Analyse des Programmes et Sémantique

Sémantique et Vérification de Processus

Irène Guessarian

ig@liafa.jussieu.fr

Ces notes ont été préparées en utilisant le package et le cours de Jiri Srba
http://www.cs.aau.dk/ srba/slides-sv.tar.gz

Analyse des Programmes et Sémantique

Sémantique et Vérification de Processus

Irène Guessarian

- Equivalence des Traces
- Bisimulations
- Caractérisation par jeux

Equivalence des Traces

On définit les équivalences sur un seul ST car : l'union de 2 ST est un ST ; de plus on veut comparer les dérivés d'un même processus. Soit $(Proc, Act, T)$ un ST.

Ensemble des Traces de $s \in Proc$

$$Traces(s) = \{w \in Act^* \mid \exists s' \in Proc.\ s \xrightarrow{w} s'\}$$

Soient $s \in Proc$ et $t \in Proc$.

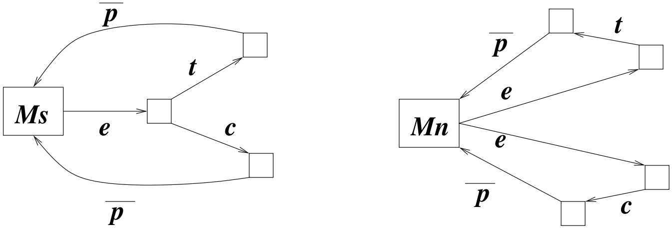
Equivalence des Traces

s et t sont équivalents pour les traces ($s \equiv_t t$) ssi

$$Traces(s) = Traces(t)$$

Equivalence des Traces

Intuition : Comportements Identiques Tout ce que fait un des processus doit pouvoir être fait par l'autre, et réciproquement. L'équivalence des traces (égalité des langages acceptés par les ST associés ne suffit pas).



Deux sytèmes de transition ayant les mêmes traces et des comportements différents.

Bisimulation forte

Soit $(Proc, Act, T)$ un ST.

Bisimulation forte

Une relation binaire $R \subseteq Proc \times Proc$ est une Bisimulation forte ssi pour tous $(s, t) \in R$ et pour tout $a \in Act$:

- si $s \xrightarrow{a} s'$ alors $t \xrightarrow{a} t'$ pour un t' tel que $(s', t') \in R$
- si $t \xrightarrow{a} t'$ alors $s \xrightarrow{a} s'$ pour un s' tel que $(s', t') \in R$.

Bisimulation forte

Deux processus $p_1, p_2 \in Proc$ sont fortement bisimilaires ($p_1 \sim p_2$) ssi il existe une bisimulation forte R telle que $(p_1, p_2) \in R$.

$$\sim \stackrel{\text{def}}{=} \cup\{R \mid R \text{ est une Bisimulation forte}\}$$

Propriétés de la Bisimulation forte

théorème

$\sim$ est une relation d'équivalence (réflexive, symétrique et transitive)

théorème

$\sim$ est la plus grande bisimulation forte

théorème

$s \sim t$ ssi pour tout $a \in Act$:

- si $s \xrightarrow{a} s'$ alors $t \xrightarrow{a} t'$ pour un t' tel que $s' \sim t'$
- si $t \xrightarrow{a} t'$ alors $s \xrightarrow{a} s'$ pour un s' tel que $s' \sim t'$.

Définition circulaire : s'agit-il d'induction ?

(Co)–Induction

Problème

Soit $f : E \longrightarrow E$, pour quels x a-t-on $x = f(x)$? x est un point fixe.

Ensemble ordonné – Treillis

- Un ensemble ordonné est un ensemble E muni d'une relation d'ordre $\sqsubseteq$
- Un ensemble ordonné est un treillis complet ssi toute partie X de E admet un sup et un inf (notés $\sqcap X$ et $\sqcup X$)

On pose $\top \stackrel{\text{def}}{=} \sqcup E$ and $\bot \stackrel{\text{def}}{=} \sqcap E$ (appelés top et bottom).

Plus petit (grand) point fixe

monotonie

$f : E \longrightarrow E$ est momotone ssi pour tout $e, e' \in E$:

$$e \sqsubseteq e' \quad \Rightarrow \quad f(e) \sqsubseteq f(e')$$

Théorème de Tarski

Soient $(E, \sqsubseteq)$ un treillis complet et $f : E \rightarrow E$ une fonction monotone. f a un unique plus grand point fixe $\nu x.f$ et un unique plus petit point fixe $\mu x.f$ définis par :

$$\nu x.f \stackrel{\text{def}}{=} \sqcup\{x \in E \mid x \sqsubseteq f(x)\}$$

$$\mu x.f \stackrel{\text{def}}{=} \sqcap\{x \in E \mid f(x) \sqsubseteq x\}$$

Calcul du plus petit (grand) point fixe

monotonie

$f : E \longrightarrow E$ est sup–continue ssi pour toute suite croissante $e_1 \sqsubseteq e_2 \sqsubseteq \cdots \in E$: $f(\sqcup\{e_n | n \in \mathbb{N}\} = \sqcup\{f(e_n) | n \in \mathbb{N}\}$ (mutatis mutandis pour inf–continue).

