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

Buts

Décrire des outils mathématiques pour analyser les programmes parallèles

- Modéliser les programmes parallèles.
- Donner leur sémantique.
- Vérifier leurs propriétés.

Programme

- Systèmes de Transition et CCS.
- Sémantique opérationnelle de CCS.
- Logique de Hennessy-Milner et vérification.

Langages

Deux langages

- CCS : pour décrire les processus et leur sémantique.
- HML : Logique de Hennessy-Milner pour exprimer les propriétés des processus et les vérifier.

Sémantique

Qu'est-ce que c'est ?
Programme $\Longrightarrow$ Sens du programme
Programme $\Longrightarrow$ ce que fait le programme
Programme $\Longrightarrow$ Fonction $etats \hookrightarrow etats$

Vision Classique (programmes impératifs séquentiels)

Que fait un Programme ?

Un programme prend une donnée en entrée et rend une donnée en sortie.

- Sémantique :
  le sens d'un programme est une fonction

$$etats \hookrightarrow etats$$

- Un programme *termine* , sinon sa sémantique n'est pas définie.
- Le résultat d'un programme est unique.

Terminaison et unicité du résultat non garantis pour les processus.

Systèmes de Processus

Système de Processus

Un Système de Processus calcule par interaction/communication avec son environnement.

Différences avec les programmes séquentiels:

- communication et interaction
- non-unicité du résultat (non-déterminisme)
- parallélisme
- un programme qui ne termine pas peut fournir des résultats

Processus Modélisés

Les Processus Modélisés

- processus légers (type Thread en Java)
- mémoire partagée
- chaque variable existe en un seul exemplaire dans la mémoire
- à un instant donné, un seul processus accède à la mémoire
- le parallélisme est réalisé par interleaving (entrelacement) : pseudo-parallélisme

Processus *versus* Fonctions

Un processus n'est pas une fonction

Deux programmes qui représentent la même fonction :

$$\texttt{X:=2} \qquad \text{et} \qquad \texttt{X:=1 ; X:=X+1}$$

Si on les met en parallèle

$$\texttt{X:=2 } \| \texttt{X:=2}$$

termine toujours avec X=2 alors que c'est faux pour

$$\texttt{X:=2 } \| \texttt{ (X:=1 ; X:=X+1)}$$

Conclusion

La sémantique d'un système de processus ne peut pas être une fonction ... sinon on obtiendrait une sémantique non compositionnelle.

Alors ?

La sémantique d'un système de processus est un système de transition étiqueté (une "super-relation").

Systèmes de Transition étiquetés

Definition

Un Système de Transition étiqueté (ST) est un triplet $(Proc, Act, T)$ où

- $Proc$ est l'ensemble des états (ou processus),
- Act est l'ensemble des étiquettes (ou actions), et
- $T \subset (Proc, Act, Proc)$ est l'ensemble des transitions.

Notation : on écrit $s \xrightarrow{a} s'$ si $(s, a, s') \in T$.
s' est un dérivé de s s'il existe $s_1, \ldots, s_n \in Proc$ et $a_1, \ldots, a_n \in Act$ tels que

$$s \xrightarrow{a_1} s_1 \cdots \xrightarrow{a_n} s_n$$

et $s_n = s'$; notation $s \longrightarrow^* s'$

Que signifie $s \xrightarrow{a} s'$?

Intuition : le processus s interagit avec son environnement en faisant l'action a et ensuite il devient le processus s'.

Pour chaque $a \in Act$, $\xrightarrow{a} \subseteq Proc \times Proc$ est une relation binaire sur les états ; on note $s \xrightarrow{a} s'$ au lieu de $(s, s') \in \xrightarrow{a}$.
On a parfois besoin de l'état initial (ou start).

