Réduction d’ordre partiel

Motivation

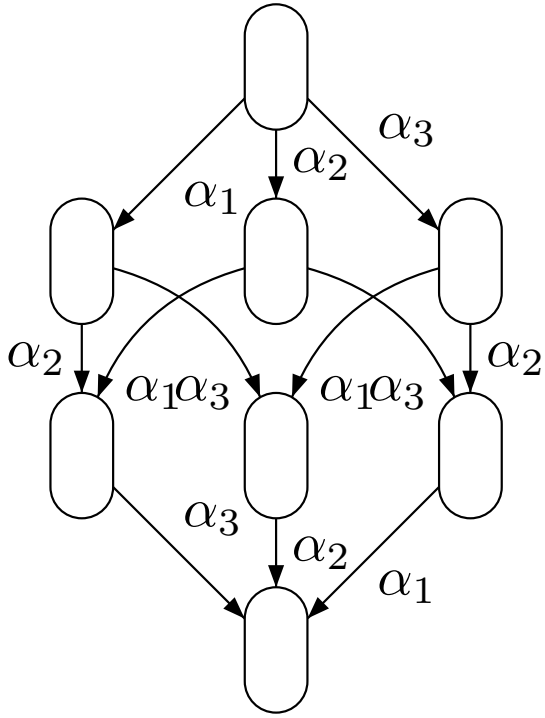
- But: réduire la taille de l'espace des états parcourus par les algorithmes de vérification en exploitant la commutativité des transitions concurrentes.
- Remarque: "réduction d'ordre partiel" devrait plutôt s'appeler "vérification par représentants".

Concurrence dans les systèmes asynch

- Les ordonnancements différents d'évènements concurrents indépendants est l'une des sources de l'explosion combinatoire.
- Exemple: n transitions concurrentes indépendantes.
  - $n!$ ordres possibles.
  - 2^n états distincts à considérer.

Illustration

- Illustration: exécution de trois transitions indépendantes.



Etiquetage des relations de transition

- Dans ce qui suit on les transitions sont étiquetées.
- Si on veut être formel, on ne parlera pas de structure de Kripke mais de structures de Kripke étendue, i.e. $M = (S, S_0, T, L)$ avec
  1. S: un ensemble (fini) d'états.
  2. $S_0 \subset S$: un ensemble d'états initiaux.
  3. T: un $ensemble$ de relations de transitions. Chaque $\alpha \in T$ est une partie de $S \times S$.
  4. $L : S \to 2^{AP}$: une fonction d'étiquetage des états par les propositions atomiques.
- Un chemin de M partant de s est une suite $\pi : s_0 \longrightarrow^{\alpha_0} s_1 \longrightarrow^{\alpha_1} \ldots$ telle que $s_0 = s$ et $\alpha_i(s_i, s_{i+1})$ pour tout i.
- Remarque: à une structure de Kripke modifiée (S, S_0, T, L) correspond une structure de Kripke (S, S_0, R, L). Prendre R tel

Transitions

- Une transition $\alpha \in T$
  1. est dite *active* dans un état s s'il existe s' avec $\alpha(s, s')$.
  2. est dite *déterministe* si pour tout état s, il existe au plus un état s' avec $\alpha(s, s')$. Dans ce cas, on note $s' = \alpha(s)$.
- On note par $enabled(s)$ les transitions actives en l'état s.
- A partir de maintenant, on ne considère plus que des systèmes dont toutes les transitions sont déterministes.

Principe

- Principes:
  1. construire un graphe d'états réduits par une recherche en profondeur,
  2. tel que l'on puisse vérifier la propriété sur le graphe réduit.

- Dans cet exposé: **on s'intéresse aux propriétés** $LTL - \circ$**.**
  1. I.e. on construit une structure de Kripke par une recherche en profondeur,
  2. telle qu'elle soit équivalente au système initial pour la clôture par bégaiement de l'équivalence de trace.