Théorème de Tarski constructif

Si $f : E \rightarrow E$ est sup–continue (resp. inf–continue) :

$$\begin{aligned} \mu x.f &\stackrel{\text{def}}{=} \sqcup\{f^n(\bot) | n \in \mathbb{N}\} \\ (\text{resp. } \nu x.f &\stackrel{\text{def}}{=} \sqcap\{f^n(\top) | n \in \mathbb{N}\}\ ) \end{aligned}$$

Si E est fini toutes les fonctions monotones sont sup–continues et inf–continues, et on peut construire les points fixes en un nombre

Dualité

Dualité :induction *versus* co–induction

Induction	co–induction
contructeurs	destructeurs
plus petit P.F.	plus grand P.F.
$\sqcap\{x \in E \mid f(x) \sqsubseteq x\}$	$\sqcap\{x \in E \mid f(x) \sqsubseteq x\}$
$\sqcup_{n\in\mathbb{N}} f^n(\bot)$	$\sqcap_{n\in\mathbb{N}} f^n(\top)$
technique de preuve :	technique de preuve :
$f(P) \subseteq P \Longrightarrow \mu x.f \subseteq P$	$f(P) \supseteq P \Longrightarrow \nu x.f \supseteq P$

Bisimulation forte = co–induction

Fonction $\mathcal{F}$: $2^{(Proc \times Proc)} \longrightarrow 2^{(Proc \times Proc)}$

Soient $S \subseteq Proc \times Proc$ et $\mathcal{F}(S)$ définie par :

$(s, t) \in \mathcal{F}(S)$ if and only if for each $a \in Act$:

- if $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ for some t' such that $(s', t') \in S$
- if $t \xrightarrow{a} t'$ then $s \xrightarrow{a} s'$ for some s' such that $(s', t') \in S$.

la bisimulation forte $\sim$ est le plus grand P.F. de $\mathcal{F}$

- $(2^{(Proc \times Proc)}, \subseteq)$ treillis complet et $\mathcal{F}$ monotone
- S bisimulation forte ssi $S \subseteq \mathcal{F}(S)$

$$\sim = \bigcup \{S \in 2^{(Proc \times Proc)} \mid S \subseteq \mathcal{F}(S)\}$$

Bisimulation forte = co–induction

Si le ST décrivant s est à branchement fini
$(\forall q \in Proc, \{q' \mid \exists a \in Act,\ q \xrightarrow{a} q'\}$ est borné) alors
$\mathcal{F} \colon 2^{(Proc \times Proc)} \longrightarrow 2^{(Proc \times Proc)}$ est inf-continue et

$$\sim = \bigcap_{n \in \mathbb{N}} \mathcal{F}^n(Proc \times Proc)$$

Soit la suite d'équivalences définie par : $\sim_0 = Proc \times Proc$ et
$s \sim_{n+1} t$ ssi :

- si $s \xrightarrow{a} s'$ alors $t \xrightarrow{a} t'$ pour un t' tel que $s' \sim_n t'$
- si $t \xrightarrow{a} t'$ alors $s \xrightarrow{a} s'$ pour un s' tel que $s' \sim_n t'$.

et soit $\sim_\omega \stackrel{\text{def}}{=} \bigcap_{n \in \mathbb{N}} \sim_n$.
Pour un ST à branchement fini, $\sim_\omega = \sim$. (faux si branchements non bornés).

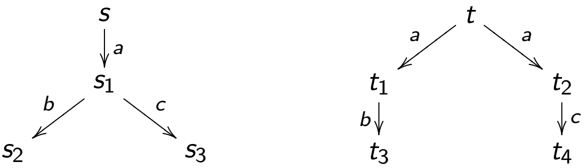
Bisimulation forte *versus* HML

Théorème pour ST à branchement fini

$s \sim t$ ssi s et t satisfont exactement les mêmes formules de la logique HML.

- Comment prouver que $s \not\sim t$?
- Comment prouver que $p \not\models F$?

Comment prouver que $s \not\sim t$?



pour prouver que $s \not\sim t$:

- Enumérer toutes les relations contenant (s, t) et montrer qu'aucune n'est une bisimulation forte. ($2^{|Proc|^2}$ relations.)
- Faire des observations disqualifient la plupart des candidats à bisimulation en une ou deux étapes.
- Technique de jeux.

Jeu pour la Bisimulation forte

Soient $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ un ST et $s, t \in Proc$.