Séquentialité, Non-déterminisme et Parallélisme

Les ST décrivent les interactions d'un processus avec son environnement, mais aussi :

- la séquentialité $(a; b)$
- le non-déterminisme choix $(a + b)$
- le parallélisme (réduit à l'interleaving) $(a \| b)$

Exemple simple

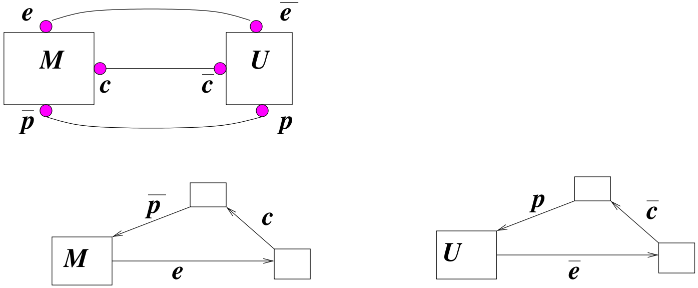


Figure: Une machine à café simple avec un utilisateur et les ST associés.

Descrition en CCS : $(U \| M) \setminus \{e, c, p\}$ avec
$M = e.c.\bar{p}.M$ et $U = \bar{e}.\bar{c}.p.U$

Un autre exemple

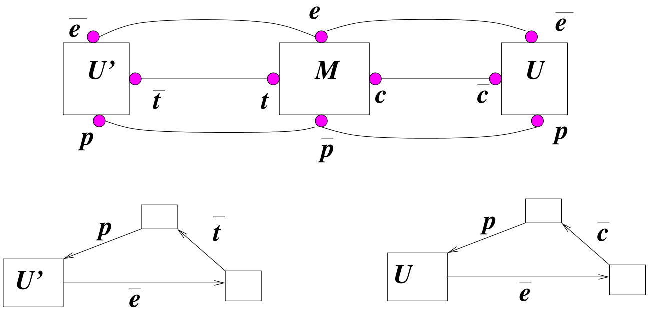
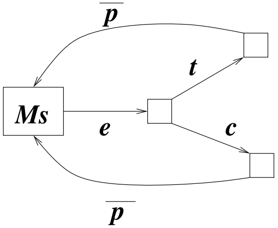


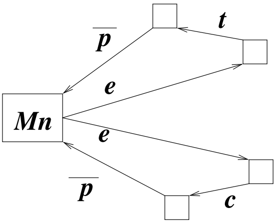
Figure: Une machine à café et à thé avec deux utilisateurs.

Description en CCS : $(U \| M \| U') \setminus \{e, c, p, t\}$ avec
... $U = \bar{e}.\bar{c}.p.U$ et $U' = \bar{e}.\bar{t}.p.U'$ et ...??

Un autre exemple

Les comportements possibles de la machine à café et à thé en système de transition





Description en CCS :

- Machine Sympa $M_s = e.(c.\bar{p}.M_s + t.\bar{p}.M_s)$
- Machine Non Sympa $M_n = e.c.\bar{p}.M_n + e.t.\bar{p}.M_n$

M_s et M_n sont deux machines différentes ... bien que les automates associés reconnaissent le même langage ...

Syntaxe versus Sémantique

Syntaxe	$\longrightarrow$	Sémantique
A trouver		connue
langage de programmation	$\longrightarrow$	que (denotationnel) ou comment (operationnel) calcule-t-il ?
??? CCS	$\longrightarrow$	Systèmes de Transition

Calculus of Communicating Systems

CCS

Une algèbre de Processus appelée "Calculus of Communicating Systems".

Idée de Robin Milner (1989)

Les processus parallèles ont une structure d'algèbre.

$$\boxed{P_1}\ op\ \boxed{P_2} \Rightarrow \boxed{P_1\ op\ P_2}$$

CCS Séquentiel non-déterministe

- *Nil* (ou 0) (processus atomique)
- préfixage par une action ($a.P$)
- noms et definitions récursives ($\stackrel{\text{def}}{=}$) (cf. λ du λ-calcul)
- choix non-déterministe ($+$)

Exemple :