Équivalence induite par $LTL - \circ$

- Deux structures de Kripke $M = (S, S_0, R, L)$,$M' = (S', S_0, R', L')$ sont équivalents par la clôture par bégaiement de l'équivalence de traces (ou plus simplement équivalentes pour $LTL - \circ$) si
  1. M et M' ont même états initiaux,
  2. pour tout chemin $\sigma = s_0 s_1 \ldots$ de M avec $s_0 \in S_0$, il existe un chemin $\sigma' = s'_0 s'_1 \ldots$ de M' avec $s'_0 \in S'_0$ avec σ et σ' équivalents par bégaiement.
  3. et symétriquement.
- Conséquence: soient M, M' deux structures équivalentes pour $LTL - \circ$:
  Alors pour toute formule $f \in LTL - \circ$, $M \models f$ ssi $M' \models f$.

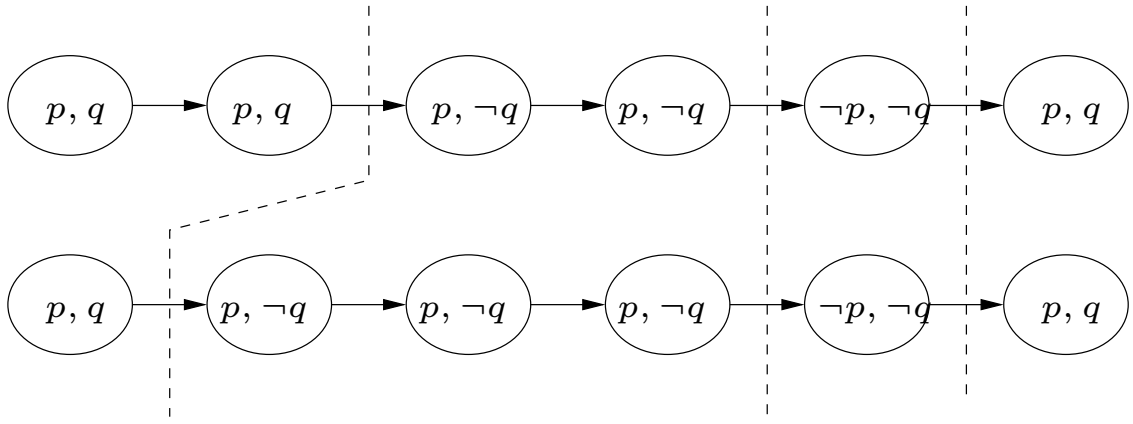


Figure 1: Chemins équivalents par bégaiement

- Algorithme Recherche en Profondeur:
  - $onstack[s_0] := true$
  - $marque[s_0] := 1$
  - appeler $expandstate(s_0)$
- avec Procédure $expandstate(s)$

  $workset[s] = ample(s)$

  tant que $workset[s] \neq \emptyset$

    choisir $\alpha \in workset[s]$ et faire

    $workset := workset - \{\alpha\}$

    $s' := \alpha(s)$

    si $mark[s'] = 0$

      $mark[s'] := 1$

      $onstack[s'] := true$

      $expandstate(s')$

    $createedge(s, \alpha, s')$

  $completed[s] := true$

- où l'on utilise:

1. $mark[s]$: booléen mis à vrai lorsqu'on va traiter l'état s. Initialement $mark[s] = 0$ pour tout s.

2. $onstack[s]$: booléen mis à vrai la première fois que l'on rencontre l'état s.
3. $completed[s]$: booléen mis à vrai lorsque l'on a fini de traiter l'état s.
4. $workset[s]$: ensemble de transitions restant à traiter en s.

5. $ample(s)$: fonction retournant un sous-ensemble de $enabled(s)$.

6. $createedge(s, \alpha, s')$: procédure qui crée une transition α dans le graphe réduit entre l'état s et l'état s'.

Difficultés

- Choix de $ample(s)$:
  1. si on prend $ample(s) = enabled(s)$ pour tout s, le graphe réduit obtenu est simplement une copie du sous-graphe atteignable.
  2. si on prend $ample(s) \subset enabled(s)$, on peut espérer réduire la taille du graphe obtenu.

- Pour que l'algorithme soit correct, il faut
  1. que l'utilisation de $ample(s)$ au lieu de $enabled(s)$ explore assez d'états pour que l'algorithme de vérification donne une réponse correcte sur le graphe réduit.
  2. que l'utilisation de $ample(s)$ au lieu de $enabled(s)$ réduise la taille du graphe.
  3. que le coût du calcul de $ample(s)$ soit faible pour gagner en performance.

