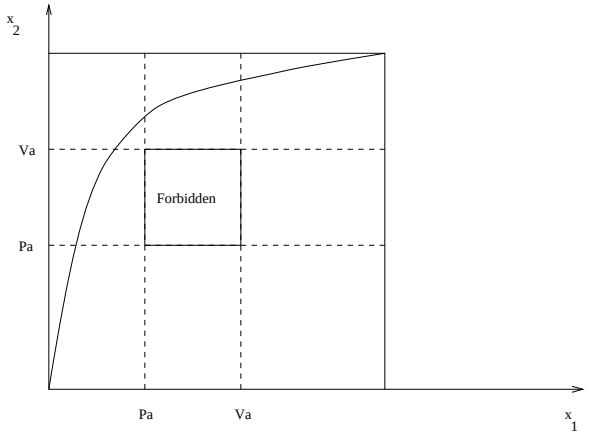


Propriétés catégoriques

Référence: Voir G. Winskel et al. (Models of Concurrency, Handbook in Logic in Computer Science, Vol.3).

Avec ces définitions, il est possible d'avoir des algèbres de modèles avec des équivalents pour les produits synchronisés etc.

Graphes de processus

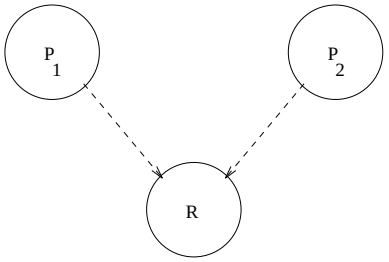


"image continue": x_i = temps local
Référence: A l'origine, E.W. Dijkstra.

Exclusion mutuelle

Problème: Compétition entre n processus pour une ressource

Seul un processus P_i accéde à un moment donné à la ressource R

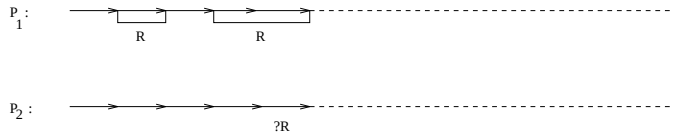


Vers une solution:

- R est muni d'un drapeau d (=0,1),
- Chaque P_i voulant accéder à R fait un **test-and-set** ATOMIQUE sur d
- Si P_i veut libérer R alors il met d à 0

Problème

Un des processus peut bloquer l'accès à tous les autres indéfiniment (**famine**).



Une solution

Sémaphore:

- R est muni d'une **file d'attente** f et d'un compteur s (initialisé à 1),
- Un processus P_i voulant accéder à R fait l'opération P sur R,
  - $s := s - 1$,
  - Si $s < 0$ alors mettre i en queue de f,
  - Sinon continuer l'exécution de P_i
- Un processus P_i voulant libérer R fait l'opération V sur R
  - $s := s + 1$,
  - Si $s \geq 0$ alors prendre j en tête de f et continuer l'exécution de P_j

Exemple:

Processus P_1, P_2 et P_3 dont le code est `P(R);R:=R+1;V(R)`

Instruction:	s:	f:
-	1	[]
`P(R)` (P_1)	0	[]
`P(R)` (P_2)	-1	$[P_2]$
`R:=R+1` (P_1)	-	-
`V(R)` (P_1)	0	[]
`R:=R+1` (P_2)	-	-
`V(R)` (P_2)	1	[]

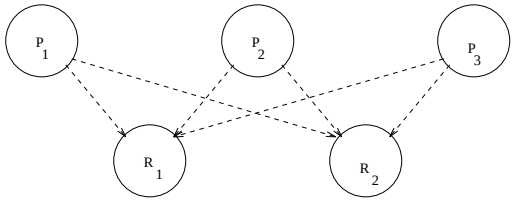
Verrou

Un sémaphore initialisé à un s'appelle un **verrou**

```
SEMAPHORE MUTEX;
INTEGER X;
MUTEX:=1;
X:=2;
A:
   P(MUTEX);
   X:=X+1;
   V(MUTEX);
B:
   P(MUTEX)
   X:=2*X;
   V(MUTEX)
```


Généralisation

Compétition entre n processus P_i pour m ressources R_j

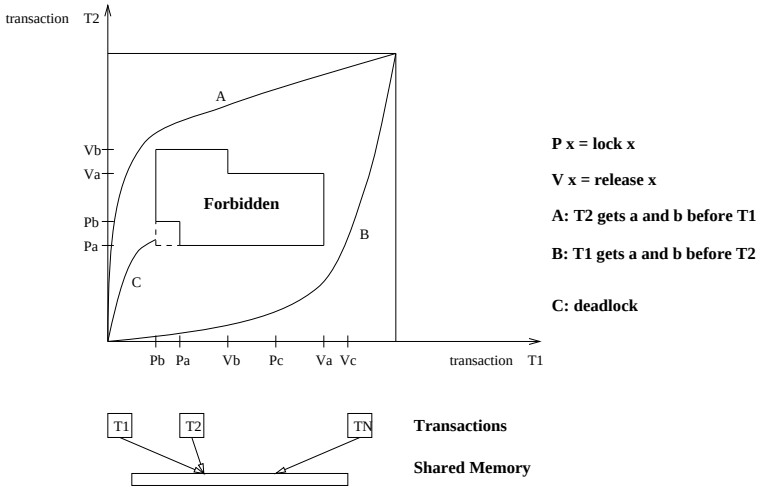


Problème

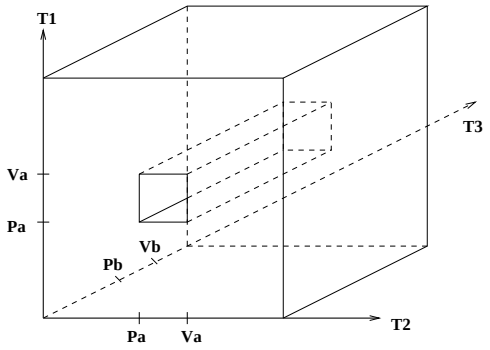
Interblocage

- P_1 a besoin de R_1 pour continuer son calcul et produire R_2
- P_2 a besoin de R_2 pour produire R_1

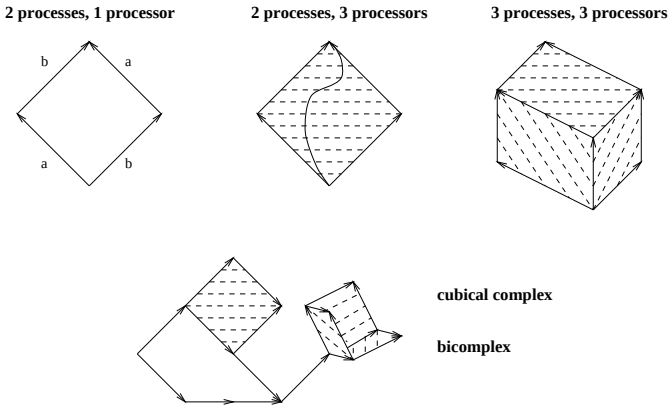
Encore des graphes de processus



En dimension supérieure



Généralisation – HDA



Référence: V. Pratt, R. van Glabbeek, J. Gunawardena, R. Cridlig, E. Goubault

Détection de Points Morts

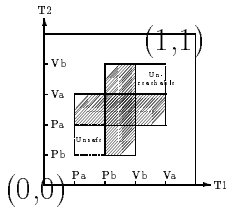
On considère une base de donnée centralisée et un nombre fini de transactions de la forme,

$$\begin{aligned} \mathrm{T} &= \epsilon \\ &\mid \mathrm{Pa.T} \\ &\mid \mathrm{Va.T} \end{aligned}$$

toutes en parallèle.

Exemple

Exemple de sémantique géométrique: $T_1 \mid T_2$ avec $T_1 = P_a P_b V_b V_a$ et $T_2 = P_b P_a V_a V_b$



Exemple

Une région "Unsafe" représente un endroit depuis lequel il est impossible de rejoindre l'état final (1,1).