On définit un jeu a deux joueurs 'attaquant' et 'défenseur' qui commence en s et t.

- Le jeu consiste de rounds les configurations du jeu sont des couples d'états dans $Proc \times Proc$.
- Dans chaque round il y a une seule configuration actuelle. Au début la configuration actuelle est (s, t).

Intuition

Le défenseur veut montrer que $s \sim t$, et l'attaquant veut montrer que $s \not\sim t$.

Règles du jeu

Règles du jeu

A chaque round les joueurs changent la configuration courante comme suit :

1. l'attaquant choisit un des processus de la configuration courante et fait une $\xrightarrow{a}$-transition pour un $a \in Act$, et
2. le défenseur doit répondre par une $\xrightarrow{a}$-transition dans l'autre processus avec la même action a.

Le nouveau couple de processus obtenu devient la configuration courante. Le jeu continue par un autre round.

Résultat du jeu

- Si un joueur ne peut plus bouger,l'autre gagne.

Caractérisation de la Bisimulation forte par jeu

Théorème

- $s \sim t$ ssi le défenseur a une stratégie gagnante universelle au départ de la configuration (s, t).
- $s \not\sim t$ l'attaquant a une stratégie gagnante universelle au départ de la configuration (s, t).

Remarque

Ce jeu permet de prouver de manière élégante à la fois bisimilarité ou nonbisimilarité de deux processus.

Limites de la logique HML

profondeur modale (nesting degree) des formules HML:

- $\mathit{prof}(\mathit{tt}) = \mathit{prof}(\mathit{ff}) = 0$
- $\mathit{prof}(F \wedge G) = \mathit{prof}(F \vee G) = \max\{\mathit{prof}(F), \mathit{prof}(G)\}$
- $\mathit{prof}([a]F) = \mathit{prof}(\langle a \rangle F) = \mathit{prof}(F) + 1$

Idée: la formule F peut "voir" jusqu'à la profondeur $\mathit{prof}(F)$.

Théorème (soit F une formule HML et $k = \mathit{prof}(F)$)

Si le défenseur a une stratégie dans le jeu de bisimulation forte pour s et t jusqu'à k rounds alors $s \models F$ ssi $t \models F$.

Conclusion

Aucune formule HML ne peut détecter (l'absence de) blocage dans un ST arbitraire.

Quelques propriété de la Bisimulation forte

la Bisimulation forte est une Congruence pour toutes les opérations de CCS

Soient P et Q des processus CCS tels que $P \sim Q$.

- $a.P \sim a.Q$ pour toute action $a \in Act$
- $P + R \sim Q + R$ et $R + P \sim R + Q$ pour tout processus CCS R
- $P \parallel R \sim Q \parallel R$ et $R \parallel P \sim R \parallel Q$ pour tout processus CCS R
- $P[f] \sim Q[f]$ pour tout renommage f
- $P \setminus L \sim Q \setminus L$ pour tout ensemble de labels L.

Soient P, Q et R des processus CCS

- $P + Q \sim Q + P$
- $P \parallel Q \sim Q \parallel P$
- $P \parallel Nil \sim P$
- $(P + Q) + R \sim P + (Q + R)$

Les Actions internes comptent

Question

$a.\tau.Nil \sim a.Nil$ vrai ? NON !

La bisimulation forte prend en compte les τ-actions qui pourtant ne sont pas observables.

Exemple: $S \not\sim Spec$

$$
\begin{aligned}
M &\stackrel{\text{def}}{=} e.\bar{c}.M \\
I &\stackrel{\text{def}}{=} ex.\bar{e}.c.I \\
S &\stackrel{\text{def}}{=} (M \parallel I) \setminus \{e, c\}
\end{aligned}
$$

$$Spec \stackrel{\text{def}}{=} ex.Spec$$

Transitions modulo τ

Soit $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ un ST tel que $\tau \in Act$.

Définition des Transitions modulo τ

$$\stackrel{a}{\Longrightarrow} = \begin{cases} (\xrightarrow{\tau})^* \circ \xrightarrow{a} \circ (\xrightarrow{\tau})^* & \text{if } a \neq \tau \\ (\xrightarrow{\tau})^* & \text{if } a = \tau \end{cases}$$

Intuition :

- Si $a \neq \tau$ alors $s \stackrel{a}{\Longrightarrow} t$ signifie que
  de s on peut aller à t en faisant zéro ou plusieurs τ actions, puis l'action a, puis τ zéro ou plusieurs actions.
- Si $a = \tau$ alors $s \stackrel{\tau}{\Longrightarrow} t$ signifie que
  de s on peut aller à t en faisant zéro ou plusieurs τ actions.

Bisimulation faible

Soit $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ un ST tel que $\tau \in Act$.

