Systèmes de transitions

Philippe Quéinnec, Xavier Thirioux, Aurélie Hurault

ENSEEIHT
Département Sciences du Numérique

Méthodes formelles ?

Contexte

- Système critique, dont la défaillance entraîne des conséquences graves (exemple : médical, transport)
- Système complexe, dont il est difficile de se convaincre de la correction (exemple : systèmes concurrents)

Pourquoi ?

- Nécessité de prouver qu'un algorithme / un système possède bien les propriétés attendues
- C'est dur $\Rightarrow$ nécessité de cadres formels précis et d'outils

Comment ?

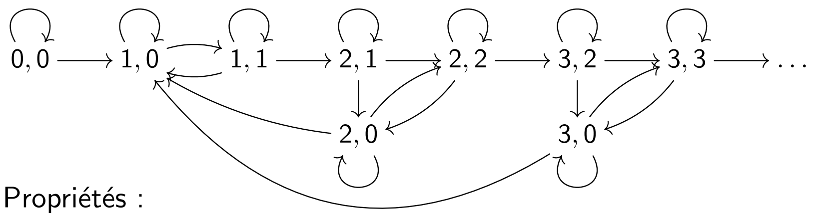
- Langage impératif classique :
  état = valeurs des variables + *flot de contrôle implicite*
- Système de transitions :
  état = valeurs des variables + *flot de contrôle explicite*

Exemple

Soit trois processus exécutant concurrentment (par entrelacement) :

boucle $x \leftarrow y+1$ $\|$ boucle $y \leftarrow x$ $\|$ boucle $y \leftarrow 0$

1. Description du système en termes d'états ?



2. Propriétés :
   - L'état 4, 2 est-il accessible ?
   - Le système s'arrête-t-il ? Toujours, parfois ?
   - Est-il toujours vrai que $y = 0 \vee 0 \leq x - y \leq 1$?
   - Si $y = 6$, est-il possible/nécessaire que x devienne > 6 ?
   - Est-il possible/nécessaire que y soit non borné ?

Approche TLA$^+$

Temporal Logic of Actions

1. Un langage de spécification logique (LTL / Logique temporelle linéaire) $\approx$ quelles sont les propriétés attendues ?
2. Un langage d'actions $\approx$ un langage de spécification plus opérationnel $\approx$ un langage de programmation
3. (en fait, langage de spécification = langage d'actions)
4. Cadre formel = système de transitions
5. Outils : vérificateur automatique, assistant de preuve

Auteur principal : Leslie Lamport

Plan du cours

1. (C) Le cadre formel : systèmes de transitions
2. (CTD) Spécification opérationnelle : TLA$^+$ – les actions (1)
3. (CTD) Spécification opérationnelle : TLA$^+$ – les actions (2)
4. (TP) Recherche de solutions par accessibilité
5. (C) Contrôler la progression : l'équité
6. (C) Énoncer des propriétés : la logique temporelle linéaire LTL
7. (CTD) Logique temporelle et équité dans TLA$^+$
8. (CTD) Étude d'un algorithme d'exclusion mutuelle
9. (CTD,TP) Étude d'un algorithme distribué
10. (C) Énoncer des propriétés : logique temporelle arborescente CTL
11. (TP) Concurrence : allocation de ressources
12. (C) Vérification par preuve et vérification de modèles
13. (TP) Étude d'un protocole de communication en mémoire partagée

Ressources

- http://queinnec.perso.enseeiht.fr/Ens/st.html
  supports de cours, TP, examens
- http://lamport.azurewebsites.net/video/videos.html
  vidéos de L. Lamport sur TLA$^+$
- http://lamport.azurewebsites.net/tla/tla.html
  autres ressources (livre *Specifying Systems*)
- https://learntla.com/
  guide d'introduction à TLA$^+$ (exemples surtout en PlusCal)

Première partie

Systèmes de transitions

Introduction

Objectifs

Représenter les exécutions d'un algorithme en faisant abstraction de certains détails :

- les détails sont la cause d'une explosion du nombre d'états et de la complexité des traitements.
- ne conserver que ce qui est pertinent par rapport aux propriétés attendues.

Utilisation

Un système de transitions peut être construit :

- *avant* l'écriture du programme, pour explorer la faisabilité de celui-ci.
  Le programme final est un raffinement en utilisant le système de transitions comme guide.
- *après* l'écriture du programme, par abstraction, en ne conservant que les aspects significatifs du programme réel.

Plan

Système de transitions ♪♪♪

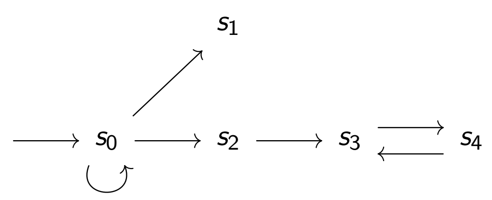
Système de transitions (ST)

Un système de transitions est un triplet $\langle S, I, R \rangle$.

- S : ensemble d'états. Peut être fini ou infini.
- $I \subseteq S$: ensemble des états initiaux.
- $R \subseteq S \times S$: relation (de transitions) entre paires d'états.
  $(s, s') \in R$ signifie qu'il existe une transition faisant passer le système de l'état s à l'état s'.

Exemple - système de transitions

$S = \{s_0, s_1, s_2, s_3, s_4\}$
$I = \{s_0\}$
$R = \{(s_0, s_0), (s_0, s_1), (s_0, s_2), (s_2, s_3), (s_3, s_4), (s_4, s_3)\}$



Séquences

Séquence

Soit S un ensemble.
$S^\star \triangleq$ l'ensemble des séquences finies sur S.
$S^\omega \triangleq$ l'ensemble des séquences infinies sur S.
$\sigma_i \triangleq$ le $i^{\text{ème}}$ (à partir de 0) élément d'une séquence σ.