Le point le plus haut de cette région, de coordonnées (Pb, Pa) est un point mort.

De même il n'y a aucun chemin de l'état initial $(0, 0)$ vers aucun état de la région marquée "Unreachable".

HDA – formalisation

- Modèle basé sur l'agglomération de points, segments, carrés, . . . ,hyperc
- les "collages" sont spécifiés par des opérateurs bords: d^0, l'opérateur bord "début" et d^1 l'opérateur bord "fin" (généralisation de ceux pour les automates standards).

Dimension un

On considère un segment,

$$0 \xrightarrow{I} 1$$

l'objet de dimension un I a comme bord début $d^0(I) = 0$, et comme bord fin $d^1(I) = 1$.

Dimension 2

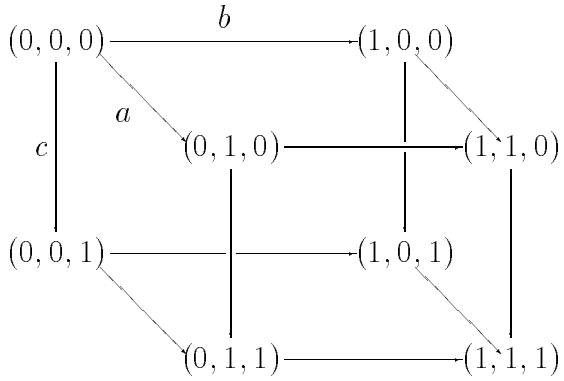
$$\begin{array}{ccc} (0,0) & \xrightarrow{a} & (0,1) \\ {\scriptstyle b}\downarrow & A & \downarrow{\scriptstyle b'} \\ (1,0) & \xrightarrow{a'} & (1,1) \end{array}$$

L'objet de dimension 2 "interieur du carré" A a deux bords début à l'ordre près que l'on se fixe sur $\{a, b\}$, $d_0^0(A) = a$ et $d_1^0(A) = b$ comme de l'état $(0,0)$ on peut faire a et b. De même il a deux bords fin $d_0^1(A) = a'$ car $d_1^1(A) = b'$. On remarque que l'on a

$$d^0(d_1^0(A)) = (0,0) = d^0(d_0^0(A))$$
$$d^1(d_1^0(A)) = (1,0) = d^0(d_0^1(A))$$

Dimension 3

Cela se généralise aisément au cube.



Le cube

Soient A, B et C les faces (respectivement)

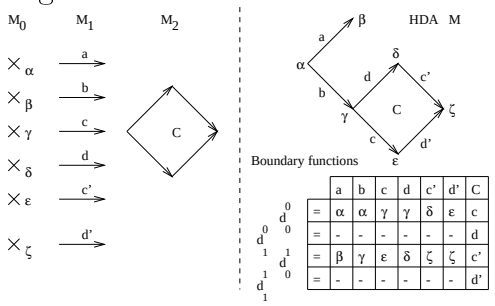
$$((0,0,0),(1,0,0),(0,0,1),(1,0,1))$$
$$((0,0,0),(0,1,0),(0,0,1),(0,1,1))$$
$$((0,0,0),(1,0,0),(0,1,0),(1,1,0))$$

Soient A', B' et C' les faces parallèles à A, B et C respectivement.

On a $d_0^0(D) = A$, $d_1^0(D) = B$, $d_2^0(D) = C$ et $d_0^1(D) = A'$, $d_1^1(D) = B'$, $d_2^1(D) = C'$. Alors $d_0^0(A) = b$, $d_1^0(A) = c$, $d_0^0(B) = a$, $d_1^0(A) = c$, $d_0^0(C) = a$, $d_1^0(C) = b$. On verifie que $d_i^0(d_j^0(D)) = d_{j-1}^0(d_i^0(D))$ pour tous les $i < j$.

Généralisation

Généralisation de ces relations aux hypercubes.
On amalgame avec des relations sur les bords:



	a	b	c	d	c'	d'	C
=	α	α	γ	γ	δ	ε	c
=	-	-	-	-	-	-	d
=	β	γ	ε	δ	ζ	ζ	c'
=	-	-	-	-	-	-	d'

Exemple plus général

On définit un petit langage "PV":

$$Proc_d = \epsilon \mid Pa.Proc_d \mid Va.Proc_d \mid Proc_d + Proc_d \mid Y$$

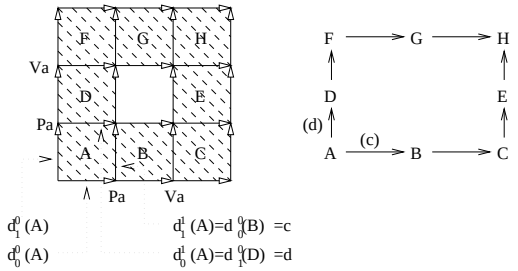
Sémantique

Un environnement $\rho : \mathcal{O} \to \mathbb{N}$ est une fonction dont la valeur pour un objet a représente le nombre de fois que a peut toujours être accédé par les processus.
Un n-rectangle (état du programme) est une paire (C, ρ) avec

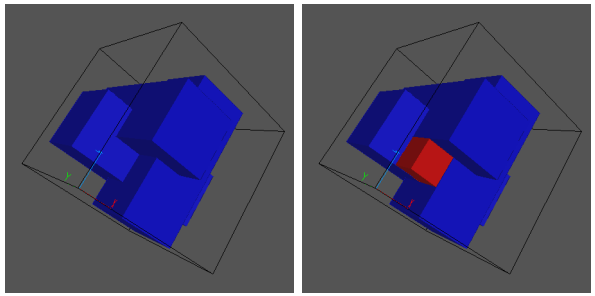
- C élement du langage,
- ρ contexte.

La collection des faces de chaque n-rectangle est separée en n faces départ, $d_i^0(A)$ et n faces fin, $d_i^1(A)$
La relation d'ordre temporel est donnée par la relation "avoir une d^1-face égale à une d^0-face".

Sémantique de $(Pa.Va \mid Pa.Va)$



Autre exemple



Sémantique générale

Deux phases:

1): arbre syntaxique de chaque processus commence par une composition séquentielle de P et de V avec un autre terme.

Ces processus sont: $X_i = Q_i a_i.Y_i$, $1 \leq i \leq k$, où Q_i est P ou V, $a_i \in \mathcal{O}$ et Y_i est un processus. La sémantique de $X_1 \mid \cdots \mid X_k$ dans l'environnement ρ est $[\![X_1 \mid \cdots \mid X_k]\!]\rho$.

Première phase

Si pour tout $a \in \mathcal{O}$, $\rho(a) \geq 0$,

$$[\![X_1 \mid \cdots \mid X_k]\!]\rho = (X_1 \mid \cdots \mid X_k, \rho) + [\![Y_1 \mid X_2 \mid \cdots \mid X_k]\!]\rho_1 + \cdots + [\![X_1 \mid \cdots \mid X_{k-1} \mid Y_k]\!]\rho_k$$

où ρ_i, $1 \leq i \leq k$ tel que $\rho_i(b) = \rho(b)$ pour tout $b \in \mathcal{O}$, $b \neq a_i$, et $\rho_i(a_i) = \rho(a_i) - 1$ si $Q_i = P$ ou $\rho_i(a_i) = \rho(a_i) + 1$ si $Q_i = V$. Si il y a un $a \in \mathcal{O}$, $\rho(a) < 0$,

$$[\![X_1 \mid \cdots \mid X_k]\!]\rho = [\![Y_1 \mid X_2 \mid \cdots \mid X_k]\!]\rho_1 + \cdots + [\![X_1 \mid \cdots \mid X_{k-1} \mid Y_k]\!]\rho_k$$

avec les mêmes environnements ρ_i, $1 \leq i \leq k$.