Bisimulation faible

La relation binaire $R \subseteq Proc \times Proc$ est une bisimulation faible ssi pour tous $(s, t) \in R$ et pour toute $a \in Act$ (y compris τ):

- si $s \xrightarrow{a} s'$ alors $t \xRightarrow{a} t'$ pour un t' tel que $(s', t') \in R$
- si $t \xrightarrow{a} t'$ alors $s \xRightarrow{a} s'$ pour un s' tel que $(s', t') \in R$.

Bisimulation faible

Bisimulation faible

Deux processus $p_1, p_2 \in Proc$ sont faiblement bisimilaires ($p_1 \approx p_2$) ssi il existe une bisimulation faible R telle que $(p_1, p_2) \in R$.

$$\approx \;=\; \cup\{R \mid R \text{ est une bisimulation faible}\}$$

Bisimulation forte vs. faible

La bisimulation faible est plus réaliste. Pourquoi la Bisimulation forte ?

- $\sim$ est plus facile à étudier que $\approx$, et de plus $\sim \subseteq \approx$;
- la théorie de $\sim$ est très proche de celle de $\approx$;
- les différences entre $\sim$ et $\approx$ correspondent à des détails très pointus de la théorie de $\approx$.

Jeu pour la Bisimulation faible

Definition

Comme pour la Bisimulation forte ; seuls changements :

- le défenseur peut faire des transitions $\stackrel{a}{\Longrightarrow}$.

L'attaquant ne peut faire que des transitions $\stackrel{a}{\longrightarrow}$.

Théorème

- s et t sont faiblement bisimilaires ssi le défenseur a stratégie gagnante universelle en partant de la configuration (s, t).
- s et t ne sont pas faiblement bisimilaires ssi l'attaquant a stratégie gagnante universelle en partant de la configuration (s, t).

Propriétés de la Bisimulation faible

- $\approx$ est une relation d'équivalence
- $\approx$ est la plus grande bisimulation faible
- $\approx$ satisfait pour tous P et Q :
  - $a.\tau.P \approx a.P$
  - $P + \tau.P \approx \tau.P$
  - $a.(P + \tau.Q) \approx a.(P + \tau.Q) + a.Q$
  - $P + Q \approx Q + P \qquad P \| Q \approx Q \| P \qquad P + Nil \approx P \qquad \ldots$
- la bisimulation forte est contenue dans la bisimulation faible $(\sim \subseteq \approx)$
- $\approx$ ne tient pas compte des boucles de τ-actions

$$P \stackrel{\text{def}}{=} \tau.P + a.NIL \qquad Q \stackrel{\text{def}}{=} a.NIL$$
$$P \quad \approx \quad Q$$

La Bisimulation faible est-elle une Congruence pour CCS?

Théorème

Soient P et Q des processus CCS tels que $P \approx Q$. Alors

- $\alpha.P \approx \alpha.Q$ pour toute action $\alpha \in Act$
- $P \parallel R \approx Q \parallel R$ et $R \parallel P \approx R \parallel Q$ pour tout processus CCS R
- $P[f] \approx Q[f]$ pour tout renommage f
- $P \setminus L \approx Q \setminus L$ pour tout ensemble d'étiquettes L.

Problème : le choix non déterministe $+$

$\tau.a.Nil \approx a.Nil$ mais $\tau.a.Nil + b.Nil \not\approx a.Nil + b.Nil$

Conclusion

La Bisimulation faible n'est pas une congruence pour CCS.

Exemple: un Protocole de Communication

$$\begin{aligned} \mathsf{Send} &\stackrel{\text{def}}{=} \mathsf{acc.Sending} \\ \mathsf{Sending} &\stackrel{\text{def}}{=} \overline{\mathsf{send}}.\mathsf{Wait} \\ \mathsf{Wait} &\stackrel{\text{def}}{=} \mathsf{ack.Send} + \mathsf{error.\ Sending} \end{aligned}$$

$$\begin{aligned} \mathsf{Rec} &\stackrel{\text{def}}{=} \mathsf{trans.Del} \\ \mathsf{Del} &\stackrel{\text{def}}{=} \overline{\mathsf{del}}.\mathsf{Ack} \\ \mathsf{Ack} &\stackrel{\text{def}}{=} \overline{\mathsf{ack}}.\mathsf{Rec} \end{aligned}$$

$$\begin{aligned} \mathsf{Med} &\stackrel{\text{def}}{=} \mathsf{send.Med}' \\ \mathsf{Med}' &\stackrel{\text{def}}{=} \tau.\mathsf{Err} + \overline{\mathsf{trans}}.\mathsf{Med} \\ \mathsf{Err} &\stackrel{\text{def}}{=} \overline{\mathsf{error}}.\mathsf{Med} \end{aligned}$$