Conventions de représentation :

- Une séquence s est notée sous la forme : $\langle s_1 \to s_2 \to \ldots \rangle$.
- $\langle\rangle$: la séquence vide.

Pour une séquence finie σ :

- $\sigma^\star \triangleq$ l'ensemble des séquences finies produites par la répétition arbitraire de σ.
- $\sigma^+ \triangleq \sigma^* \setminus \{\langle\rangle\}$
- $\sigma^\omega \triangleq$ la séquence infinie produite par la répétition infinie de σ.

Traces finies

Traces finies

Soit $\langle S, I, R\rangle$ un système de transitions.
On appelle trace finie une séquence finie $\sigma \in S^\star$ telle que :

- $\sigma = \langle s_0 \to s_1 \to \ldots \to s_{n-1} \to s_n \rangle$
- $\forall i \in [0..n[: (s_i, s_{i+1}) \in R$

Traces finies maximales

Soit $\langle S, I, R\rangle$ un système de transitions.
Une trace finie $\langle s_0 \to s_1 \to \ldots \to s_{n-1} \to s_n \rangle \in S^\star$ est maximale $\triangleq$ il n'existe pas d'état successeur à s_n, i.e. $\forall s \in S : (s_n, s) \notin R$.

Une trace maximale va le plus loin possible.

Traces infinies et traces issues d'un état

Traces infinies

Soit $\langle S, I, R\rangle$ un système de transitions, et $s_0 \in S$.
On appelle trace infinie à partir de s_0 un élément $tr \in S^\omega$ tel que :

- $tr = \langle s_0 \to s_1 \to s_2 \ldots \rangle$
- $\forall i \in \mathbb{N} : (s_i, s_{i+1}) \in R$

Traces issues d'un état

Soit $\langle S, I, R\rangle$ un système de transitions, et $s \in S$.
$Traces(s) \triangleq$ l'ensemble des traces infinies ou finies maximales commençant à l'état s.

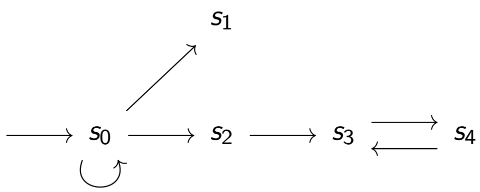
Exécutions

Exécutions

Soit $\mathcal{S} = \langle S, I, R\rangle$ un système de transitions.
Une exécution $\sigma = \langle s_0 \to \ldots \rangle$ est une trace infinie ou finie maximale telle que $s_0 \in I$.
$Exec(\mathcal{S}) \triangleq$ l'ensemble des exécutions de $\mathcal{S} = \bigcup_{s_0 \in I} Traces(s_0)$.

On a une (seule et unique) exécution vide $\langle\rangle$ ssi $I = \emptyset$.

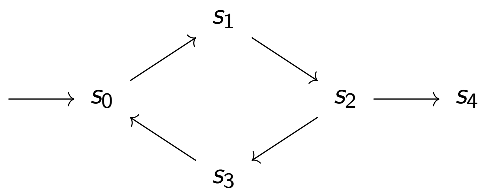
Exemple - traces, exécutions



$s_0 \to s_0 \to s_2 \to s_3$ est une trace finie non maximale

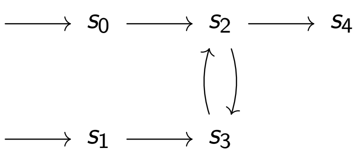
$$
\begin{array}{lcl}
Traces(s_1) & = & \langle s_1 \rangle \\
Traces(s_3) & = & \langle (s_3 \to s_4)^\omega \rangle \\
Traces(s_2) & = & \langle s_2 \to (s_3 \to s_4)^\omega \rangle \\
Traces(s_0) & = & \langle s_0{}^\omega \rangle, \langle s_0{}^+ \to s_1 \rangle, \langle s_0{}^+ \to s_2 \to (s_3 \to s_4)^\omega \rangle \\
Exec(\mathcal{S}) & = & Traces(s_0)
\end{array}
$$

Exemple 2 - traces, exécutions



$$
\begin{array}{lcl}
Traces(s_2) & = & \\
Traces(s_0) & = & \\
Exec(\mathcal{S}) & = &
\end{array}
$$

Exemple 3 - traces, exécutions



$$
\begin{array}{lcl}
Traces(s_2) & = & \\
Traces(s_0) & = & \\
Traces(s_1) & = & \\
Exec(\mathcal{S}) & = &
\end{array}
$$

États accessibles

État accessible

Soit $\mathcal{S} = \langle S, I, R \rangle$ un système de transitions.
$s \in S$ est un état accessible $\triangleq$ il existe une exécution qui passe par s (ou équivalent, il existe un préfixe d'exécution qui aboutit à s).
$Acc(\mathcal{S}) \triangleq$ l'ensemble des états accessibles de $\mathcal{S}$.

Graphe des exécutions

Graphe des exécutions

Soit $\mathcal{S} = \langle S, I, R \rangle$ un système de transitions.
Le graphe des exécutions est le graphe orienté où :

- l'ensemble des sommets est $Acc(\mathcal{S})$;
- l'ensemble des arêtes orientées est R, restreint aux seuls états accessibles.

Il s'agit donc du graphe $\langle S \cap Acc(\mathcal{S}), R \cap (Acc(\mathcal{S}) \times Acc(\mathcal{S})) \rangle$.

Système de transitions étiqueté ♪♪♪

ST étiqueté

Un système de transitions étiqueté est un quintuplet $\langle S, I, R, L, Etiq \rangle$.

- S : ensemble d'états.
- $I \subseteq S$: ensemble des états initiaux.
- $R \subseteq S \times S$: relation de transitions entre paires d'états.
- L : ensemble d'étiquettes.
- $Etiq$: fonction qui associe une étiquette à chaque transition : $Etiq \in R \to L$.