Deuxième phase

(Elimination des variables de processus)

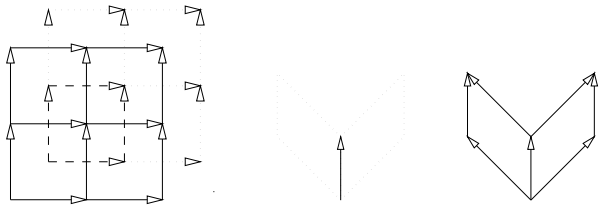
$$[\![X_1 \mid \cdots \mid Y.Y_i \mid \cdots \mid X_k]\!]\rho = [\![X_1 \mid \cdots \mid Proc_Y.Y_i \mid \cdots \mid X_k]\!]\rho$$

(Elimination de plus)

$$[\![X_1 \mid \cdots \mid Y_i + Z_i \mid \cdots \mid X_k]\!]\rho = [\![X_1 \mid \cdots \mid Y_i \mid \cdots \mid X_k]\!]\rho + [\![X_1 \mid \cdots \mid Z_i \mid \cdots \mid X_k]\!]\rho$$

Le plus de l'equation ici est une union en collant la face $(X_1 \mid \cdots \mid Y_i \mid \cdots X_k, \rho, i)$ avec la face $(X_1 \mid \cdots \mid Z_i \mid \cdots X_k, \rho, i)$.

Complexité de la représentation



Plus formellement

$$(I^k)_0 = \{\frac{j}{k},\ 0 \leq j \leq k\}$$

$$(I^k)_1 = \{[\frac{j-1}{k}, \frac{j}{k}],\ 1 \leq j \leq k\}$$

donc, I^k a $k+1$ 0-rectangles et k 1-rectangles.
Soit $(I^k)^n$ le n-rectangle k-subdivisé.
Soit $t_i^{k,n}$ le nombre de i-rectangles dans $(I^k)^n$, alors

$$t_i^{k,n} = C_i^n k^i (k+1)^{n-i}$$

où $C_i^n = \frac{n!}{i!(n-i)!}$

Autre résultat

Soit $S_i^{k,n}$ la classe des sous-complexes connexes générés par des n-rectangles de $(I^k)^n$ tels que quelque soient deux n-rectangles t_1 and t_2, on a $t_1 \bigcap t_2 = t_1 = t_2$ ou $t_1 \bigcap t_2 = \emptyset$ ou $\dim(t_1 \bigcap t_2) = i$. (barrières de synchronisation).

Pour des résultats asymptotiques on considère plutôt $S_i^{k,n}$ construit à partir de $(I^\infty)^n$

Autre Résultat

Soit $r_{i,j}^n$ le ratio du nombre $t_{i,j}^n$ de j-rectangles de $X \in S_i^n$ par $t_{i,n}^n$

$$\text{Pour } 0 \leq j \leq i,\ C_j^n \leq r_{i,j}^n \leq C_j^i 2^{i-j}(2^{n-i}\frac{n!}{i!} - 1)$$

$$\text{Pour } i+1 \leq j \leq n,\ r_{i,j}^n = C_j^n 2^{n-j}$$

Cadre

Soit I l'intervalle unité, et $I^n = I_1 \times \cdots \times I_n$ le cube unité.

On appelle $R = [a_1, b_1] \times \cdots \times [a_n, b_n]$ un *n-rectangle*, et on considère un ensemble $F = \cup_1^r R^i$ de n-rectangles $R^i = [a_1^i, b_1^i] \times \cdots \times [a_n^i, b_n^i]$.

L'intérieur $\overset{\circ}{F}$ de F est la "zone interdite" de I^n; son complément est $X = I^n \setminus \overset{\circ}{F}$. On suppose $\mathbf{0} = (0, \ldots, 0) \notin F$, et $\mathbf{1} = (1, \ldots, 1) \notin F$.

Premières définitions

Définition 15 • 1. Un chemin continu $\alpha : I \to I^n$ est appelé un *chemin orienté* si *toutes* ses compositions $\alpha_i = pr_i \circ \alpha : I \to I$, $1 \leq i \leq n$, sont croissantes : $t_1 \leq t_2 \Leftarrow \alpha_i(t_1) \leq \alpha_i(t_2)$, $1 \leq i \leq n$.

• 2. Un point $y \in X = I^n \setminus \overset{\circ}{F}$ est dans le *futur* $J^+(x)$ d'un point $x \in X$ si il existe un chemin orienté $\alpha : I \to X$ avec $\alpha(0) = x$ et $\alpha(1) = y$. Le passé $J^-(x)$ est défini de façon similaire.

• 3. Un futur immédiat $J_0^+(x)$ de $x \in X$ est de la forme $J^+(x) \cap ([x_1, x_1 + \varepsilon] \times \cdots \times [x_n, x_n + \varepsilon])$ où $\varepsilon < \min\{a_j^i - x_j > 0, b_j^i - x_j > 0,\ 0 \leq i \leq r, 0 \leq j \leq n\}$.

Premières définitions

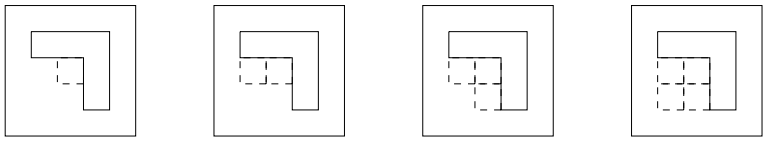
• *4. Un point $x \in I^n \setminus \overset{\circ}{F}$ est appelé* admissible, *si $\mathbf{1} \in J^+(x)$; et* unsafe *sinon.*

• *5. Soit $\mathcal{A}(F) \subset I^n$ la* region admissible *contenant tous les points admissibles de X, et $\mathcal{U}(F) \subset I^n$ la* region dangereuse *contenant tous les points dangereux de X.*

• *6. Un point $x \in X$ est un* point mort *ssi $J^+(x) = \{x\}$.*

Premier Algorithme

Pour trouver les deadlocks, on part de la région interdite puis on complète par les n-rectangles dont toutes les faces sont collées à cette région.



Référence: pour cet algorithme et le suivant, L. Fajstrup, E. Goubault et M. Raussen.

Complexité

Proposition 3 *Pour un terme pur avec n transactions (région interdite $F = \cup_1^r R_i$), la complexité dans le cas le pire d'un tel parcours est de l'ordre de $nVol(F) + \Sigma_1^r Vol(R_i)$.*

Sémantique duale

Nouvelle sémantique pour PV:

Soit $T = X_1 \mid \cdots \mid X_n$ un terme pur tel que tous ses sous-termes sont purs. $X_i(j)$ est la $j$ème lettre de la chaîne X_i. On suppose que la longueur des chaînes X_i $(1 \leq i \leq n)$ sont les entiers l_i, on a

$$[k_1, r_1] \times \cdots \times [k_n, r_n] \in [\![X_1 \mid \cdots \mid X_n]\!]_2$$

s'il y a une partition de $\{1, \cdots, n\}$ en $U \cup V$ avec $card(U) = s(a) + 1$ pour un objet a avec, $X_i(k_i) = Pa$, $X_i(r_i) = Va$ pour $i \in U$ et $k_j = 0$, $r_j = l_j$ avec $j \in V$.

Deuxième algorithme

Soit $R = [a_1, b_1] \times \cdots \times [a_n, b_n]$ un n-rectangle. Son bord $\partial(R)$ se décompose en,

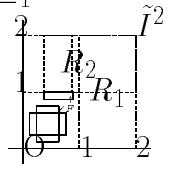
- le *bord inférieur* $\partial_-(R) := \{\mathbf{x} \in R | \forall j : x_j < b_j,\ \exists j : x_j = a_j\}$;
- le *bord supérieur* $\partial_+(R) := \{\mathbf{x} \in R | \forall j : x_j > a_j,\ \exists j : x_j = b_j\}$;
- le *bord intermédiaire* $\partial_\pm(R) := \{\mathbf{x} \in R | \exists j_1, j_2 : x_{j_1} = a_{j_1},\ x_{j_2} = b_{j_2}\}$.