Relation d'indépendance

- Une relation d'indépendance est une relation $I \subset T \times T$ symétrique, antiréfléxive telle que pour tout α, β avec $I(\alpha, \beta)$,
  1. $\alpha, \beta \in enabled(s)$ implique $\alpha \in enabled(\beta(s))$,
  2. $\alpha, \beta \in enabled(s)$ implique $\alpha(\beta(s)) = \beta(\alpha(s))$.
- Deux transitions α et β sont dites dépendantes si elle ne sont pas indépendantes.

- Illustration:

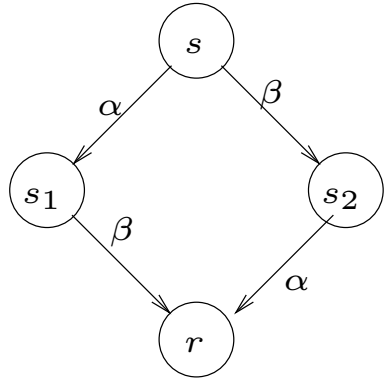


Figure 2: Exécution de deux transitions indépendantes

Problèmes

- Problèmes:
  1. la propriété du système que l'on cherche à vérifier peut dépendre du choix entre l'état s_1 et l'état s_2 de l'exploration.
  2. les états s_1 et s_2 peuvent avoir d'autres successeurs autres que r qui peuvent être éliminés si s_1 ou s_2 sont éliminés.

  $\leadsto$ notion d'invisibilité.

Notion d'invisibilité

- Notion d'invisibilité:
- Soit AP' l'ensemble des propositions atomiques qui apparaissent dans propriété $LTL - \circ$ que l'on cherche à vérifier sur la structure de Kripke $M = (S, S_0, L, R)$.
- 1. Une transition $\alpha \in T$ est dite *invisible* si pour tout $s, s' \in S$ avec $s' = \alpha(s)$,

$$L(s) \cap AP' = L(s') \cap AP'.$$

2. La transition est *visible* sinon.

Réd. d'ordre part. pour $LTL - \circ$

- Plutôt que de proposer directement un algorithme pour calculer $ample(s)$, on va proposer des conditions qui le garantissent.
- Condition C_0: $ample(s) = \emptyset$ ssi $enabled(s) = \emptyset$.
  (autrement dit: si s a au moins un successeur, alors cela est aussi vrai dans le graphe réduit.)
- Condition C_1: si un chemin partant de s exécute à un certain moment une transition qui dépend d'une transition de $ample(s)$, alors il a exécuté avant une transition de $ample(s)$.

Conséquence

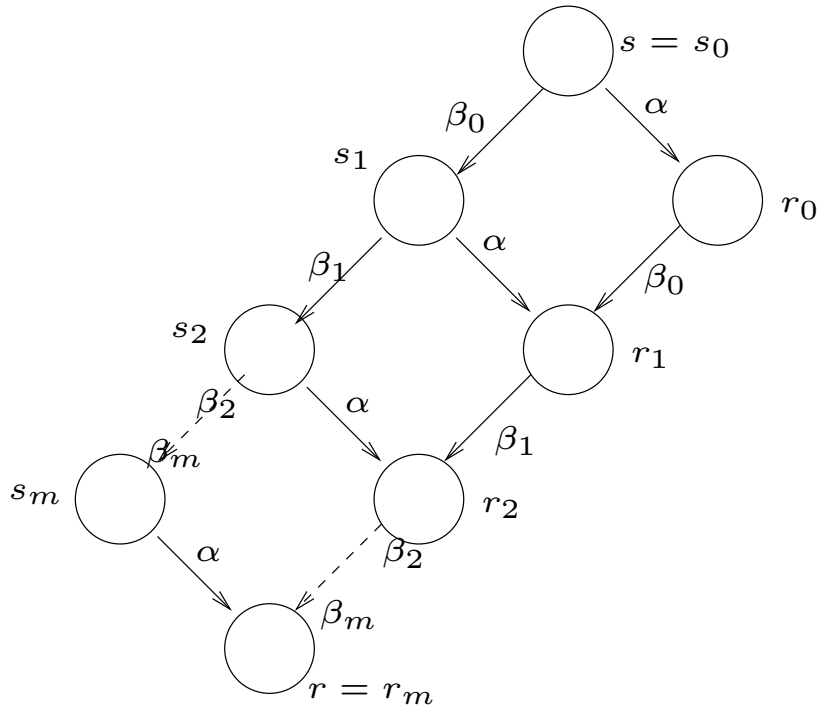
- Propriété: Si C_1 est vérifiée, les transitions de $enabled(s) - ample(s)$ sont toutes indépendantes de celle de $ample(s)$.