Vérification

$$\mathsf{Impl} \stackrel{\text{def}}{=} (\mathsf{Send} \parallel \mathsf{Med} \parallel \mathsf{Rec}) \setminus \{\mathsf{send}, \mathsf{trans}, \mathsf{ack}, \mathsf{error}\}$$

$$\mathsf{Spec} \stackrel{\text{def}}{=} \mathsf{acc}.\overline{\mathsf{del}}.\mathsf{Spec}$$

Question

$$\mathsf{Impl} \stackrel{?}{\approx} \mathsf{Spec}$$

1. Dessiner les ST de Impl et Spec et montrer l'équivalence.
2. Utiliser Concurrency WorkBench (CWB).

Les Expressions CCS en CWB

Définitions CCS

$\mathrm{Med} \stackrel{\mathrm{def}}{=} \mathrm{send}.\mathrm{Med}'$

$\mathrm{Med}' \stackrel{\mathrm{def}}{=} \tau.\mathrm{Err} + \overline{\mathrm{trans}}.\ \mathrm{Med}$

$\mathrm{Err} \stackrel{\mathrm{def}}{=} \overline{\mathrm{error}}.\mathrm{Med}$

$\vdots$

$\mathrm{Impl} \stackrel{\mathrm{def}}{=}$
$(\mathrm{Send} \parallel \mathrm{Med} \parallel \mathrm{Rec}) \setminus \{\mathrm{send}, \mathrm{trans}, \mathrm{ack}, \mathrm{error}\}$

$\mathrm{Spec} \stackrel{\mathrm{def}}{=} \mathrm{acc}.\overline{\mathrm{del}}.\mathrm{Spec}$

Programme CWB (protocol.cwb)

agent Med = send.Med';
agent Med' = (tau.Err + 'trans.Med);
agent Err = 'error.Med;
$\vdots$
set L = {send, trans, ack, error};
agent Impl = (Send ∥ Med ∥ Rec) ∖ L;

agent Spec = acc.'del.Spec;

Session CWB

```
fire1$ /pack/FS/CWB/cwb

> help;

> input "protocol.cwb";

> vs(5,Impl);

> sim(Spec);

> eq(Spec,Impl);              ** weak bisimilarity **

> strongeq(Spec,Impl);        ** strong bisimilarity **
```

HML avec récursion – Syntaxe

Syntaxe des Formules

Les Formules sont définies par la syntaxe abstraite suivante

$$F ::= X \mid tt \mid ff \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \langle a \rangle F \mid [a]F$$

avec $a \in Act$ et X variable définie par
$X = \mu X.F_X$, ou $X = \nu X.F_X$
(F_X est une formule qui peut contenir la variable X).

Sémantique ?

Pour chaque formule F définir une fonction $O_F : 2^{Proc} \longrightarrow 2^{Proc}$ telle que

- si S est l'ensemble des processus satisfaisant X alors $O_F(S)$ est l'ensemble des processus satisfaisant F

Sémantique de HML

Définition de $O_F : 2^{Proc} \rightarrow 2^{Proc}$. Soit $S \subseteq Proc$.

$$
\begin{array}{rcl}
O_X(S) & = & S \\
O_{tt}(S) & = & Proc \\
O_{ff}(S) & = & \emptyset \\
O_{F_1 \wedge F_2}(S) & = & O_{F_1}(S) \cap O_{F_2}(S) \\
O_{F_1 \vee F_2}(S) & = & O_{F_1}(S) \cup O_{F_2}(S) \\
O_{\langle a \rangle F}(S) & = & \langle \cdot a \cdot \rangle O_F(S) \\
O_{[a] F}(S) & = & [\cdot a \cdot] O_F(S)
\end{array}
$$

Sémantique de HML

O_F est monotone

Pour toute formule F :

$$S_1 \subseteq S_2 \;\Rightarrow\; O_F(S_1) \subseteq O_F(S_2)$$

Preuve : induction structurelle sur la structure de F.
$(2^{Proc}, \subseteq)$ est un treillis complet et O_F est (sup et inf)continue, donc O_F a un unique plus grand point fixe et un unique plus petit point fixe.

Sémantique de la variable X

- Si $X = \nu X.F_X$ alors $[\![X]\!] = \bigcap_{n\in\mathbb{N}} O^n_{F_X}(Proc)$
- Si $X = \mu X.F_X$ alors $[\![X]\!] = \bigcup_{n\in\mathbb{N}} O^n_{F_X}(\emptyset)$

Caractérisation par jeux

Intuition : l'attaquant veut prouver $s \not\models F$, le défenseur veut prouver $s \models F$.