Soit $\overset{\circ}{F} \subset I^n$ la région interdite et soit $X = I^n \setminus \overset{\circ}{F}$. On va supposer dans la suite la propriété:

Si $\overset{\circ}{R^{i_1}} \cap \overset{\circ}{R^{i_2}} \neq \emptyset$, alors $a_j^{i_1} = a_j^{i_2} \Rightarrow a_j^{i_1} = 0$ et $b_j^{i_1} = b_j^{i_2} \Rightarrow b_j^{i_1} = 1$, $1 \leq j \leq n$.

Extension du modèle

On change un peu le modèle (pour des raisons purement mathématiques) $\partial_+(I^n)$ de I^n dans la région interdite. Soit $\tilde{I} = [0, 2]$ et $I^n \subset \tilde{I}^n$. Soit $R^i = [0, 2]^{i-1} \times [1, 2] \times [0, 2]^{n-i}$, $1 \leq i \leq n$, et en changeant les indices de n, $R^{n+1}, \ldots, R^{n+r}$ sont les n-rectangles utilisés dans le modèle précédent F avec si $b_j^i = 1$, on pose $b_j^{i+n} = 2$. Alors $\cup_1^n R^i = \tilde{I}^n \setminus \overset{\circ}{I^n}$, et $\tilde{F} = F \cup \cup R^i = \cup_{i=1}^{n+r} R^i$.



Propriétés

Le *bord* $\partial F \subset F$ se décompose en $\partial F = \partial_- F \cup \partial_+ F \cup \partial_\pm F$ avec $\partial F = (\cup_i \partial R^i) \setminus \overset{\circ}{F}$, $\partial_- F = (\cup_i \partial_- R^i) \setminus \overset{\circ}{F}$, $\partial_+ F = (\cup_i \partial_+ R^i) \setminus \overset{\circ}{F}$ et $\partial_\pm F = (\cup_i \partial_\pm R^i) \setminus \overset{\circ}{F}$.
Quand on examine un chemin orienté partant de $\mathbf{x} \in X$, on peut ne faire attention qu'aux points $\mathbf{x} \in \partial_- F$, comme il n'y a aucune obstruction locale en les autres points:

Lemma 1 *Pour* $\mathbf{x} = (x_1, \ldots, x_n) \in (X \setminus \partial_- F)$, *le futur* $J^+(\mathbf{x})$ *contient un* cône complet $[x_1, x_1 + \varepsilon] \times \cdots \times [x_n, x_n + \varepsilon]$ *avec* $\varepsilon > 0$.
□

Stratification du bord

Pour les points $\mathbf{x} \in \partial_- F$, la structure du futur immédiat $J_0^+(\mathbf{x})$ peut être expliquée en terme de ce que l'on appelle une *stratification du bord*:

Soit $R^i = [a_1^i, b_1^i] \times \cdots \times [a_n^i, b_n^i]$, et pour tous les ensembles d'indices $J = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n + r\}$ on définit

$$R^J = R^{i_1} \cap \cdots \cap R^{i_k}$$

c.a.d.,

$$R^J = [a_1^J, b_1^J] \times \cdots \times [a_n^J, b_n^J]$$

avec $a_j^J = \max\{a_j^i | i \in J\}$ et $b_j^J = \min\{b_j^i | i \in J\}$.

Strates

Cet ensemble est un $n-$rectangle sauf s'il est vide (si $a_j^k > b_j^l$ avec $1 \leq j \leq n$ et $k, l \in J$).

A J on associe

$$\partial_- R^J = \partial_- R^{i_1} \cap \cdots \cap \partial_- R^{i_k}$$

et la *strate du bord* (in $\partial_- F$)

$$\partial_-^J F = R^J \cap \partial_- F = \partial_- R^J \setminus \overset{\circ}{F} .$$

f-adéquation

Un ensemble d'indices $\emptyset \neq J \subseteq \{1, \ldots, n + r\}$ est *f-adéquat* (f pour futur) si $\partial_-^J F \neq \emptyset$, c.a.d., $R^J \neq \emptyset$ et $\mathbf{a}^J \notin \overset{\circ}{F}$.

Lemma 2 *Si* $I \subsetneq J$ *sont tous les deux f-adéquats, alors* $\partial_-^J F \subsetneq \partial_-^I F$; *c.a.d., pour tout* $i \in J$ *il y a au moins une coordonnée telle que* $a_j^J = a_j^i \geq a_j^k$ *pour tout* $k \in J$. □

Stratification induite

En particulier on obtient la *stratification de bord*

$$\partial_- F = \bigcup_{J \text{ f-adéquat}} \partial_-^J F$$

Chaque sous-ensemble f-adéquat $\emptyset \neq J \subseteq \{1, \ldots, n+r\}$ vient avec une *partition* p^J de l'ensemble $\{1, \ldots, n\}$:

$$p^J(i) = \{j | 1 \leq j \leq n,\ a_j^J = a_j^i\}.$$

Donc, $j \in p^J(i)$ ssi $a_j^i = a_j^J = \max\{a_j^k | k \in J\}$.

Résultat

Lemma 3 • *1. Quelquesoit l'ensemble f-adéquat* $\emptyset \neq J \subseteq \{1, \ldots, n+r\}$, p^J *est en fait une partition de* $\{1, \ldots, n\}$, *c.a.d.,*

- $p^J(i) \neq \emptyset \quad \forall i \in J$;
- $p^J(i_1) \cap p^J(i_2) = \emptyset$ *pour* $i_1 \neq i_2$;
- $\cup_{i \in I} p^J(i) = \{1, \ldots, n\}$.

• *2. La stratification de* $\partial_- F$ *peut être décrite comme suit:*

$$\mathbf{x} \in \partial_-^J F \Leftrightarrow \forall i \in J \quad \exists j \in p^J(i) : x_j = a_j^J = a_j^i.$$

Donc,

$\mathbf{x} \in \partial_-^J F$ *ssi* x_j *est minimal dans* R^J *pour au moins un* $j \in p^J(i)$.

Preuve

1. Pour tout j, $0 \leq j \leq n$ il y a un unique $i \in J$ tel que $a_j^i = a_j^J$.
2. Un point $\mathbf{x} = (x_1, \ldots, x_n)$ est contenu dans $\partial_- R^i$ ssi $x_j = a_j^i$ pour au moins un $1 \leq j \leq n$.

Description du futur local

Proposition 4 *Soit* $\mathbf{x} \in \partial_-^J F$. *Alors* $J_0^+(\mathbf{x}) \subset \partial_-^J F$*:*

$\mathbf{y} = (y_1, \ldots, y_n) \in J_0^+(\mathbf{x}) \Rightarrow$: $\quad \forall i \in J\ \exists j \in p^J(i) : x_j = y_j = a_j^i$.

Preuve. Un point $\mathbf{y} > \mathbf{x}$ avec $\varepsilon + x_j > y_j > a_j^i$ *pour tout* $j \in p^J(i)$ est contenu dans $\overset{\circ}{R^i} \subset \overset{\circ}{F}$. □

Points Morts

Les points morts sont les points $\mathbf{x} \in \partial_- F$ avec $J^+(\mathbf{x}) = J_0^+(\mathbf{x}) = \{\mathbf{x}\}$.

Proposition 5 *Un point* $\mathbf{x} \in \partial_- F$ *est un point mort ssi* $\mathbf{x} \neq \mathbf{1}$ *et il existe un ensemble d'indices f-adéquat à* n éléments $J = \{i_1, \ldots, i_n\}$, *et* $\mathbf{x} = \mathbf{a}^J = [a_1^J, \ldots, a_n^J] = \min(R^{i_1} \cap \cdots \cap R^{i_n})$. *Dans ce cas,* $\partial_-^J(F)$ *est constitué de l'unique point* $\{\mathbf{a}^J\}$.

Preuve

On a,

1. $|J| = n \Rightarrow \forall i_k \in J : \quad |p^J(i_k)| = 1$;
2. $|J| < n \Rightarrow \exists i_k \in J : \quad |p^J(i_k)| > 1$.