Un ST étiqueté se rapproche beaucoup des automates.
Mais : pas d'état terminal + exécution infinie.

Équivalence aux ST sans étiquette ♪♪♪

Un système de transitions étiqueté $\langle S, I, R, L, Etiq \rangle$ est équivalent au système sans étiquette $\langle S', I', R' \rangle$ défini par :

- $S' = (L \cup \{\epsilon\}) \times S$
- $I' = \{\epsilon\} \times I$
- $R' = \{(\langle l, s \rangle, \langle l', s' \rangle) \mid (s, s') \in R \wedge l' = Etiq(s, s')\}$

Une transition $s_1 \xrightarrow{a} s_2$ devient $\langle \_, s_1 \rangle \longrightarrow \langle a, s_2 \rangle$,
où _ est n'importe quelle étiquette.

Exemple - équivalence avec/sans étiquette

$$\begin{tikzcd}
 & & s_1 \arrow[dr, "c"] & \\
{} \arrow[r] & s_0 \arrow[ur, "a"] \arrow[rr, "c"] \arrow[dr, "b"'] & & s_3 \\
 & & s_2 \arrow[ur, "d"'] &
\end{tikzcd}$$

$$\begin{tikzcd}
 & & \langle a, s_1 \rangle \arrow[r] & \langle c, s_3 \rangle \\
{} \arrow[r] & \langle \epsilon, s_0 \rangle \arrow[ur] \arrow[urr, bend right] \arrow[dr] & & \\
 & & \langle b, s_2 \rangle \arrow[r] & \langle d, s_3 \rangle
\end{tikzcd}$$

Différences entre système de transition et automate

Système de transitions $\neq$ automate

- Pas d'étiquette sur les transitions (ou comme si)
- Une transition n'est pas causée par l'environnement
- Pas d'états terminaux
- Nombre d'états infini possible
- Exécution infinie possible

Plan

Représentation en extension ♪♪♪

Représentation en extension

Donnée en extension du graphe des exécutions, par exemple sous forme graphique ou par l'ensemble des sommets et arêtes.

Ne convient que pour les systèmes de transitions où le nombre d'états et de transitions est fini.

Représentation en intention ♪♪♪

Représentation symbolique à l'aide de variables.

Système de transitions à base de variables

Un triplet $\langle V, Init, Trans \rangle$ où

- $V = \{v_1, \ldots, v_n\}$: ensemble fini de variables.
- $Init(v_1, \ldots, v_n)$: prédicat définissant les états initiaux et portant sur les variables v_i.
- $Trans(v_1, \ldots, v_n, v'_1, \ldots, v'_n)$: prédicat définissant les transitions, portant sur les variables v_i représentant l'état courant et les variables v'_i représentant l'état suivant.

Exemple : un compteur borné ♪♪♪

```
i = 0;
while (i < N) {
  i = i+1;
}
```

En extension pour $N = 5$:
$\langle (0,1,2,3,4,5), \{0\}, \{(0,1),(1,2),(2,3),(3,4),(4,5)\}\rangle$

Graphe d'exécution pour $N = 5$:

$$\longrightarrow 0 \longrightarrow 1 \longrightarrow 2 \longrightarrow 3 \longrightarrow 4 \longrightarrow 5$$

Symboliquement (en intention) :

$V \triangleq i \in \mathbb{N}$
$I \triangleq i = 0$
$T \triangleq i < N \land i' = i + 1$ ou $T \triangleq i' \leq N \land i' - i = 1$

Exemple : un compteur cyclique

```
i = 0;
while (true) {
  i = (i+1) % N;
}
```

En extension pour $N = 5$:
$\langle (0,1,2,3,4,5), \{0\}, \{(0,1),(1,2),(2,3),(3,4),(4,0)\}\rangle$

Graphe d'exécution pour $N = 5$.

$$\begin{tikzcd} {} \arrow[r] & 0 \arrow[r] & 1 \arrow[r] & 2 \arrow[r] & 3 \arrow[r] & 4 \arrow[llll, bend left] \end{tikzcd}$$

Symboliquement :

$V \triangleq i \in \mathbb{N}$
$I \triangleq i = 0$
$T \triangleq i' = (i + 1) \bmod N$

Exemple : un entier oscillant ♪♪♪

```
i = 0;
while (true) {
      i > 0 -> i = i - 1;
  or  i < N -> i = i + 1;
}
```

En extension pour $N = 5$: $\langle (0,1,2,3,4,5), \{0\}$,
$\{(0,1),(1,0),(1,2),(2,1),(2,3),(3,2),(3,4),(4,3),(4,5),(5,4)\}\rangle$

Graphe d'exécution pour $N = 5$.

$$\begin{tikzcd} {} \arrow[r] & 0 \arrow[r, bend left] & 1 \arrow[l, bend left] \arrow[r, bend left] & 2 \arrow[l, bend left] \arrow[r, bend left] & 3 \arrow[l, bend left] \arrow[r, bend left] & 4 \arrow[l, bend left] \arrow[r, bend left] & 5 \arrow[l, bend left] \end{tikzcd}$$

Symboliquement :

$V \triangleq i \in \mathbb{N}$
$I \triangleq i = 0$
$T \triangleq \quad i > 0 \land i' = i - 1$ ou $T \triangleq |i' - i| = 1 \land 0 \leq i' \leq N$
$\quad \lor\ i < N \land i' = i + 1$

Système de transition correspondant ♪♪♪

Pour une description symbolique $\langle V, Init, Trans\rangle$, le système de transitions correspondant est $\langle S, I, R\rangle$ où :

- $S = \prod_{i \in 1..n} D_i$,
  où $D_1, \ldots, D_n$ sont les domaines (types) des variables $v_1, \ldots, v_n$
- $I = \{(v_1, \ldots, v_n) \mid Init(v_1, \ldots, v_n)\}$
- $R = \{((v_1, \ldots, v_n), (v'_1, \ldots, v'_n)) \mid Trans(v_1, \ldots, v_n, v'_1, \ldots, v'_n)\}$

Prédicats

Prédicat d'état

Un prédicat d'état est un prédicat portant sur les variables (d'état) d'un système donné en intention.