Configurations du jeu: (s, F) configuration initiale

- (s, tt) et (s, ff) n'ont pas de successeurs
- (s, X) a un successeur (s, F_X)
- $(s, F_1 \wedge F_2)$ a deux successeurs (s, F_1) et (s, F_2)
  (l'attaquant en choisit un)
- $(s, F_1 \vee F_2)$ a deux successeurs (s, F_1) and (s, F_2)
  (défenseur en choisit un)

Qui gagne ?

- $(s, [a]F)$ a les successeurs (s', F) pour tout s' t.q. $s \xrightarrow{a} s'$
  (l'attaquant en choisit un)
- $(s, \langle a \rangle F)$ a les successeurs (s', F) pour tout s' t.q. $s \xrightarrow{a} s'$
  (le défenseur en choisit un)

Une partie est une suite maximale de configurations formées en suivant les règles données.

Qui gagne ?

Partie Finie

- L'attaquant gagne une partie finie si le défenseur est bloqué ou si on atteint la configuration (s, ff).
- Le défenseur gagne une partie finie si l'attaquant est bloqué ou si on atteint la configuration (s, tt).

Partie Infinie

- L'attaquant gagne une partie infinie si X est défini par $X \mu X.F_X$.
- Le défenseur gagne une partie infinie si X est défini par $X \nu X.F_X$.

Caractérisation par jeux

Théorème

- $s \models F$ ssi le défenseur a une stratégie gagnante universelle à partir de (s, F)
- $s \not\models F$ ssi l'attaquant a une stratégie gagnante universelle à partir de (s, F)

Rappel : Limites de la logique de Hennessy-Milner

Certaines propriétés temporelles utiles ne sont pas exprimables dans la logique de HML :

$s \models Inv(F)$ ssi tous les états accessibles depuis s satisfont F

$s \models Pos(F)$ ssi il existe un état accessible depuis s qui satisfait F

- Les propriétés $Inv(F)$ et $Pos(F)$ ne sont pas exprimables par des formules HML
- $Inv(F)$ et $Pos(F)$ sont exprimables par des formules HML avec récursion

Sélection de Propriétés Temporelles

- $Inv(F)$: $X \stackrel{\max}{=} F \wedge [Act]X$
- $Pos(F)$: $X \stackrel{\min}{=} F \vee \langle Act \rangle X$

- $Safe(F)$: $X \stackrel{\max}{=} F \wedge ([Act]ff \vee \langle Act \rangle X)$
- $Even(F)$: $X \stackrel{\min}{=} F \vee (\langle Act \rangle tt \wedge [Act]X)$

- $F\ \mathcal{U}^w\ G$: $X \stackrel{\max}{=} G \vee (F \wedge [Act]X)$
- $F\ \mathcal{U}^s\ G$: $X \stackrel{\min}{=} G \vee (F \wedge \langle Act \rangle tt \wedge [Act]X)$

Avec until on peut exprimer $Inv(F)$ et $Even(F)$:

$$Inv(F) \equiv F\ \mathcal{U}^w\ ff \qquad\qquad Even(F) \equiv tt\ \mathcal{U}^s\ F$$

Exemples

Récursion imbriquée et formules mutuellement récursives

$$X \overset{\min}{=} Y \vee \langle Act \rangle X \qquad Y \overset{\max}{=} \langle a \rangle tt \wedge \langle Act \rangle Y$$

Solution : calculer d'abord $[\![Y]\!]$ et après $[\![X]\!]$.

Formules mutuellement récursives

$$X \overset{\max}{=} [a]Y \qquad Y \overset{\max}{=} \langle a \rangle X$$

Solution : Soit le treillis complet $(2^{Proc} \times 2^{Proc}, \sqsubseteq)$ où $(S_1, S_2) \sqsubseteq (S_1', S_2')$ ssi $S_1 \subseteq S_1'$ et $S_2 \subseteq S_2'$.

Théorème (Propriété caractéristique des processus à nombre d'états fini)

Soit s un processus ayant un nombre fini d'états atteignables. Il existe une propriété X_s t.q. pour tout processus t : $s \sim t$ ssi $t \in [\![X_s]\!]$.

- systèmes de transitions temporisés
- automates temporisés
- bisimulation (temporisée ou non)
- équivalence de langages (temporisée ou non)

Pourquoi le Temps ?

- Timeout dans le protocole du Bit Alterné :
  - En CCS on modélise les timeouts avec le nondéterminisme.
  - Suffit à prouver la sureté du protocole.
  - Trop abstrait pour modéliser par exemple le temps nécessaire pour faire parvenir un message.

- Beaucoup de systèmes dépendent du temps:
  - controleurs temps réel (lignes de production, ordinateurs des voitures, régulation de la circulation ferroviaire).
  - systèmes embarqués (téléphones portables, controleur à distance, télécommandes, montres numériques).
  - ...