  Démonstration: supposons $\gamma \in enabled(s) - ample(s)$ dépendant d'une transition de $ample(s)$. Puisque $\gamma \in enabled(s)$, il y a un chemin dans la structure de Kripke qui commence par γ. Cela contredit la condition C_1.

Forme des chemins

- **Forme des Chemins:** La condition C_1 implique que tout chemin π partant de s doit avoir une des deux formes suivantes:
  1. π a un préfixe $\beta_0\beta_1 \ldots \beta_m\alpha$ avec $\alpha \in ample(s)$ tel que chaque β_i est indépendant de toutes les transitions de $ample(s)$.
  2. π est de la forme $\beta_0\beta_1 \ldots \beta_m \ldots$ tel que chaque β_i est indépendant de toutes les transitions de $ample(s)$.

Illustration: Cas 1.



- Dans le cas 1., la transition α commmute avec $\beta_0\beta_1 \ldots \beta_m$.

Condition C_2

- On voudrait que dans le cas 1. le chemin σ correspondant à $\beta_0\beta_1 \ldots \beta_m\alpha$ et le chemin ρ correspondant à $\alpha\beta_0\beta_1 \ldots \beta_m$ soient équivalents. (i.e. équivalents pour la propriété $LTL - \circ$ à vérifier = équivalents par bégaiements sur AP').

- Moyen: il suffirait que α soit invisible. En effet, on aurait $L(s_i) \cap AP' = L(r_i) \cap AP'$ pour tout i.
- Moyen: condition C_2.
- Un état s est dit *complètement étendu* si $ample(s) = enabled(s)$.
- Condition C_2: si s n'est pas complètement étendu, alors tout $\alpha \in ample(s)$ est invisible.

Conséquences de C_2

- Dans le cas $1.$ avec la condition C_2, on aura bien σ et ρ équivalents par bégaiements sur AP'.
- Dans le cas $2.$, le chemin $\beta_0\beta_1 \ldots \beta_m \ldots$ est bien équivalent par bégaiement à $\alpha\beta_0\beta_1 \ldots \beta_m$ sur AP'.
  Car par la condition C_2, α est invisible.
- Autrement dit, pour tout chemin partant de s, l'algorithme de parcours en profondeur construira bien dans tous les cas un chemin commençant par un $\alpha \in ample(s)$, équivalent par bégaiement.

Problème

- Les conditions C_0,C_1 et C_2 ne suffisent pas pour que le graphe réduit soit équivalent par bégaiement au graphe initial.

  Contre-exemple:

  - Suppose α_i invisibles, β visible.
  - Construit $ample(s_i) = \{\alpha_i\}$ successivement pour $i = 1, 2, 3$.
  - Cela vérifie C_0, C_1, C_2.
  - Mais on ignore la possibilité de prendre la transition β.