Un prédicat d'état peut être vu comme la fonction caractéristique d'une partie de S.

Prédicat de transition

Un prédicat de transitions est un prédicat portant sur les variables (d'état) primées et non primées.

Un prédicat de transitions peut être vu comme la fonction caractéristique d'une partie de $S \times S$.

Exemple - prédicats

$$
\begin{array}{lcl}
V & \triangleq & n \in \mathbb{N} \\
I & \triangleq & -5 \leq n \leq 5 \\
T & \triangleq & n \neq 1 \;\wedge\; \left(\begin{array}{l} (n' = n/2 \wedge n \equiv 0[2]) \\ \vee (n' = (3n+1)/2 \wedge n \equiv 1[2]) \end{array} \right)
\end{array}
$$

Prédicats d'état : I, $n < 20$
Prédicats de transition : T, $n' - n > 3$

Plan

Blocage

Interblocage

Un système possède un interblocage (deadlock) $\triangleq$ il existe un état accessible sans successeur par la relation R.
De manière équivalente un système possède un interblocage s'il existe des exécutions finies.

Pour les systèmes modélisant des programmes séquentiels classiques, l'interblocage est équivalent à la terminaison.

Réinitialisable

Réinitialisable

Un système est réinitialisable $\triangleq$ depuis tout état accessible, il existe une trace finie menant à un état initial.

Cette propriété signifie qu'à n'importe quel moment, il existe une séquence de transitions pour revenir à l'état initial du système et ainsi redémarrer. Un tel système n'a que des exécutions infinies.

Bégaiement ♪♪♪

Bégaiement

Un état s bégaie $\triangleq$ l'état possède une boucle : $(s, s) \in R$.
Un système de transitions bégaie $\triangleq$ tout état possède une boucle vers lui-même : $Id \subseteq R$.

Utilité :

- Modéliser l'avancement arbitraire : $\longrightarrow s_0 \longrightarrow s_1$
  on peut aller en s_1 après être resté arbitrairement longtemps en s_0.
- N'avoir que des exécutions infinies : tout état sans successeur (dans un système sans bégaiement) a un unique successeur avec bégaiement : lui-même. La terminaison (l'interblocage) $\ldots \rightarrow s_i$ est alors $\ldots \rightarrow s_i{}^{\omega}$.
- Composer plusieurs systèmes de transitions.

Plan

Composition : produit libre ♪♪♪

Produit libre

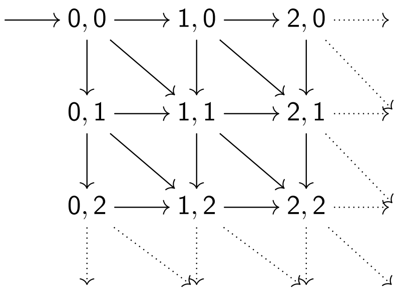
La composition des ST avec bégaiement $\langle V_1, I_1, T_1\rangle$ et $\langle V_2, I_2, T_2\rangle$ est $\langle V, I, T\rangle$ où :

- $V \triangleq V_1 \cup V_2$ (union des variables)
- $I \triangleq I_1 \wedge I_2$ (chaque sous-système démarre dans un de ses états initiaux)
- $T \triangleq T_1 \wedge T_2$ (chaque sous-système évolue selon ses transitions)

Comme T_1 et T_2 peuvent bégayer, $T_1 \wedge T_2$ signifie donc qu'on peut exécuter une transition de T_1 seule et T_2 bégayant, ou bien réciproquement, ou bien encore exécuter T_1 en même temps que T_2.

Exemple - produit libre

$$\begin{pmatrix} V_1 \triangleq i \in \mathbb{N} \\ I_1 \triangleq i = 0 \\ T_1 \triangleq \; i' = i+1 \\ \qquad \vee\; i' = i \end{pmatrix} \otimes \begin{pmatrix} V_2 \triangleq j \in \mathbb{N} \\ I_2 \triangleq j = 0 \\ T_2 \triangleq \; j' = j+1 \\ \qquad \vee\; j' = j \end{pmatrix} \to \begin{pmatrix} V \triangleq i,j \in \mathbb{N} \\ I \triangleq i = 0 \wedge j = 0 \\ T \triangleq \quad i' = i+1 \wedge j' = j \\ \qquad \vee i' = i \wedge j' = j+1 \\ \qquad \vee i' = i+1 \wedge j' = j+1 \\ \qquad \vee i' = i \wedge j' = j \end{pmatrix}$$



+bégaiement

Composition : produit synchronisé strict (ou fermé) ♪♪♪

Produit synchronisé strict

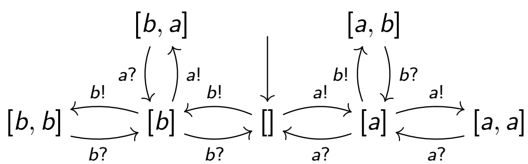
Le produit synchrone des ST étiquetés $\langle S_1, I_1, R_1, L_1\rangle$ et $\langle S_2, I_2, R_2, L_2\rangle$ est $\langle S, I, R, L\rangle$ où :