Systèmes de transitions temporisés

Systèmes de transitions étiqueté et temporisé (STET)

un STET est un triplet $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ où

- *Proc* est l'ensemble des états (ou processes),
- $Act = N \cup \mathbb{R}^{\geq 0}$ est l'ensemble des actions (formées d'un label et d'une contrainte de temps), et
- pour toute $a \in Act$, $\xrightarrow{a} \subseteq \; Proc \times Proc$ est une relation binaire sur les états la relation de transition.

On écrit

- $s \xrightarrow{a} s'$ si $a \in N$ et $(s, s') \in \xrightarrow{a}$, et
- $s \xrightarrow{d} s'$ si $d \in \mathbb{R}^{\geq 0}$ et $(s, s') \in \xrightarrow{d}$.

Description des systèmes temporisés

Syntaxe inconnue	$\longrightarrow$	Sémantique connue
CCS	$\longrightarrow$	Systèmes de Transitions
???	$\longrightarrow$	Systèmes de Transitions Temporisés

Automates Temporisés [Alur, Dill'90]

Automates finis + horloge .

Définition d'un AT: Contraintes de temps

Soit $C = \{x, y, \ldots\}$ un ensemble fini d'horloges.

Soit $\mathcal{B}(C)$ un ensemble de contraintes de temps sur C

$\mathcal{B}(C)$ est défined par la syntaxe abstraite

$$g, g_1, g_2 \ ::= x \sim n \ \mid \ x - y \sim n \ \mid \ g_1 \wedge g_2$$

où $x, y \in C$ sont des horloges, $n \in \mathbb{N}$ et $\sim \in \{\leq, <, =, >, \geq\}$.

Exemple: $x \leq 3 \ \wedge \ y > 0 \ \wedge \ y - x = 2$

Valuation d'horloge

valuation d'horloge

Une valuation d'horloge v est une fonction $v : C \to \mathbb{R}^{\geq 0}$.

Soit v une valuation d'horloge. Alors

- $v + d$ est une valuation d'horloge pour tout $d \in \mathbb{R}^{\geq 0}$ et elle est définie par

$$(v + d)(x) = v(x) + d \text{ pour tout } x \in C$$

- $v[r]$ est une valuation d'horloge pour tout $r \subseteq C$ et elle est définie par

$$v[r](x) \begin{cases} 0 & \text{si } x \in r \\ v(x) & \text{sinon.} \end{cases}$$

Evaluation des contraintes d'horloge

Evaluation des contraintes d'horloge ($v \models g$)

$$
\begin{array}{ll}
v \models x < n & \text{ssi } v(x) < n \\
v \models x \leq n & \text{ssi } v(x) \leq n \\
v \models x = n & \text{ssi } v(x) = n \\
\vdots & \\
v \models x - y < n & \text{ssi } v(x) - v(y) < n \\
v \models x - y \leq n & \text{ssi } v(x) - v(y) \leq n \\
\vdots & \\
v \models g_1 \wedge g_2 & \text{ssi } v \models g_1 \text{ et } v \models g_2
\end{array}
$$

Syntaxe des Automates temporisés

Définition

Un Automate Temporisé sur un ensemble d'horloges C et un ensemble de labels N est un quadruplet (L, ℓ_0, E, I) où

- L est un ensemble fini de locations
- $\ell_0 \in L$ est la location initiale
- $E \subseteq L \times \mathcal{B}(C) \times N \times 2^C \times L$ est l'ensemble des arcs
- $I : L \to \mathcal{B}(C)$ assigne des invariants aux locations.

On écrit $\ell \xrightarrow{g,a,r} \ell'$ quand $(\ell, g, a, r, \ell') \in E$.

Sémantique des Automates Temporisés

Soit $A = (L, \ell_0, E, I)$ un AT.

Système de transitions temporisé engendré par A

$T(A) = (Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ où

- $Proc = L \times (C \to \mathbb{R}^{\geq 0})$, i.e. les états sont de la forme (ℓ, v) où ℓ est une location et v est une valuation
- $Act = N \cup \mathbb{R}^{\geq 0}$
- $\longrightarrow$ est défini par :

$(\ell, v) \xrightarrow{a} (\ell', v')$ s'il y a $(\ell \xrightarrow{g,a,r} \ell') \in E$ t.q. $v \models g$ et $v' = v[r]$
$(\ell, v) \xrightarrow{d} (\ell, v + d)$ pour tout $d \in \mathbb{R}^{\geq 0}$ t.q. $v \models I(\ell)$ et $v + d \models I(\ell)$

Bisimulation temporisée

Soient A_1 et A_2 des AT

Bisimulation temporisée

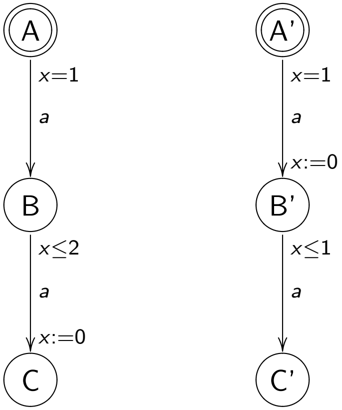
On dit que A_1 et A_2 sont temporellement bisimilaires ssi les systèmes de transition $T(A_1)$ et $T(A_2)$ engendrés par A_1 et A_2 sont fortement bisimilaires.