$M_s \stackrel{\text{def}}{=} e.(c.\bar{p}.M_s + t.\bar{p}.M_s)$

$U \stackrel{\text{def}}{=} \bar{e}.\bar{c}.p.U$ et $U' \stackrel{\text{def}}{=} \bar{e}.\bar{t}.p.U'$

CCS (Parallélisme et Renommage)

- composition parallèle ($\|$)
  (communication synchrone = handshake ou rendez-vous)
- restriction ($P \setminus L$) (L est un ensemble de labels)
- renommage ($P[f]$)

Définition de CCS

Soit

- $\mathcal{A}$ un ensemble de noms de canaux (e.g. t, c sont des noms de canaux)
- $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ un ensemble de labels où
  - $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$
    ($\mathcal{A}$ sont les noms et $\overline{\mathcal{A}}$ sont les co-noms)
  - par convention $\overline{\overline{a}} = a$
- $Act = \mathcal{L} \cup \{\tau\}$ est l'ensemble des actions où
  - τ est l'action interne ou invisible

  (e.g. τ, t, $\overline{c}$ sont des actions)
- $\mathcal{K}$ un ensemble de symboles de noms de processus (e.g. U, U', M, M_s, M_n).

Syntaxe de CCS : les expressions CCS

$P :=$	K	$\vert$	symbole de nom de processus ($K \in \mathcal{K}$)
	$\alpha.P$	$\vert$	préfixage ($\alpha \in Act$)
	$\sum_{i\in I} P_i$	$\vert$	sommation (I ensemble d'indices arbitraire)
	$P_1 \Vert P_2$	$\vert$	composition parallèle
	$P \setminus L$	$\vert$	restriction ($L \subseteq \mathcal{A}$)
	$P[f]$	$\vert$	renommage ($f : Act \rightarrow Act$) tel que ▶ $f(\tau) = \tau$ ▶ $f(\overline{a}) = \overline{f(a)}$

L'ensemble des termes ainsi engendré est appelé l'ensemble des expressions CCS (et noté $\mathcal{P}$).

Notation

$$P_1 + P_2 = \sum_{i\in\{1,2\}} P_i \qquad\qquad Nil = 0 = \sum_{i\in\emptyset} P_i$$

Priorités de parenthésage

Dans l'ordre

1. restriction et renommage (plus forte priorité)
2. préfixage par une action
3. composition parallèle
4. sommation

Exemple: $R + a.P \| b.Q \setminus L$ s'écrit aussi $R + \big((a.P) \| (b.(Q \setminus L))\big)$.

Syntaxe de CCS : les programmes CCS

Programme CCS

Un programme CCS est un ensemble d'équations de la forme

$$K \stackrel{\text{def}}{=} P$$

où $K \in \mathcal{K}$ est un symbole de nom de processus et $P \in \mathcal{P}$ est une expression CCS.

- Une et une seule équation par nom de processus.
- La récursion est autorisée: e.g. $U = \bar{e}.\bar{c}.p.U$.

Sémantique de CCS

Syntaxe

CCS

(un programme CCS :ensemble d'équations)

$\longrightarrow$

Sémantique

ST

(un système de transition)

Comment?

- dénotationnelle : non car pas de compositionnalité
- opérationnelle : oui

Sémantique Opérationnelle Structurée (SOS) – G. Plotkin 1981

Sémantique Opérationnelle : le comportement d'un programme est calculé en utilisant des règles dirigées par sa syntaxe.

Etant donné un programme CCS, sa sémantique est le ST $(Proc, Act, T)$:

- $Proc = \mathcal{P}$ (l'ensemble des expressions CCS)
- $Act = \mathcal{L} \cup \{\tau\}$ (l'ensemble des actions CCS incluant τ)
- les transitions sont données par des règles SOS de la forme :

$$\text{REGLE} \quad \frac{premisses}{conclusion} \quad conditions$$

Règles SOS pour CCS ($\alpha \in Act$, $a \in \mathcal{L}$)

$$\text{ACT} \quad \frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad\qquad \text{SUM}_j \quad \frac{P_j \xrightarrow{\alpha} P_j'}{\sum_{i \in I} P_i \xrightarrow{\alpha} P_j'} \quad j \in I$$

$$\text{COM1} \quad \frac{P \xrightarrow{\alpha} P'}{P \| Q \xrightarrow{\alpha} P' \| Q} \qquad\qquad \text{COM2} \quad \frac{Q \xrightarrow{\alpha} Q'}{P \| Q \xrightarrow{\alpha} P \| Q'}$$

$$\text{COM3} \quad \frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\overline{a}} Q'}{P \| Q \xrightarrow{\tau} P' \| Q'}$$

$$\text{RES} \quad \frac{P \xrightarrow{\alpha} P'}{P \setminus L \xrightarrow{\alpha} P' \setminus L} \quad \alpha, \overline{\alpha} \notin L \qquad\qquad \text{REN} \quad \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$$

$$\text{REC} \quad \frac{P \xrightarrow{\alpha} P'}{K \xrightarrow{\alpha} P'} \quad K \stackrel{\text{def}}{=} P$$

Dérivation de Transitions dans CCS

Soit $A \stackrel{\mathrm{def}}{=} a.A$.

$$\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a].$$

Dérivation de Transitions dans CCS

Soit $A \stackrel{\text{def}}{=} a.A$.

$$\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a].$$

$$\text{REN}\ \frac{}{\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a]}$$

Soit $A \stackrel{\text{def}}{=} a.A$.

$$\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a].$$

$$\text{REN}\ \dfrac{\text{COM1}\ \dfrac{}{(A \parallel \overline{a}.Nil) \parallel b.Nil \xrightarrow{a} (A \parallel \overline{a}.Nil) \parallel b.Nil}}{\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a]}$$

Dérivation de Transitions dans CCS

Soit $A \stackrel{\text{def}}{=} a.A$.

$$\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a].$$

$$\text{REN}\ \dfrac{\text{COM1}\ \dfrac{\text{COM1}\ \dfrac{}{A \parallel \overline{a}.Nil \xrightarrow{a} A \parallel \overline{a}.Nil}}{(A \parallel \overline{a}.Nil) \parallel b.Nil \xrightarrow{a} (A \parallel \overline{a}.Nil) \parallel b.Nil}}{\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a]}$$

Dérivation de Transitions dans CCS

Soit $A \stackrel{\text{def}}{=} a.A$.

$$\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a].$$

$$
\text{REN}\ \cfrac{\text{COM1}\ \cfrac{\text{COM1}\ \cfrac{\text{REC}\ \cfrac{}{A \xrightarrow{a} A}\ A \stackrel{\text{def}}{=} a.A}{A \parallel \overline{a}.Nil \xrightarrow{a} A \parallel \overline{a}.Nil}}{(A \parallel \overline{a}.Nil) \parallel b.Nil \xrightarrow{a} (A \parallel \overline{a}.Nil) \parallel b.Nil}}{\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a]}
$$

Dérivation de Transitions dans CCS

Soit $A \stackrel{\text{def}}{=} a.A$.

$$\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a].$$

$$
\text{REN}\;\dfrac{\text{COM1}\;\dfrac{\text{COM1}\;\dfrac{\text{REC}\;\dfrac{\text{ACT}\;\dfrac{}{a.A \xrightarrow{a} A}}{A \xrightarrow{a} A}\;A \stackrel{\text{def}}{=} a.A}{A \parallel \overline{a}.Nil \xrightarrow{a} A \parallel \overline{a}.Nil}}{(A \parallel \overline{a}.Nil) \parallel b.Nil \xrightarrow{a} (A \parallel \overline{a}.Nil) \parallel b.Nil}}{\big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a] \xrightarrow{c} \big((A \parallel \overline{a}.Nil) \parallel b.Nil\big)[c/a]}
$$