- $S \triangleq S_1 \times S_2$ (couple d'états)
- $I \triangleq I_1 \times I_2$
  (chaque sous-système démarre dans un de ses états initiaux)
- $R \triangleq \{((s_1, s_2), (s_1', s_2')) \mid (s_1, s_1') \in R_1 \wedge (s_2, s_2') \in R_2 \wedge Etiq((s_1, s_1')) = Etiq((s_2, s_2'))\}$
  (les deux sous-systèmes évoluent selon des transitions portant les mêmes étiquettes)
- $L = L_1 \cap L_2$ (étiquettes communes seulement)

Exemple – produit synchronisé strict ♪♪♪

$$\begin{tikzcd} {} \arrow[r] & e_1 \arrow[r, "a!"] & e_2 \arrow[r, "b!"] & e_3 \arrow[r, "a?"] \arrow[dr, "b?"'] & e_4 \arrow[r, "b?"] & e_5 \\ &&&& e_6 \arrow[r, "a?"] & e_7 \end{tikzcd}$$

Synchronizé strict avec LIFO 2 éléments (pile)



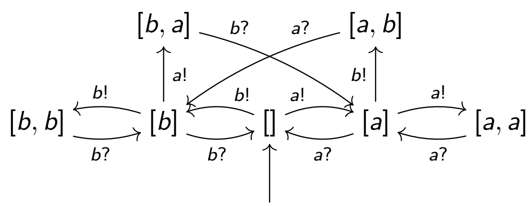
Donne

$$\longrightarrow (e_1, []) \xrightarrow{a!} (e_2, [a]) \xrightarrow{b!} (e_3, [a, b]) \xrightarrow{b?} (e_6, [a]) \xrightarrow{a?} (e_7, [])$$

Exemple – produit synchronisé strict ♪♪♪

$$\begin{tikzcd} {} \arrow[r] & e_1 \arrow[r, "a!"] & e_2 \arrow[r, "b!"] & e_3 \arrow[r, "a?"] \arrow[dr, "b?"'] & e_4 \arrow[r, "b?"] & e_5 \\ &&&& e_6 \arrow[r, "a?"] & e_7 \end{tikzcd}$$

Synchronizé strict avec FIFO 2 éléments (file)



Donne

$$\longrightarrow (e_1, []) \xrightarrow{a!} (e_2, [a]) \xrightarrow{b!} (e_3, [a, b]) \xrightarrow{a?} (e_4, [b]) \xrightarrow{b?} (e_5, [])$$

Composition : produit synchronisé ouvert ♪♪♪

Produit synchronisé ouvert

Le produit synchrone des ST étiquetés $\langle S_1, I_1, R_1, L_1\rangle$ et $\langle S_2, I_2, R_2, L_2\rangle$ est $\langle S, I, R, L\rangle$ où :

- $S \triangleq S_1 \times S_2$ (couple d'états)
- $I \triangleq I_1 \times I_2$
- $R \triangleq$
$$\left\{\begin{array}{l} ((s_1,s_2),(s_1',s_2')) \mid (s_1,s_1') \in R_1 \wedge (s_2,s_2') \in R_2 \\ \qquad\qquad\qquad \wedge\ Etiq((s_1,s_1')) = Etiq((s_2,s_2')) \\ ((s_1,s_2),(s_1',s_2)) \mid (s_1,s_1') \in R_1 \wedge Etiq((s_1,s_1')) \notin L_2 \\ ((s_1,s_2),(s_1,s_2')) \mid (s_2,s_2') \in R_2 \wedge Etiq((s_2,s_2')) \notin L_1 \end{array}\right\}$$
- $L = L_1 \cup L_2$

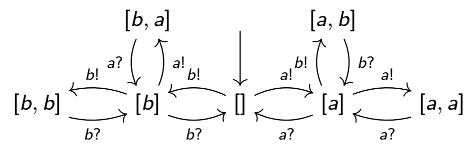
Synchronisation sur étiquette commune, bégaiement sur étiquette absente.

Définitions
Représentations
Propriétés générales
Composition

Exemple – produit synchronisé ouvert ♪♪♪

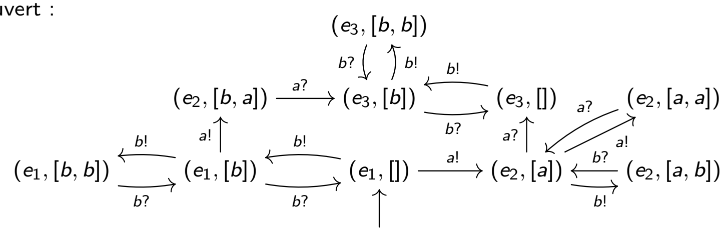
$$\longrightarrow e_1 \xrightarrow{a!} e_2 \xrightarrow{a?} e_3$$

Synchronizé avec LIFO 2 éléments (pile)



Donne

- strict : $\longrightarrow (e_1, []) \xrightarrow{a!} (e_2, [a]) \xrightarrow{a?} (e_3, [])$
- ouvert :



Définitions
Représentations
Propriétés générales
Composition

Bilan

Cette séance a présenté :

- la définition de système de transitions (états, transitions)
- la notion de trace et d'exécution
- la représentation explicite (en extension) ou symbolique (en intention)
- quelques propriétés génériques, dont le bégaiement
- diverses formes de composition de systèmes de transition

Deuxième partie

TLA$^+$– les actions

Objectifs

Décrire des systèmes de transition

- en intention
- de manière abstraite

Le formalisme de description doit être

- aussi naturel que possible (= proche d'un langage de programmation)
- parfaitement rigoureux (pas d'ambiguïté ou de sémantique approximative)
- complet (tout système de transition est descriptible)
- minimaliste dans les concepts (pour axiomatiser)
- extensible (pour décrire des concepts dérivés)