Remarque:

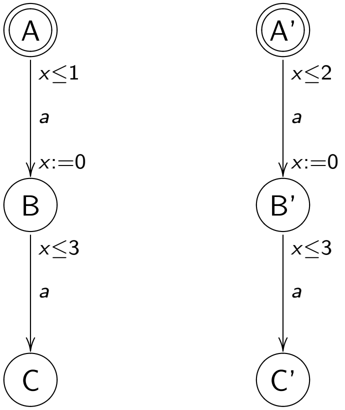
- $\xrightarrow{a}$ pour $a \in N$ et
- $\xrightarrow{d}$ pour $d \in \mathbb{R}^{\geq 0}$

sont considérées comme des transitions (visibles).

Exemple d'automates temporisés bisimilaires



Exemple d'automates temporisés non bisimilaires



Bisimulation non-temporisée

Soient A_1 et A_2 des AT. Soit ϵ une nouvelle action.

Bisimulation non-temporisée

On dit que A_1 et A_2 sont Bisimaires non-temporellement ssi les systèmes de transition $T(A_1)$ et $T(A_2)$ engendrés par A_1 et A_2 où toute transition de la forme $\xrightarrow{d}$ pour $d \in \mathbb{R}^{\geq 0}$ est remplacée par $\xrightarrow{\epsilon}$ sont fortement bisimilaires.

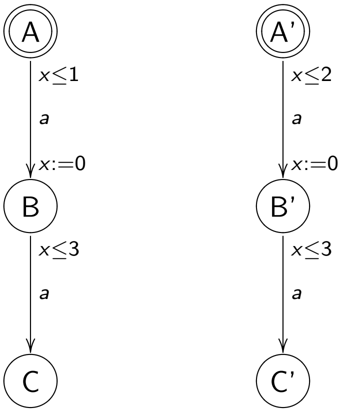
Remarque:

- $\xrightarrow{a}$ pour $a \in N$ est traitée comme une transition visible , mais
- $\xrightarrow{d}$ pour $d \in \mathbb{R}^{\geq 0}$ sont tous étiquetés par la même action visible $\xrightarrow{\epsilon}$.

Corollaire

Deux AT qui sont temporellement bisimilaires sont aussi bisimilaires non-temporellement.

des AT non temporellement bisimilaires qui sont bisimilaires non-temporellement



Décidabilité des bisimulations temporelles et non-temporelles

Théorème [Cerans'92]

La bisimulation temporelle pour les AT est décidable en EXPTIME (temps déterministic exponentiel).

Théorème [Larsen, Wang'93]

La bisimulation non-temporelle pour les AT est décidable en EXPTIME.

Traces temporisées

Soit $A = (L, \ell_0, E, I)$ un AT sur l'ensemble d'horloges C et l'ensemble d'étiquettes N.

Traces temporisées

Une suite $(t_1, a_1)(t_2, a_2)(t_3, a_3)\dots$ où $t_i \in \mathbb{R}^{\geq 0}$ et $a_i \in N$ est appelée une trace temporisée de A ssi il y a une suite de transitions

$$(\ell_0, v_0) \xrightarrow{d_1} . \xrightarrow{a_1} . \xrightarrow{d_2} . \xrightarrow{a_2} . \xrightarrow{d_3} . \xrightarrow{a_3} \dots$$

de A t.q. $v_0(x) = 0$ pour tous $x \in C$ et $t_i = t_{i-1} + d_i$ où $t_0 = 0$.

Intuition: t_i est le temps absolu (estampille de temps) datant l'événement a_i à partir du début de l'AT A.

Equivalence temporisée et non-temporisée des langages

L'ensemble des traces temporisées d'un AT A est notée $L(A)$ et est appelée le langage temporisé de A.

Théorème [Alur, Courcoubetis, Dill, Henzinger'94]

L'équivalence temporisée des langages (le problème de décider si $L(A_1) = L(A_2)$ pour des AT A_1 et A_2) est indécidable.

On dit que $a_1 a_2 a_3 \ldots$ est une trace non temporisée de A ssi il y a $t_1, t_2, t_3, \ldots \in \mathbb{R}^{\geq 0}$ such that tel que $(t_1, a_1)(t_2, a_2)(t_3, a_3) \ldots$ soit une trace temporisée de A.

Théorème [Alur, Dill'94]

L'équivalence non-temporisée des langages pour des AT A_1 et A_2 est décidable.