ST du Processus $a.Nil \parallel \overline{a}.Nil$

$$
\begin{tikzcd}
 & a.Nil \parallel \overline{a}.Nil \arrow[dl, "a"'] \arrow[dd, "\tau"] \arrow[dr, "\overline{a}"] & \\
Nil \parallel \overline{a}.Nil \arrow[dr, "\overline{a}"] & & a.Nil \parallel Nil \arrow[dl, "a"'] \\
 & Nil \parallel Nil &
\end{tikzcd}
$$

Machines à café

$U \stackrel{\text{def}}{=} \bar{e}.\bar{c}.p.U$ et $U' \stackrel{\text{def}}{=} \bar{e}.\bar{t}.p.U'$

- machine sympa
  $M_s \stackrel{\text{def}}{=} e.(c.\bar{p}.M_s + t.\bar{p}.M_s)$ et $P_s \stackrel{\text{def}}{=} (M_s \| U \| U') \setminus \{c, e, p, t\}$
  $P_s \xrightarrow{\tau} ((c.\bar{p}.M_s + t.\bar{p}.M_s) \| \bar{c}.p.U \| U') \setminus \{c, e, p, t\} \xrightarrow{\tau}$
  $((\bar{p}.M_s) \| p.U \| U') \setminus \{c, e, p, t\} \xrightarrow{\tau} (M_s \| U \| U') \setminus \{c, e, p, t\} = P_s$
- machine non sympa
  $M_n \stackrel{\text{def}}{=} e.c.\bar{p}.M_n + e.t.\bar{p}.M_n$ et $P_n \stackrel{\text{def}}{=} (M_n \| U \| U') \setminus \{c, e, p, t\}$
  $P_s \xrightarrow{\tau} (c.\bar{p}.M_s \| \bar{c}.p.U \| U') \setminus \{c, e, p, t\} \not\longrightarrow$ blocage

Vérification de processus CCS

Soit *Impl* une implémentation par un programme CCS d'un système et *Prop* une propriété. Est-ce que *Impl* vérifié *Prop* ?

Model Checking

$$\mathit{Impl} \models \mathit{Prop}$$

- $\models$ relation de satisfaction
- *Prop* une propriété, exprimée dans une logique
- *Prop* peut aussi être une spécification du comportement du système

But

Développer une logique où l'on puisse exprimer des propriétés des programmes CCS.

Propriétés des processus CCS

Propriétés Modales – ce qui **peut/doit** arriver **maintenant**

- demander café
- donner café
- donner thé et café

Propriétés temporelles – ce qui **peut/doit** arriver **un jour/jamais**

- ne boit jamais de café
  (sureté (safety): rien de mauvais n'arrivera)
- boira un jour du thé
  (vivacité (liveness): une bonne chose arrivera un jour)

Peut-on exprimer ces propriétés ?

Logique de Hennessy-Milner Logic – Syntaxe

Syntaxe des Formules ($a \in Act$)

$$F, G \ ::= \ tt \ | \ ff \ | \ F \wedge G \ | \ F \vee G \ | \ \langle a \rangle F \ | \ [a]F$$

Intuition:

- tt tous les processus satisfont cette propriété
- ff aucun processus ne satisfait cette propriété
- $\wedge, \vee$ ET et OU usuels
- $\langle a \rangle F$ il y a au moins un a-successeur satisfaisant F
- $[a]F$ tous les a-successeurs satisfont F

Remarque

Les propriétés temporelles – ce qui peut/doit arriver **un jour/jamais/toujours** ne sont pas exprimées.

Logique de Hennessy-Milner – Sémantique

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

Validité de $p \models F$ ($p \in Proc$, F une formule HML)

$$
\begin{array}{rl}
p \models tt & \text{pour tout } p \in Proc \\
p \models ff & \text{pour aucun } p \text{ (on écrit aussi } p \not\models ff\text{)} \\
p \models F \wedge G & \text{ssi } p \models F \text{ et } p \models G \\
p \models F \vee G & \text{ssi } p \models F \text{ ou } p \models G \\
p \models \langle a \rangle F & \text{ssi } p \xrightarrow{a} p' \text{ pour un } p' \in Proc \text{ tel que } p' \models F \\
p \models [a]F & \text{ssi } p' \models F \text{, pour tout } p' \in Proc \text{ tel que } p \xrightarrow{a} p'
\end{array}
$$

On écrit $p \not\models F$ ssi p ne satisfait pas F.