→ variables, ensembles et fonctions, actions de transition

TLA$^+$: Temporal Logic of Actions

TLA$^+$: Temporal Logic of Actions

- Un langage outillé pour modéliser les programmes et systèmes
- Particulièrement adapaté aux programmes et systèmes distribués / concurrents
- Basé sur des systèmes de transition
- Une toolbox embarquant un éditeur de texte, un outil de vérification par model checking (TLC) et un outil pour faire des preuves (TLAPS)
- `http://lamport.azurewebsites.net/tla/tla.html`

Plan

Structure d'une spécification

Un ≪ programme ≫ = une spécification de système de transition =

- des constantes
- des variables (états = valuation des variables)
- un ensemble d'états initiaux défini par un prédicat d'état
- des actions = prédicat de transition reliant deux états :
  - l'état courant, variables non primées
  - l'état d'arrivée, variables primées
- un prédicat de transition construit par disjonction des actions (≈ actions répétées infiniment)

Exemple

MODULE *exemple1*

EXTENDS *Naturals*

VARIABLE x

États initiaux

$Init \triangleq x \in 0\,..\,2$ équivalent à $x \in Nat \land 0 \leq x \land x < 3$

Actions

$Plus \triangleq x' = x + 1$

$Moins \triangleq x > 0 \land x' = x - 1$

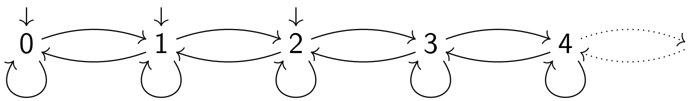
$Next \triangleq Plus \lor Moins$

$Spec \triangleq Init \land \Box[Next]_{\langle x \rangle}$

Exemple

Correspond au système de transitions :

$$V \triangleq x \in \mathbb{N}$$
$$I \triangleq 0 \leq x \leq 2$$
$$\begin{aligned} R \triangleq\ & \ x' = x + 1 \\ & \lor\ x > 0 \land x' = x - 1 \\ & \lor\ x' = x \end{aligned}$$



Constantes

- Constantes explicites : 0, 1, TRUE, FALSE, "toto"
- Constantes nommées : CONSTANT N
  généralement accompagnées de propriétés :
  ASSUME $N \in Nat \land N \geq 2$

Expressions autorisées

Tout ce qui est axiomatisable :

- expressions logiques : $\neg$, $\wedge$, $\vee$, $\forall x \in S : p(x)$, $\exists x \in S : p(x) \ldots$
- expressions arithmétiques : $+$, $-$, $>$ ...
- expressions ensemblistes : $\in$, $\cup$, $\cap$, $\subset$, $\{e_1, e_2, \ldots, e_n\}$, $n..m$, $\{x \in S : p(x)\}$, $\{f(x) : x \in S\}$, UNION S, SUBSET S
- IF $pred$ THEN e_1 ELSE e_2
- fonctions de X dans Y
- tuples, séquences, ...

Opérateurs ensemblistes

$\{e_1, \ldots, e_n\}$	ensemble en extension
$n..m$	$\{i \in Nat : n \leq i \leq m\}$
$\{x \in S : p(x)\}$	l'ensemble des éléments de S vérifiant la propriété p
	$\{n \in 1..10 : n\%2 = 0\} = \{2, 4, 6, 8, 10\}$
	$\{n \in Nat : n\%2 = 1\} =$ les nombres impairs
$\{f(x) : x \in S\}$	l'ensemble des valeurs de l'operateur f en S
	$\{2 * n : n \in 1..5\} = \{2, 4, 6, 8, 10\}$
	$\{2 * n + 1 : n \in Nat\} =$ les nombres impairs
UNION S	l'union des éléments de S
	UNION $\{\{1, 2\}, \{2, 3\}, \{3, 4\}\} = \{1, 2, 3, 4\}$
SUBSET S	l'ensemble des sous-ensembles de S
	SUBSET $\{1, 2\} = \{\{\}, \{1\}, \{2\}, \{1, 2\}\}$

Plan

Actions

Action

Action = prédicat de transition = expression booléenne contenant des constantes, des variables et des variables primées.

Une action n'est pas une affectation.

$$
\begin{aligned}
& x' = x + 1 \\
\equiv\ & x' - x = 1 \\
\equiv\ & x = x' - 1 \\
\equiv\ & (x > 1 \wedge x'/x = 1 \wedge x'\%x = 1) \vee (1 = x \wedge 2 = x') \\
& \quad \vee (x = 0 \wedge x' \in \{y \in Nat : y + 1 = 2 * y\})
\end{aligned}
$$

Autres exemples d'actions :

- $x' > x$ ou $x' \in \{x + 1, x + 2, x + 3\}$ (non déterministe)
- $x' \in \{y \in \mathbb{N} : \exists z \in \mathbb{N} : z * y = x \wedge z\%2 = 0\}$ (non évaluable)
- $x' = y \wedge y' = x$ (plusieurs variables)

Action gardée

Action gardée

Action constituée d'une conjonction :

1. un prédicat d'état portant uniquement sur l'état de départ
2. un prédicat de transition déterministe $var' = \ldots$
   ou un prédicat de transition non déterministe $var' \in \ldots$

Se rapproche d'une instruction exécutable.

$$\begin{array}{rl} & x < 10 \land x' = x + 1 \\ \text{plutôt que} & x' = x + 1 \land x' < 11 \\ \text{ou} & x' - x = 1 \land x' < 11 \end{array}$$

Bégaiement

Bégaiement

$[\mathcal{A}]_f \triangleq \mathcal{A} \lor f' = f$, où f est un tuple de variables.

exemple : $$[x' = x+1]_{\langle x,y\rangle} = (x' = x+1 \lor (\langle x,y\rangle' = \langle x,y\rangle)) = (x' = x+1 \lor (x' = x \land y' = y))$$