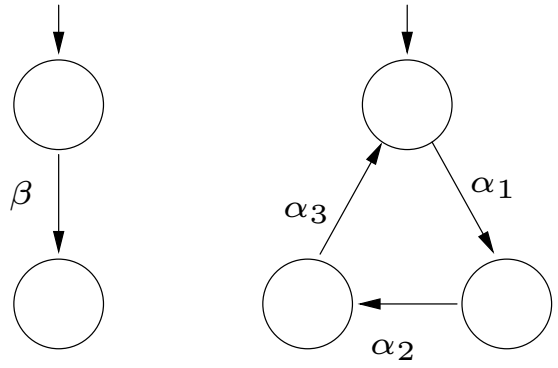


Figure 3: Deux processus concurrents

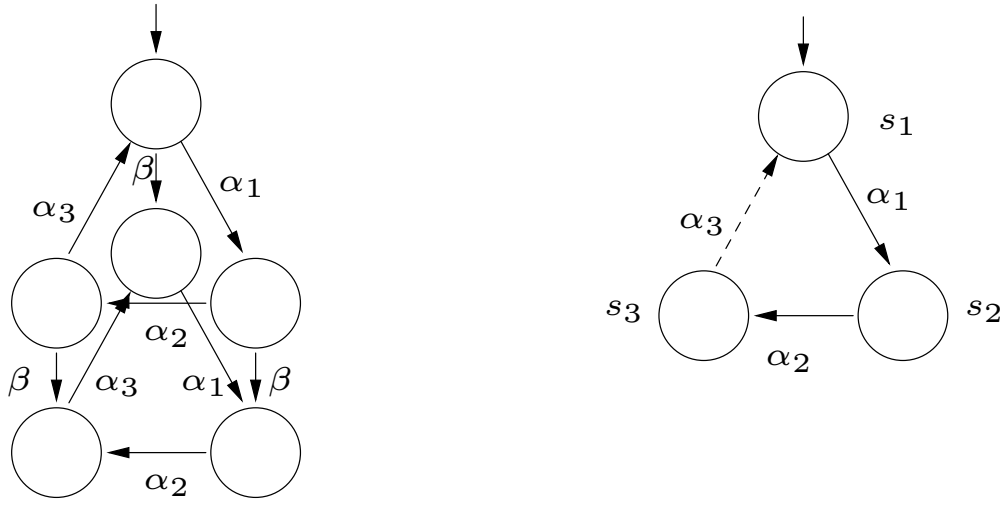


Figure 4: Graphe complet et graphe réduit

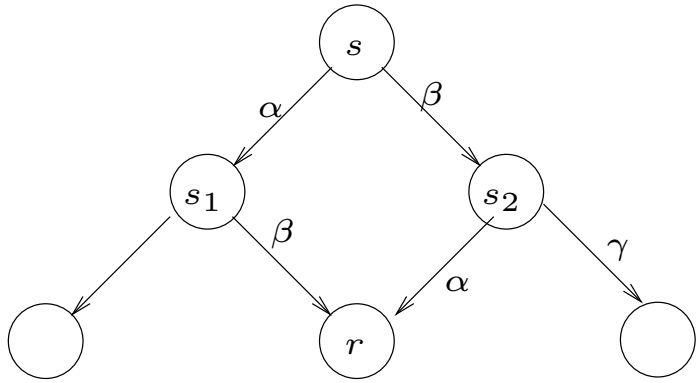
- **Le problème:** Le long du cycle s_1, s_2, s_3, on a pas exploré la transition β car on peut bien la retarder en s_1, s_2, s_3.
- Mais la propriété β que l'on cherche à vérifier peut dépendre de l'exécution de cette transition β, et donc il faut explorer à un moment ou à un autre un chemin qui contient cette transition.

Condition C_3

- Condition C_3: un cycle n'est pas autorisé s'il contient un état s dans lequel un certain α est dans $enabled(s)$ mais jamais inclus dans un $ample(t)$ pour aucun état t du cycle.
- Théorème: si les conditions C_0, C_1, C_2, C_3 sont vérifiées, alors l'algorithme de parcours en profondeur construit bien un graphe réduit M' équivalent au graphe initial pour la propriété $LTL - \circ$ que l'on cherche à vérifier.
- Autrement dit, le système initial vérifie la propriété $LTL - \circ$ que l'on cherche à vérifier ssi le système réduit M' la satisfait.