Dans le cas 1., le lemme 3.2 nous dit que $\partial_-^J F$ est soit vide soit l'ensemble contenant uniquement l'élément $\mathbf{x} = \mathbf{a}^J$. Dans ce dernier cas, par la Proposition plus haut, le futur immédiat $J_0^+(\mathbf{x}) \subset \partial_-^J F = \{\mathbf{x}\}$, c.a.d., $\mathbf{x}$ est un point mort.

Preuve

Dans le cas 2., soit $\mathbf{x} \in \partial_-^J F$ pour $J = \{i_1, \ldots, i_k\}$ et $k < n$. Alors $\mathbf{x} \notin \partial_- R^i$ pour $i \notin J$.

Supposons que $\{1, 2\} \subseteq p^J(s)$ et que $x_1 = a_1^J$. Alors pour un $\varepsilon > 0$, le segment de $\mathbf{x} = [a_1^J, x_2, \ldots, x_n]$ à $[a_1^J, x_2 + \varepsilon, x_3, \ldots, x_n] > \mathbf{x}$ *n'est pas* contenu dans l'intérieur d'un des R^i de F.

Il est en fait contenu dans le futur immédiat $J_0^+(\mathbf{x}) \subset \partial_-^J(F)$, et donc $\mathbf{x}$ ne peut pas être un point mort.

Absence de points morts

Corollaire 2 *Une région interdite* $F = \cup_1^{n+r} R^i \subset I^n$ *a un complément* $X = I^n \setminus F$ *sans point mort ssi pour tout ensemble d'indices* $J = \{i_1, \ldots, i_n\}$ *avec* $|J| = n$

$$R^J = R^{i_1} \cap \cdots \cap R^{i_n} = \emptyset \text{ ou } R^J = \{1\} \text{ ou } \min R^J \in \overset{\circ}{F}.$$

□

Description de la région dangereuse

Soit $J = \{i_1, \ldots, i_n\} \subset \{1, \ldots, n+r\}$ un ensemble d'indices à n éléments avec $\partial_-^J(F) = \{\mathbf{a} = (a_1^J, \ldots, a_n^J) = (a_1^{i_1}, \ldots, a_n^{i_n}) = \min R^J, \}$, c.a.d., $\mathbf{a}$ est un point mort. Pour tout $1 \leq j \leq n$, on prend $\widetilde{a_j^J}$ qui est le "deuxième plus grand" des $a_j^{i_k}$, c.a.d.,

$$\widetilde{a_j^J} = a_j^{i_s} \text{ with } a_j^{i_k} \leq a_j^{i_s} < a_j^J \text{ pour } a_j^{i_k} \neq a_j^J.$$

On associe à $\mathbf{a}$ le n-rectangle $U_{\mathbf{a}} = [\widetilde{a_1^J}, a_1^J] \times \cdots \times [\widetilde{a_n^J}, a_n^J]$.

Proposition

Proposition 6 *Le $n-$rectangle*

$$U_{\mathbf{a}} \setminus \partial_-(U_{\mathbf{a}}) =]\widetilde{a_1^J}, a_1^J] \times \cdots]\widetilde{a_n^J}, a_n^J]$$

demi-ouvert est dangereux, *c.a.d., tout chemin orienté dams I^n d'un point $\mathbf{x} \in (U_{\mathbf{a}} \setminus \partial_-(U_{\mathbf{a}}))$ va entrer dans $\overset{\circ}{F}$.*

Preuve

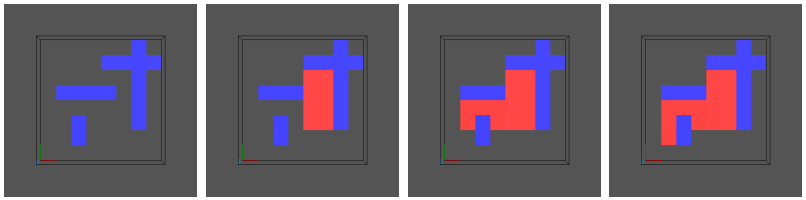
Preuve. Tout chemin orienté commençant dans $U_{\mathbf{a}}$ rencontre $\overset{\circ}{U}_{\mathbf{a}} \cap \overset{\circ}{F}$ ou quitte $U_{\mathbf{a}}$ par $\partial_+(U_{\mathbf{a}})$.
Dans le dernier cas, Il contient un élément $\mathbf{b} = [b_1, \ldots, b_i, \ldots, b_n]$ avec $b_j > \widetilde{a_j^J}$ pour tout j, et de plus, $b_i > a_i^J = a_i^{i_s}$ pour un $i_s \in J$.

En particulier, $\mathbf{b} \in \overset{\circ}{R^{i_s}} \subset \overset{\circ}{F}$. □

Algorithme de calcul de la région dangereuse

- Trouver l'ensemble $\mathcal{D}$ des points morts de X et pour tous les points morts $\mathbf{a} \in \mathcal{D}$, le n-rectangle dangereux $U_{\mathbf{a}}$.
- Soit $F_1 = F \cup \cup_{\mathbf{a} \in \mathcal{D}} U_{\mathbf{a}}$.
- Trouver l'ensemble $\mathcal{D}_1$ des points morts de $X_1 = X \setminus F_1 \subset X$, et pour tous les points morts $\mathbf{a} \in \mathcal{D}_1$, le n-rectangle dangereux $U_{\mathbf{a}}$.
- Soit $F_2 = F_1 \cup \cup_{\mathbf{a} \in \mathcal{D}_1} U_{\mathbf{a}}$ etc.

Exemple



Trois itérations au lieu de 26 (pour le “premier” algorithme).

Implémentation

- L'implémentation utilise une pile: `pile[0],...,pile[n-1]` (`n` étant la dimension du problème).
- Pour calculer l'effet d'un ajout d'un n-rectangle S le programme appelle la procédure `complete(S,`$\emptyset$`)`. Elle appelle une fonction auxiliaire `derive`.
- L'union de la région interdite et de la région dangereuse se retrouve dans `pile[0]`.

“Complete”

```
complete(S,l)
   Si S est inclus dans un X de pile[0] return
   for i=n-2 to 0 by -1 do
      pile[i+1]=intersection(pile[i]\l,S)
   pile[0]=union(pile[0],S)
   for all X in pile[n-1] do
      pile[n-1]=pile[n-1]\X
      derive(X)
```


“Derive”

```
derive(X)
   for all i do
      yi=max({Xj(i) / j=1,...,n}\{X(i)})
   Y=[y1,X(1)]x...x[yn,X(n)]
   Si Y n'est pas inclus dans un des Xj
      complete(Y,(X1,...,Xn))
```

Complexité de l'algorithme

- Soit n le nombre de processus et r le nombre de n-rectangles.
- `pile` nécessite le calcul de $S(r,n) = \Sigma_{i=1}^{n} C_i^r$ intersections, chacune d'entre elles nécessitant n coordonnées.
- Pour trouver les points morts on doit comparer (n coordonnées de) au plus C_n^r éléments non-vides de `pile[n]` avec les r éléments de `pile[0]`.
- Le pire cas peut donc être estimé à $S(r,n) \leq 2^r$ pour tout n, et $S(r,n) \leq nC_n^r$ pour $r > 2n$

Exemple

```
/* 3 philosophers ``3phil'' */
A=Pa.Pb.Va.Vb
B=Pb.Pc.Vb.Vc
C=Pc.Pa.Vc.Va
```


Résultat

```
(P(b).V(a).V(b)|P(c).V(b).V(c)|P(a).V(c).V(a),
                              [c,0][b,0][a,0])
```


Exemple

```
/* 5 philosophers ``5phil'' */
A=Pa.Pb.Va.Vb
B=Pb.Pc.Vb.Vc
C=Pc.Pd.Vc.Vd
D=Pd.Pe.Vd.Ve
E=Pe.Pa.Ve.Va
```