Négation

Pour toute formule F on définit la formule F^c :

- $tt^c = ff$
- $ff^c = tt$
- $(F \wedge G)^c = F^c \vee G^c$
- $(F \vee G)^c = F^c \wedge G^c$
- $(\langle a \rangle F)^c = [a]F^c$
- $([a]F)^c = \langle a \rangle F^c$

Théorème (F^c est équivalente à la négation de F)

Pour $p \in Proc$ et F formule HML

$$p \models F \iff p \not\models F^c$$

Logique de Hennessy-Milner – Sémantique Dénotationnelle

Soit F une formule et soit $[\![F]\!] \subseteq Proc$ l'ensemble des états (processus) qui satisfont F.

Sémantique Dénotationnelle : $[\![\_]\!] : Formulae \longrightarrow 2^{Proc}$

- $[\![tt]\!] = Proc$
- $[\![ff]\!] = \emptyset$
- $[\![F \vee G]\!] = [\![F]\!] \cup [\![G]\!]$
- $[\![F \wedge G]\!] = [\![F]\!] \cap [\![G]\!]$
- $[\![\langle a \rangle F]\!] = \langle \cdot a \cdot \rangle [\![F]\!]$
- $[\![[a]F]\!] = [\cdot a \cdot][\![F]\!]$

Exemples

où : $\langle\cdot a\cdot\rangle, [\cdot a\cdot] : 2^{(Proc)} \to 2^{(Proc)}$ sont définis par

$$\langle\cdot a\cdot\rangle S = \{p \in Proc \mid \exists p'.\ p \xrightarrow{a} p' \text{ and } p' \in S\}$$

$$[\cdot a\cdot] S = \{p \in Proc \mid \forall p'.\ p \xrightarrow{a} p' \implies p' \in S\}.$$

Exemple : $U \stackrel{\text{def}}{=} \bar{e}.\bar{c}.p.U$

- $F = \langle\bar{e}\rangle\langle\bar{c}\rangle tt$ $\qquad U \models F$
- $F = [\bar{e}][\bar{c}] tt$ $\qquad U \models F$
- $F = \langle\bar{e}\rangle\langle\bar{t}\rangle tt$ $\qquad U \not\models F$

Exemples

$$M_s \stackrel{\text{def}}{=} e.(c.\bar{p}.M_s + t.\bar{p}.M_s) \quad \text{et} \quad M_n \stackrel{\text{def}}{=} e.c.\bar{p}.M_n + e.t.\bar{p}.M_n$$

$$\begin{array}{rl}
F = \langle e\rangle\langle c\rangle tt & \quad M_s \models F \text{ et } M_n \models F \\
F = [e]\langle c\rangle tt & \quad M_s \models F \text{ et } M_n \not\models F \\
F = [e][c]tt & \quad M_s \not\models F \text{ et } M_n \not\models F \\
F = \langle e\rangle(\langle c\rangle \wedge \langle t\rangle tt) & \quad M_s \models F \text{ et } M_n \not\models F
\end{array}$$

Théorème d'adéquation

théorème

Soit $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ un ST, $p \in Proc$ et F une formule de Hennessy-Milner.

$$p \models F \quad \text{si et seulement si} \quad p \in [\![F]\!].$$

Preuve: induction structurelle sur la structure de la formule F.

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 aucun état accessible depuis s ne satisfait F

- Aucune formule HML ne peut détecter (l'absence de) blocage dans un ST arbitraire
- Les propriétés $Inv(F)$ et $Pos(F)$ ne sont pas exprimables par des formules HML

Pour aller plus loin :

- formules infinies ..
- récursion