Démonstration

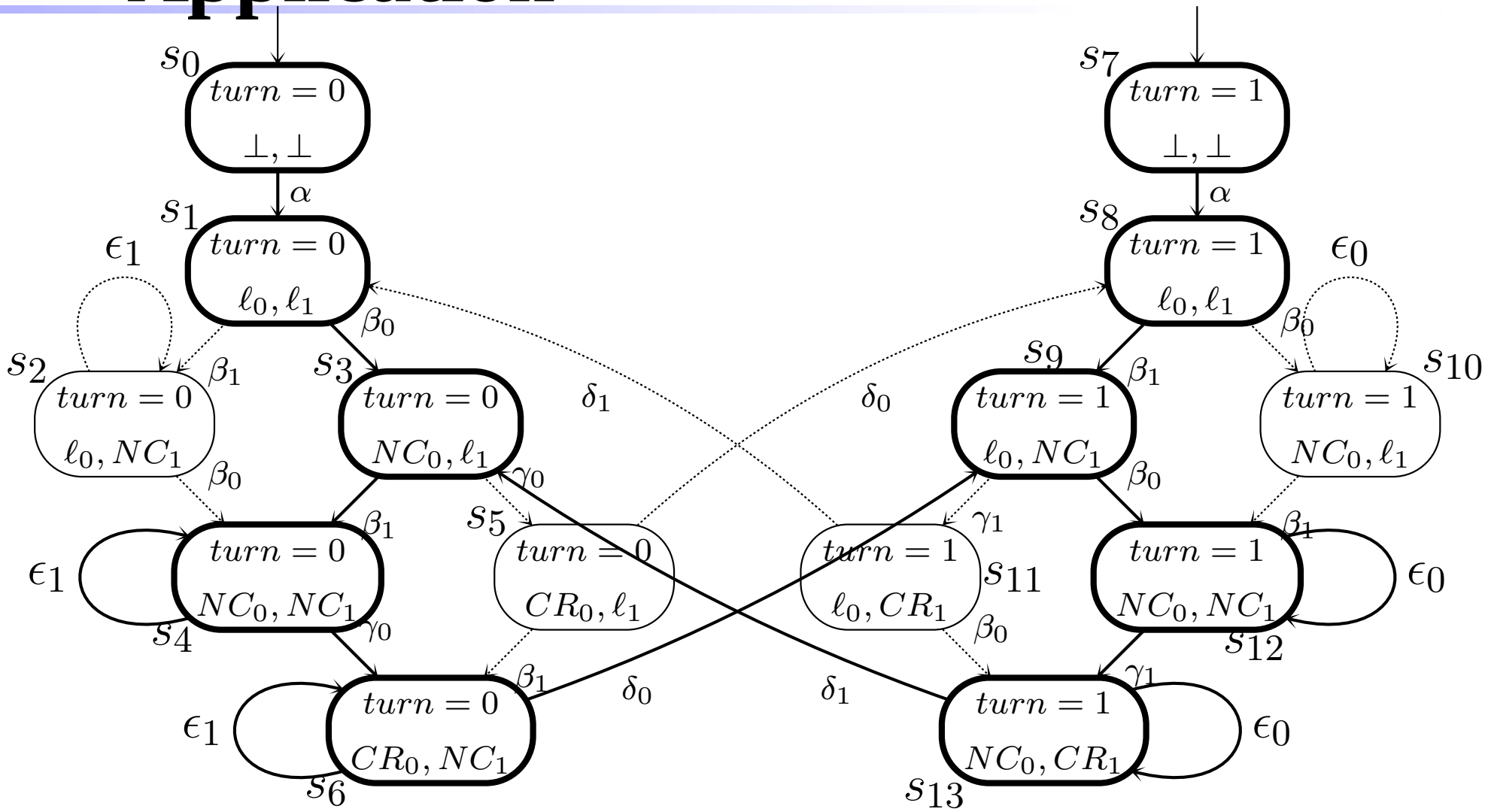
- Démonstration: Il faut montrer que le graphe réduit construit est bien équivalent au sens de la clôture par bégaiement de l'équivalence de traces au système initial.



- Il faut montrer que ne pas considérer la transition β 1) ne modifie pas la propriété à vérifier 2) n'oublie pas des successeurs éventuels de s_2 autre que r.

- Pour 1): par la condition C_2, β doit être invisible, et donc les chemins $\alpha\beta$ et $\beta\alpha$ sont bien équivalents par bégaiement.
- Pour 2): supposons que γ soit activée en s_2. γ ne peut être dépendante de α par la propriété C_1. Puisque γ est indépendante de α et activée en s_2 elle doit être active en r. Puisque β est invisible, les chemins $\alpha\beta\gamma$ et $\beta\gamma$ sont bien équivalents par bégaiement.

Application



Application: suite.