Résultat

```
(P(b).V(a).V(b)|P(c).V(b).V(c)|P(d).V(c).V(d)|
P(e).V(d).V(e)|P(a).V(e).V(a),[e,0][d,0][c,0][b,0][a,0])
```


Exemple

```
/* ``example'' */
A=Pa.Pb.Vb.Pc.Va.Pd.Vd.Vc
B=Pb.Pd.Vb.Pa.Va.Pc.Vc.Vd
```


Résultat

```
(P(b).V(b).P(c).V(a).P(d).V(d).V(c)|P(d).V(b).P(a).
          V(a).P(c).V(c).V(d),[d,1][c,1][b,0][a,0])+
(P(b).V(b).P(c).V(a).P(d).V(d).V(c)|V(b).P(a).V(a).
               P(c).V(c).V(d),[d,0][c,1][b,0][a,0])+
(P(b).V(b).P(c).V(a).P(d).V(d).V(c)|P(a).V(a).P(c).
                     V(c).V(d),[d,0][c,1][b,1][a,0])+
(V(b).P(c).V(a).P(d).V(d).V(c)|P(a).V(a).P(c).V(c).
                           V(d),[d,0][c,1][b,0][a,0])+
(P(c).V(a).P(d).V(d).V(c)|V(b).P(a).V(a).P(c).V(c).
                           V(d),[d,0][c,1][b,0][a,0])+
(P(c).V(a).P(d).V(d).V(c)|P(a).V(a).P(c).V(c).V(d),
                                [d,0][c,1][b,1][a,0])+
```

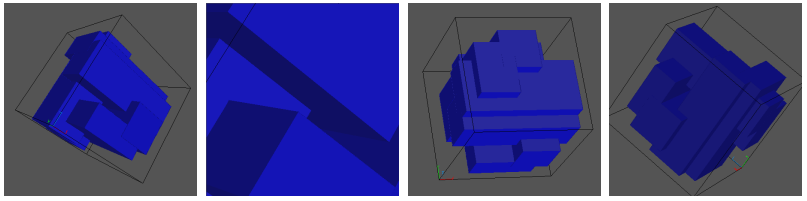

Suite...

```
(V(a).P(d).V(d).V(c)|V(b).P(a).V(a).P(c).V(c).V(d),
                                [d,0][c,0][b,0][a,0])+
(V(a).P(d).V(d).V(c)|P(a).V(a).P(c).V(c).V(d),
                                [d,0][c,0][b,1][a,0])+
(P(d).V(d).V(c)|V(b).P(a).V(a).P(c).V(c).V(d),
                                [d,0][c,0][b,0][a,1])+
(P(d).V(d).V(c)|P(a).V(a).P(c).V(c).V(d),[d,0][c,0]
                                        [b,1][a,1])+
(V(a).P(d).V(d).V(c)|P(c).V(c).V(d),[d,0][c,0][b,1][a,0])
(P(d).V(d).V(c)|V(a).P(c).V(c).V(d),[d,0][c,0][b,1][a,0])
(P(d).V(d).V(c)|P(c).V(c).V(d),[d,0][c,0][b,1][a,1])
```

Exemple

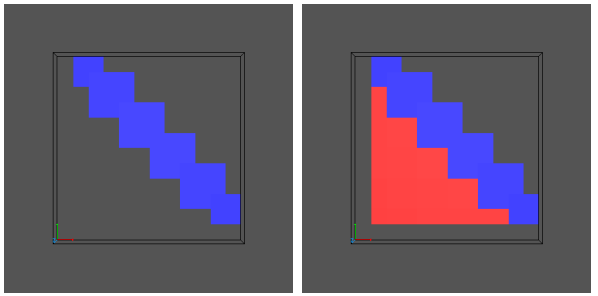
```
/* ``lipsky'' */
A=Px.Py.Pz.Vx.Pw.Vz.Vy.Vw
B=Pu.Pv.Px.Vu.Pz.Vv.Vx.Vz
C=Py.Pw.Vy.Pu.Vw.Pv.Vu.Vv
```


Résultat



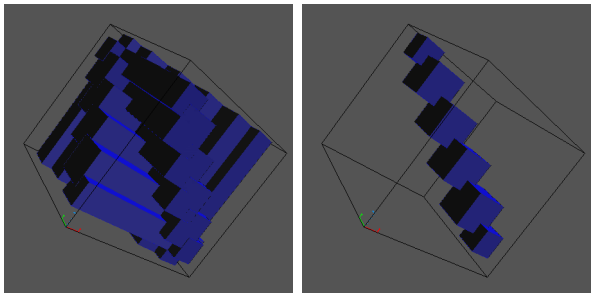
Exemple

```
/* ``stair2'' */
A=Pa.Pb.Va.Pc.Vb.Pd.Vc.Pe.Vd.Pf.Ve.Vf
B=Pf.Pe.Vf.Pd.Ve.Pc.Vd.Pb.Vc.Pa.Vb.Va
```



Exemple

```
/* ``stair3'' */
A=Pa.Pb.Va.Pc.Vb.Pd.Vc.Pe.Vd.Pf.Ve.Vf
B=Pf.Pe.Vf.Pd.Ve.Pc.Vd.Pb.Vc.Pa.Vb.Va
C=Pf.Pe.Vf.Pd.Ve.Pc.Vd.Pb.Vc.Pa.Vb.Va
```



Et les invariants?

Il faut examiner les propriétés du modèle (produit synchronisé?).

On va lister les principales propriétés d'un de ces modèles

Définition

Définition 16 *Un HDA semi-régulier est une collection d'ensembles* M_n *(*$n \in \mathbb{N}$*) et de fonctions*

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

pour tout $n \in \mathbb{N}$ *et* $0 \leq i, j \leq n-1$*, telles que* $d_i^k \circ d_j^l = d_{j-1}^l \circ d_i^k$ $(i < j, k, l = 0, 1)$ *et* $\forall n, m\, n \neq m,\ M_n \cap M_m = 0$.

Les éléments x de M_n ($\dim x = n$) sont appelés des n-transitions (ou états si $n = 0$).

Morphismes

Définition 17 *Soient* M *et* N *deux HDA semi-réguliers, et* f *une famille* $f_n : M_n \to N_n$ *de fonctions.* f *est un morphisme de HDA semi-réguliers ssi*

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

pour tout $n \in \mathbb{N}$.

Cela définit la catégprie Υ_{sr} de HDA semi-réguliers.

Troncation

Foncteur troncation $T_n : \Upsilon_{sr} \to \Upsilon_{sr}^n$ defini par, $T_n(M)_m = M_m$ ssi $m \leq n$ et $T_n(M)_m = \emptyset$ si $m > n$.

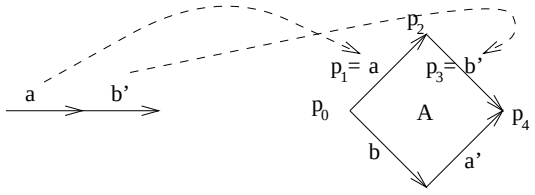
Son effet est de restreindre l'exécution à n processeurs.

Chemins

Définition 18 *Un chemin dans un HDA semi-régulier M est $p = (p_0, \ldots, p_n)$ tel que p_0 et p_n sont des états et*

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

Exemple



Propriétés catégoriques

Proposition 7 *Υ_{sr} est un* topos élémentaire. *Il est complet co-complet.*

Preuve. Soit $\square$ la catégorie libre dont les objets sont $[n]$, où $n \in \mathbb{N}$, et dont les morphismes sont engendrés par

$$[n] \overset{\delta^0_i}{\underset{\delta^1_j}{\rightrightarrows}} [n-1]$$

pour tout $n \in \mathbb{N}^*$ et $0 \leq i, j \leq n-1$, tels que

Preuve

$$\delta^k_i \delta^l_j = \delta^l_{j-1} \delta^k_i \quad (i < j)$$

Maintenant, la catégorie $\square$**Set** des foncteurs de $\square$ vers **Set** (les morphismes sont les transformations naturelles) est isomorphe à Υ_{sr}. C'est donc un topos élémentaire. Elle est complète et co-complète parce-que **Set** est complète et co-complète. $\square$

Description combinatoire

Soit $D_{[n]}$ le HDA semi-régulier $Hom_{\square}([n], .)$.