Non bégaiement

$\langle \mathcal{A}\rangle_f \triangleq \mathcal{A} \land f' \neq f$

Variables non contraintes

$$\begin{array}{rcl} (x' = x+1) & = & (x' = x+1 \land y' = \text{n'importe quoi}) \\ & \neq & (x' = x+1 \land y' = y) \end{array}$$

UNCHANGED

$\text{UNCHANGED } e \triangleq e' = e$


```
──────────── MODULE AlternatingBit ────────────
EXTENDS Naturals
CONSTANT Data
VARIABLES val, ready, ack
───────────────────────────────────────────────
Init ≜ ∧ val ∈ Data
       ∧ ready ∈ {0, 1}
       ∧ ack = ready

Send ≜ ∧ ready = ack
       ∧ val' ∈ Data
       ∧ ready' = 1 − ready
       ∧ UNCHANGED ack

Receive ≜ ∧ ready ≠ ack
          ∧ ack' = 1 − ack
          ∧ UNCHANGED ⟨val, ready⟩

Next ≜ Send ∨ Receive
Spec ≜ Init ∧ □[Next]_⟨val, ready, ack⟩
───────────────────────────────────────────────
```


Mise en pratique : factorielle

Écrire la spécification d'un programme qui définit la factorielle d'un nombre N, c'est-à-dire écrire une spécification telle qu'une variable contiendra, en un point déterminé d'une exécution, la valeur de $N!$ et ne changera plus ensuite.

- En une transition (!)
- En N transitions déterministes, par multiplications successives, par ordre croissant ou décroissant
- En $\lceil \frac{N}{2} \rceil$ à N transitions non déterministes, en pouvant faire deux multiplications en une transition
- En N transitions non déterministes, sans ordre particulier des multiplications
- En $1..N$ transitions non déterministes, en pouvant faire plusieurs multiplications en une transition

Plan

Fonctions

Fonction au sens "mapping", correspondance.

- $[X \to Y]$ = ensemble des fonctions de X dans Y.
- f fonction de X dans Y : $f \in [X \to Y]$
- $f[x] \triangleq$ la valeur de f en x.

Une fonction est une valeur.
Une variable contenant une fonction peut changer de valeur $\Rightarrow$ la "fonction change".

Définition

Définition d'un symbole

$f[x \in Nat] \triangleq$ expression utilisant x

Exemple : $Inc[x \in Nat] \triangleq x + 1$

Définition d'une valeur

$[x \in S \mapsto expr]$

Exemple : $[x \in 1..4 \mapsto 2 * x]$

Tableaux

Tableau : fonction $t \in [X \to Y]$ où X est un intervalle d'entiers.

Domaine/Codomaine

Domain

$\text{DOMAIN } f$ = domaine de définition de f

Codomaine (range)

$Codomain(f) \triangleq \{f[x] : x \in \text{DOMAIN } f\}$

EXCEPT

Une variable contenant une fonction peut changer de valeur :

MODULE m

VARIABLE a

$Init \triangleq a = [i \in 1..3 \mapsto i+1]$

$Act1 \triangleq \land a[1] = 2$
$\quad\quad\quad \land a' = [i \in 1..6 \mapsto i * 2]$

$Act2 \triangleq \land a[2] = 4$
$\quad\quad\quad \land a' = [i \in 1..6 \mapsto \text{IF } i = 2 \text{ THEN } 8 \text{ ELSE } a[i]]$

EXCEPT

$[a \text{ EXCEPT } ![i] = v]$ équivalent à
$[j \in \text{DOMAIN } a \mapsto \text{IF } j = i \text{ THEN } v \text{ ELSE } a[j]]$

$(a' = [a \text{ EXCEPT } ![2] = 8]) \not\equiv (a[2]' = 8)$

Fonction $\neq$ opérateur

$IncF[x \in Nat] \triangleq x+1$
$IncO(x) \triangleq x+1$

- $IncF$ est une définition de fonction au sens mathématique
  - Équivalent à $IncF \triangleq [x \in Nat \mapsto x+1]$
  - Son domaine est un ensemble : DOMAIN $IncF$
  - Son co-domaine est un ens. : $\{IncF[x] : x \in \text{DOMAIN } IncF\}$
  - $IncF \in [X \to Y]$ a du sens
- $IncO$ est la définition d'un opérateur
  - Factorisation d'écriture : similaire à une macro dont on peut substituer le texte
  - N'a pas de domaine ou de co-domaine
  - $IncO \in [X \to Y]$ n'a pas de sens

Enregistrements

Enregistrement

Un enregistrement (record) est une fonction de $[X \to Y]$ où X est un ensemble de chaînes.

Écriture simplifiée :

$[\text{“}toto\text{”} \mapsto 1, \text{“}titi\text{”} \mapsto 2] \quad = \quad [toto \mapsto 1, titi \mapsto 2]$
$rec[\text{“}toto\text{”}] \quad = \quad rec.toto$

Définition récursive

Lors de la définition de symbole (fonction ou opérateur), il est possible de donner une définition récursive :

- Fonction :
  $fact[n \in Nat] \triangleq \text{IF } n = 0 \text{ THEN } 1 \text{ ELSE } n * fact[n-1]$
- Opérateur :
  $RECURSIVE\ fact(\_)$
  $fact(n) \triangleq \text{IF } n = 0 \text{ THEN } 1 \text{ ELSE } n * fact(n-1)$

En théorie, il faudrait démontrer la validité de ces définitions (terminaison dans tous les cas).

Tuple

n-tuple

Notation : $\langle a, b, c \rangle$.
Un n-tuple est une fonction de domaine $= \{1,\ldots,n\}$:
$\langle a, b, c \rangle[3] = c$

Pratique pour représenter des relations :
$\{\langle x, y \rangle \in X \times Y : R(x, y)\}$.
Exemple : $\{\langle a, b \rangle \in Nat \times Nat : a = 2 * b\}$.