- Propriété $LTL - \circ$ à vérifier: $f = \forall\neg(CR_0 \wedge CR_1)$.
- $\alpha : pc = m \wedge pc'_0 = l_0 \wedge pc'_1 = l_1 \wedge pc' = \bot$
- $\beta_i : pc_i = l_i \wedge pc'_i = NC_i \wedge turn' = turn$
- $\gamma_i : pc_i = NC_i \wedge pc'_i = CR_i \wedge turn = i \wedge turn' = turn$
- $\delta_i : pc_i = CR_i \wedge pc'_i = l_i \wedge turn' = (i+1) mod 2$
- $\epsilon_i : pc_i = NC_i \wedge pc'_i = NC_i \wedge turn \neq i \wedge turn' = turn$

- $AP' = \{CR_0, CR_1\}$.
- Transitions visibles: $\gamma_0, \gamma_1, \delta_0, \delta_1$.
- Dépendances:
  - toutes transitions dépendent de α,
  - transitions interprocessus dépendantes,
  - $(\gamma_1, \delta_0), (\gamma_0, \delta_1), (\epsilon_1, \delta_0), (\epsilon_0, \delta_1), (\delta_0, \delta_1)$ sont dépendantes,
  - et toutes les symétriques des précédentes.
- Exemple de déroulement de l'algorithme:
  - Etat initial: s_0. $ample(s_0) = \{\alpha\}$
  - $ample(s_1) = \{\beta_0\}$ (on pourrait aussi choisir $\{\beta_1\}$ ou $\{\beta_0, \beta_1\}$) vérifie les conditions.
  - $ample(s_3) = \{\beta_1\}$ vérifie les conditions.
  - $ample(s_3) = \{\gamma_0, \epsilon_1\}$ vérifie les conditions.
  - $ample(s_6) = \{\epsilon_1, \delta_0\}$ vérifie les conditions.
  - $ample(s_{10}) = \{\beta_0\}$ vérifie les conditions.

- $ample(s_{11}) = \{\gamma_1, \epsilon_0\}$ vérifie les conditions.
- $ample(s_{13}) = \{\delta_1, \epsilon_0\}$ vérifie les conditions.
- $ample(s_7) = \{\alpha\}$ vérifie les conditions.
- $ample(s_8) = \{\beta_1\}$ vérifie les conditions.
- Le parcours s'arrête là.

Calcul automatique de $ample(s)$

Motivation

- Si l'on sait construire automatiquement des ensembles $ample(s)$ qui satisfont C_0, C_1, C_2, C_3, alors l'algorithme de recherche en profondeur retourne automatiquement pour tout système initial M un graphe réduit M'.
- Pour vérifier une propriété $LTL - \circ \phi$ sur M, il suffit de la vérifier sur M' car par ce qui précède: $M \models \phi$ ssi $M' \models \phi$.

Automatisation

- Propriété C_0: facile à garantir.
- Propriété C_1: difficile à vérifier car définie en termes globaux.

  Théorème: vérifier la condition C_1 pour un état s et un ensemble $T \subset enabled(s)$ de transitions est au moins aussi dur que le problème de l'atteignabilité pour le graphe complet dans le cas général.

  Démonstration: exercice.

  $\rightsquigarrow$ on évite de vérifier la condition C_1 pour sous-ensemble quelconque $T \subset enabled(s)$ mais on donne des conditions suffisantes pour des T bien choisis.
- Propriété C_2: facile à garantir.

Automatisation (suite)

- Propriété C_3: difficile à vérifier car définie en termes globaux, mais il est possible de trouver des conditions suffisantes.
- Exemple de condition suffisante pour C_3: au moins un état de chaque cycle est complètement étendu.
  Démonstration: supposons par l'absurde que C_3 ne soit pas vérifiée: il y a un cycle avec une transition α activée en un état s' mais incluse dans aucun $ample(s)$ du cycle. Puisque on sait qu'en tout état les transitions de les transitions de $enabled(s) - ample(s)$ sont toutes indépendantes de celle de $ample(s)$, α est indépendante de tous les transitions des $ample(s)$ du cycle. Par conséquent α est active en tous les états du cycle. Par hypothèse,le cycle a un état s'' complètement étendu, et donc pour lequel on a $\alpha \in ample(s'') = enabled(s'')$. Absurde.

- Application. Stratégie: choisir systématiquement dans $ample(s)$ des transitions qui mène à des états s' tel que $onstack(s')$ soit faux. Si on n'y arrive pas, on rend l'état s complètement étendu.