Définition 19 *Un n-cube singulier d'un HDA M est un morphisme $\sigma : D_{[n]} \to M$.*

Un lemme préparatoire

Lemma 4 *L'ensemble des n-cubes singuliers d'un HDA semi-régulier M est en correspondance bi-univoque avec M_n. L'unique n-cube singulier correspondant à un n-cube $x \in M_n$ est appelé $\sigma_x : D_{[n]} \to M$. C'est l'unique n-cube singulier σ tel que $\sigma(Id_{[n]}) = x$.*

Preuve. Par le lemme de Yoneda. Υ_{sr} isomorphe à la categorie de foncteurs de $\square$ dans **Set**. Donc $D_{[n]}$ sont les foncteurs représentables et $Nat(D_{[n]}, M) \cong M([n])$, où M est un foncteur de $\square$ vers **Set**. Cela se traduit en $\Upsilon_{sr}(D_{[n]}, M) \cong M_n$. $\square$

Une proposition préparatoire

Proposition 8 *Soit M un HDA semi-régulier. Le diagramme suivant est co-cartesien (pour $n \in \mathbb{N}$),*

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

où $\dot{D}_{[n+1]} = T_n(D_{[n+1]})$ et $\dot{\sigma} x = \sigma x_{|\dot{D}_{[n+1]}}$.

Preuve

Preuve. Il suffit de prover que le diagramme suivant (dans la categorie des ensembles) est co-cartesien pour tout $p \leq n+1$,

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

comme les co-limites (donc les pushouts) sont calculées point par point dans une catégorie de foncteurs dans **Set**.

Preuve

Pour tout $p < n+1$, les inclusions sont en fait des bijections, et le diagramme est alors co-cartesien de manière évidente.

Pour $p = n+1$, le complément de $\coprod_{x \in \dot{M}_{n+1}} (\dot{D}_{[n+1]})_p$ dans $\coprod_{x \in \dot{M}_{n+1}} (D_{[n+1]})_p$ est l'ensemble des copies de cubes $Id_{[n+1]}$, une pour chaque cube de M_{n+1}. Cela signifie que la fonction $\coprod_{x \in \dot{M}_{n+1}} (\sigma_x)_p$ induit une bijection du complément de $\coprod_{x \in \dot{M}_{n+1}} (\dot{D}_{[n+1]})_p$ dans le complément de $(T_n(M))_p$. Cela implique que le diagramme est co-cartesien pour $p = n+1$ également. □

Signification

Aller du squelette de dimension n au squelette de dimension $n+1$ c'est:

- rajouter des relations d'indépendance,
- ou de façon duale, supprimer des exclusions mutuelles.

Cf. les systèmes de transition asynchrones etc.

Constructions catégoriques

Le produit cartesien de deux HDA semi-réguliers F, G est le HDA semi-régulier $F \times G$ avec

$$(F \times G)_n = F_n \times G_n$$
$$d^\epsilon_k[F \times G] = d^\epsilon_k[F] \times d^\epsilon_k[G]$$

C'est le produit complètement synchronisé des deux automates.

Coproduit

Le coproduit de deux HDA F, G est

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

C'est l'opérateur choix non-déterministe.

Foncteur Hom

L'adjoint à droite $\Rightarrow$ du produit cartésien est,

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

et pour $f \in G \Rightarrow H([n])$,

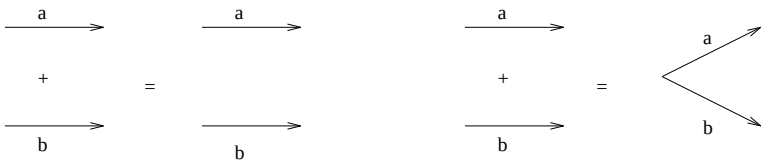
$$G \Rightarrow H(\delta^\epsilon_k)(f) : D_{[n-1]} \times G \longrightarrow H$$
$$(u, v) \longrightarrow f(u \circ \delta^\epsilon_k, v)$$

(Nat est l'ensemble des transformations naturelles). Donc,

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

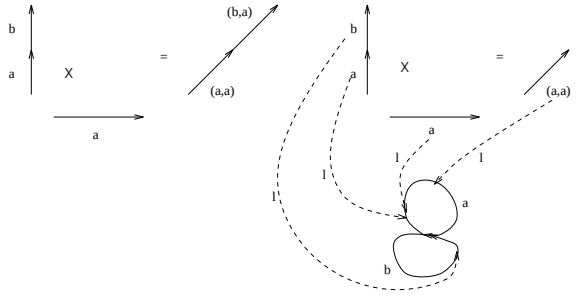
$$d^\epsilon_k[G \Rightarrow H](f) : D_{[n-1]} \times G \longrightarrow H$$
$$(u, v) \longrightarrow f(u \circ \delta^\epsilon_k, v)$$

Exemple



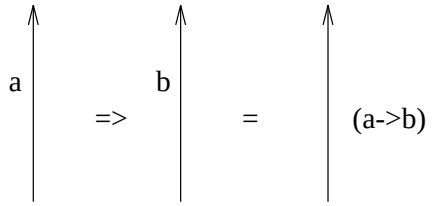
Coproduit et somme amalgamée

Exemple



Produit cartésien et produit fibré.

Exemple



Espace de fonction "synchrone"

Autres propriétés

Proposition 9 *Υ_{sr} est une catégorie monoidale fermée.*

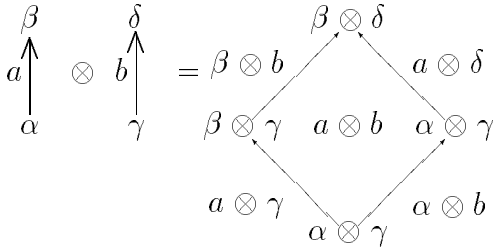
Schéma de preuve. On construit $F \otimes G$ pour représenter la composition parallèle sans interférence,

$$(F \otimes G)_n = \cup_{i+k=n} F_i \times G_k$$

et pour $x \in F_i, y \in G_k$,

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

Exemple



Flèche linéaire

Le produit tensoriel a un adjoint à droite car il commute avec les colimites qui est encore donné par le lemme de Yoneda. C'est

$$(G \multimap H)_n = \{f : D_{[n]} \otimes G \to H / f \text{ est un morphisme}\}$$

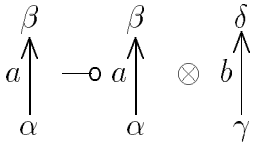
et pour $f \in (G \multimap H)_n, d^\epsilon_k[G \multimap H](f) : D_{[n-1]} \otimes G \longrightarrow H$

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

Dans $G \multimap H$ on a les fonctions qui créent d'autres processus (comme `pvm_spawn`).

Exemple

Dans



Il y a par exemple une 1-transition "spawn l'action b"

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

Propriétés fines

Sur les traces en général et non plus uniquement sur les états.

Quelle complexité?

Deux propriétés classiques

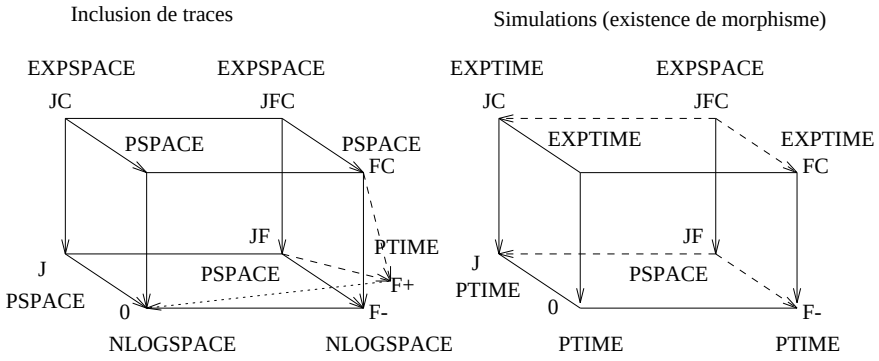
- Le problème d'inclusion des ensembles de traces,
- Le problème de simulation (et son inverse la relation d'implémentation): Est-ce qu'un automate simule ou est simulé par un autre automate?