  Démonstration: un cycle contient au moins une "arrête arrière" dans un parcours en profondeur.

Heuristiques pour $ample(s)$

- Considérons un système constitué de processus P_i qui ont chacun une variable pc_i qui indique leur instruction courante.
- Notations:
  - $pre(\alpha)$ désigne un ensemble de transitions qui contient les transitions dont l'exécution peut activer α.
  - $dep(\alpha)$ désigne les transitions dépendantes de α.
  - T_i désigne les transitions du processus P_i.
  - $T_i(s)$ désigne $T_i \cap enabled(s)$.
  - $current_i(s)$ désigne l'ensemble des transitions de P_i actives dans un état s' tel que $pc_i(s') = pc_i(s)$.

- Les définitions précédentes s'étendent aux ensembles. Exemple: $dep(T) = \bigcup_{\alpha \in T} dep(\alpha)$.
- On suppose les transitions de l'ensemble $T_i(s)$ dépendantes deux à deux pour tout s.

Heuristiques

- Fonction $CheckC_1(s, P_i)$:
  (résultat vrai implique que $ample(s) = T_i(s)$ vérifie C_1, mais réciproque peut être fausse).

  pour tout $P_j \neq P_i$
  si $dep(T_i(s)) \cap T_j \neq \emptyset$
  ou $pre(current_i(s) - T_i(s)) \cap T_j \neq \emptyset$ alors
  retourner Faux

  retourner Vrai

- Fonction $CheckC_2(X)$

  pour tout $\alpha \in X$
  si $visible(\alpha)$ alors retourner Faux

  retourner Vrai

- Fonction $CheckC'_3(s, X)$

  pour tout $\alpha \in X$

  si $onstack(\alpha(s))$ retourner Faux

  retourner Vrai.

Fonction $ample(s)$

- Fonction $ample(s)$

  pour tout P_i tel que $T_i(s) \neq \emptyset$
  Si $CheckC_1(s, P_i)$ et $CheckC_2(T_i(s))$
  et $CheckC'_3(s, T_i(s))$ alors
  retourner $T_i(s)$

  retourner $enabled(s)$

- Théorème: La fonction $ample(s)$ retourne bien des ensembles qui vérifient C_0, C_1, C_2, C'_3.

Démonstration

- Démonstration. Les propriétés C_0, C_1, C_2, C_3' sont toujours trivialement vérifiées en un état s complètement étendu. Lorsque la fonction $ample(s)$ retourne un ensemble $ample(s)$ non-complètement étendu, elle retourne $ample(s) = T_i(s)$ et les fonctions $CheckC_2$ et $CheckC_3'$ qui ont nécessairement retourné vrai garantissent les propriétés C_2 et C_3'.
- Il reste uniquement à montrer que quand la fonction $CheckC_1$ retourne vrai alors $ample(s) = T_i(s)$ vérifie la propriété C_1 (la réciproque peut être fausse).

- Par l'absurde: sinon, cela signifie qu'il existe une suite de transitions $\beta_1\beta_2\ldots\beta_n$ indépendantes de celles de $T_i(s)$ qui active une transition α dépendante de $T_i(s)$.

  1. Si α appartient à un autre processus P_j, alors $dep(T_i(s))$ contient au moins α, et donc $dep(T_i(s)) \cap T_j(s) \neq \emptyset$: impossible car $CheckC_1$ a retourné vrai.
  2. Donc α appartient au processus P_i. Puisque toutes les transitions de $T_i(s)$ sont interdépendantes, les transitions $\beta_1, \ldots, \beta_n$ appartiennent à un autre processus que T_i. On doit donc avoir $\alpha \in current_i(s)$. On peut pas avoir $\alpha \in T_i(s)$ car sinon la suite de transitions ne violerait pas C_1, et donc α est désactivée en s. Par conséquent, une transition de $pre(\alpha)$ doit apparaître parmi $\beta_1, \ldots, \beta_n$ sur un autre processus P_j. On a donc $(pre(\alpha) \cap T_j) \neq \emptyset$ soit $pre(current_i(s) - T_i(s) \cap T_j) \neq \emptyset$ puisque $\alpha \in current_i(s) - T_i(s)$: impossible car $CheckC_1$ a retourné vrai.