Référence: D. Harel, O. Kupferman et M. Y. Vardi (On the Complexity of Verifying Concurrent Transition Systems).

Résultats

- Dans ce qui suit F veut dire que l'on considère des systèmes de transition équitables, C veut dire que l'on considère des systèmes parallèles (résultats pour CCS ou CSP), J représente l'union des complexités,
- les flèches pleines représentent un bond d'une exponentielle dans les classes de complexité,
- les flèches hachurées représentent le passage d'une classe de complexité en espace à la même classe de complexité en temps,
- les lignes en pointillé représentent le passage d'une classe de complexité en temps vers la classe de complexité en espace qu'elle contient.

Résultats



Transactions parallèles (suite)

Transactions parallèles dans une base de donnée, on protège chaque accès par des `LOCK`.

Problème:

```
P: LOCK A,B          Q: LOCK B
   A:=B+1;              B:=B+2;
   UNLOCK A,B           UNLOCK B
   LOCK B               LOCK A,B
   B:=3;                A:=2*B;
   UNLOCK B             UNLOCK A,B
```


Traces

En partant des valeurs initiales `A=0`, `B=0`,

```
P: A=1   P: A=1   P: A=1   Q: B=2   Q: B=2   Q: B=2
P: B=3   Q: B=2   Q: B=2   P: A=3   P: A=3   Q: A=6
Q: B=5-  P: B=3-  Q: A=4-  Q: A=4-  P: B=3-  P: A=3-
Q: A=10- Q: A=6-  P: B=3-  P: B=3-  Q: A=6-  P: B=3-
```


Problème

Seules la première (avec résultat `A=10, B=5`) et la dernière trace (avec résultat `A=3, B=3`) sont correctes. Les autres traces sont des interférences non souhaitées:

On veut que toutes les traces d'eécution donnent le même résultat qu'une trace *séquentielle*, i.e. que `P` puis `Q` ou `Q` puis `P` dans leurs totalité. C'est la propriété de *séquentialisation*.

Une solution - 2-phase locking

Pour tout processus P voulant accéder à la base de donnée, on fait tous les LOCK, le corps du processus, puis tous les UNLOCK

Exemple

```
P:                   Q:
   LOCK A;              LOCK A;
   LOCK B;              LOCK B;
   A:=B+1;              B:=2;
   B:=3;                A:=2*B;
   UNLOCK B;            UNLOCK B;
   UNLOCK A;            UNLOCK A;
```

Homotopie

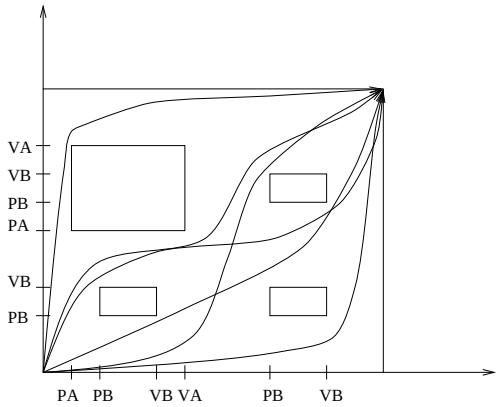
- Un chemin est une fonction continue $\alpha : I = [0,1] \to \mathbb{R}^n$,
- Deux chemins α, β sont homotopes ssi il existe une fonction continue $F : I \times I \to \mathbb{R}^n$ telle que $F(x,0) = \alpha(x)$ et $F(x,1) = \beta(x)$ et $F(0,t) = F(0,0) = F(1,0)$ et $F(1,t) = F(1,0) = F(1,1)$.

Fait

Théorème 4 *Deux chemins croissants (traces) homotopes accèdent aux objets dans le même ordre.*

Corollaire 3 *Deux traces homotopes décrivent les mêmes modifications sur les objets partagés.*

Exemple



2-phase locking

Théorème 5 *Le 2-phase locking est séquentialisable (donc correct).*

Preuve de J. Gunawardena

Schéma de preuve. La condition de 2-phase locking implique qu'il existe pour chaque transaction T_i, $1 \leq i \leq n$ un nombre $c_i \in I$ tel que c_i soit un temps local entre tous les `LOCK` et tous les `UNLOCK`.

Alors on peut voir que $(c_1, \cdots, c_n)$ est dans le centre de la région interdite.

En effet, soit a un objet partagé par les transactions T_j, $j \in S \subseteq \{1, \cdots, n\}$, le n-rectangle de coin inf $\overline{a}$ et de coin sup $\overline{b}$ avec,

Preuve

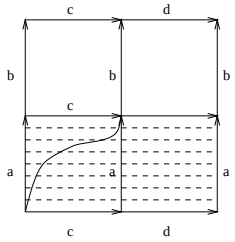
$$a_k = \begin{cases} 0 & \text{si } k \notin S \\ LOCK\, a & \text{sinon} \end{cases} \quad \text{et } b_k = \begin{cases} 1 & \text{si } k \notin S \\ UNLOCK\, a & \text{sinon} \end{cases}$$

est contenu dans chaque n-rectangle interdit pour l'objet a. $c = (c_1, \cdots, c_n)$ est inclus dans chacun de ces n-rectangles de façon évidente. De plus pour tout point p de la région interdite F, le segment $[c, p]$ est inclus dans F. F est "étoilée".

Parce que F est étoilée, la projection radiale de centre c sur les bords de I^n est une homotopie qui envoie toutes les traces qui ne vont pas dans des points morts sur un chemin séquentiel.

Un argument supplémentaire permet de transformer ces chemins (non forcément croissants) en des vraies traces séquentielles.

Homotopie et séquentialisation



PROPERTIES

ac ~ ca **a, c confluent (asynchronous)**

ad ~ da **a, d confluent (independent)**

a < c or c < b

b < d or d < b

a < b and c < d

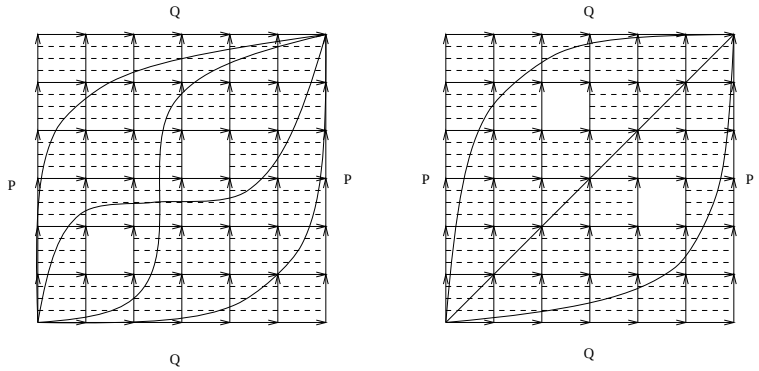
Ordonnanceurs

On peut s'intéresser ensuite aux "ordonnanceurs" généraux et à leurs calculs approximés.

Applications à la séquentialisation, aux protocoles de systèmes distribués tolérants aux pannes, à l'exclusion mutuelle etc.

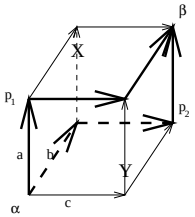
Référence: E. Goubault (Schedulers as abstract interpretations of HDA)

Classes d'homotopie



Calcul approximé avec de l'homologie.

En dimension supérieure



X = three faces above and behind
(a or b) < (a or b or c)
Y = three faces below and in front
(b or c) < (a or b or c)

Si le 3-cube est plein alors X peut être déformé en Y
"X et Y sont 2-processeurs équivalent"