Séquences

Séquences

$Seq(T) \triangleq \text{UNION } \{[1\,..\,n \rightarrow T] : n \in Nat\}$
$\triangleq$ ensemble des séquences finies contenant des T.
Opérateurs $Len(s)$, $s \circ t$ (concaténation), $Append(s, e)$, $Head(s)$, $Tail(s)$.

Exemple de définition des opérateurs :

$$s \circ t \triangleq [i \in 1..(Len(s) + Len(t)) \mapsto \text{IF } i \leq Len(s) \text{ THEN } s[i] \text{ ELSE } t[i - Len(s)]]$$

$$Append(s, e) \triangleq s \circ \langle e \rangle$$

Plan

Définition de symbole local

LET

Expression : $\text{LET } v \triangleq e \text{ IN } f$

Équivalent à l'expression f où toutes les occurrences du symbole v sont remplacées par e.
Exemple : $\text{LET } i \triangleq g(x) \text{ IN } f(i)$
$\equiv f(g(x))$

$$pythagore(x, y, z) \triangleq \text{LET } carre(n) \triangleq n * n \text{ IN } carre(x) + carre(y) = carre(z)$$

Choix déterministe

Opérateur de choix

$\text{CHOOSE } x \in S : p \triangleq$ choix arbitraire *déterministe* d'un élément dans l'ensemble S et qui vérifie le prédicat p.

$max[S \in \text{SUBSET } Nat] \triangleq \text{CHOOSE } m \in S : (\forall p \in S : m \geq p)$

Choix déterministe

$\text{CHOOSE } x \in S : p = \text{CHOOSE } x \in S : p$ (aïe)
Pour un ensemble S et une propriété p, l'élément choisi est toujours le même, dans toutes les exécutions et tout au long de celles-ci. Ce n'est pas un sélecteur aléatoire qui donne un élément distinct à chaque appel.

Choix déterministe - 2

- La spécification

$$(x = \text{CHOOSE } n : n \in Nat) \wedge \Box[x' = \text{CHOOSE } n : n \in Nat]_{\langle x \rangle}$$

a une unique exécution : $x = c \to x = c \to \ldots$ où c est un nombre entier indéterminé (spécifié par le choose).

- La spécification

$$(x \in Nat) \wedge \Box[x' \in Nat]_{\langle x \rangle}$$

a une infinité d'exécutions, dont certaines où x est différent dans chaque état, d'autres où x est constant, d'autres où x cycle...

Troisième partie

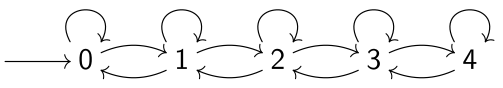
L'équité dans les systèmes de transitions

Plan

Pourquoi de l'équité ?

MODULE *oscillant*
CONSTANT N
VARIABLE i

$$\begin{aligned}
Init &\triangleq i = 0\\
Next &\triangleq \lor\ i > 0 \land i' = i - 1\\
&\ \lor\ i < N \land i' = i + 1\\
Spec &\triangleq Init \land \Box[Next]_i
\end{aligned}$$



On pourrait vouloir éliminer les exécutions :

- 0^ω
- $(0 \to 1)^\omega$
- $\ldots \to n^\omega$
- qui ne passent pas infiniment souvent par l'état 2
- qui ne contiennent pas une infinité d'incrémentation

Contraintes d'équité / *fairness* ♪♪♪

Les contraintes d'équité spécifient que certains états (resp. certaines transitions) doivent être visités (resp. exécutées) infiniment souvent dans toute exécution du programme.

D'une façon générale, les contraintes d'équité servent à contraindre un programme ou son environnement à être vivace, sans entrer dans les détails concernant la réalisation pratique de ces contraintes.

Les contraintes d'équité réduisent l'ensemble des exécutions légales, en éliminant les exécutions qui ne respectent pas les contraintes d'équité.

États récurrents

Ensemble récurrent d'états

Soit $\mathcal{S} = \langle S, I, R \rangle$ un système de transitions et $\sigma = \langle s_0 \to \ldots \rangle$ une exécution.
Un ensemble d'états P est récurrent dans σ si :

- cas σ infinie : $\forall i \in \mathbb{N} : \exists j \geq i : s_j \in P$
  (P apparaît une infinité de fois dans σ).
- cas σ finie : l'état final de σ est dans P.

$Inf_S(P, \sigma) \triangleq P$ est un ensemble récurrent d'états dans σ.

Note : on dit aussi *infiniment souvent présent* dans σ.

Transitions récurrentes

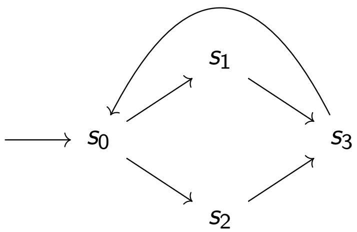
Ensemble récurrent de transitions

Soit $\mathcal{S} = \langle S, I, R \rangle$ un système de transitions et $\sigma = \langle s_0 \to \ldots \rangle$ une exécution.
Un ensemble de transitions Q est récurrent dans σ si :

- cas σ infinie : $\forall i \in \mathbb{N} : \exists j \geq i : s_j \to s_{j+1} \in Q$
  (des transitions de Q apparaissent une infinité de fois dans σ).
- cas σ finie : la transition finale de σ est dans Q
  ($\sigma = \langle s_0 \to \ldots \to s \to s' \rangle \wedge s \to s' \in Q$).

$Inf_T(Q, \sigma) \triangleq Q$ est un ensemble récurrent de transitions dans σ.

Exemple - états récurrents



s_1	récurrent dans	$\langle (s_0 \to s_1 \to s_3)^\omega \rangle$
s_1	récurrent dans	$\langle (s_0 \to s_1 \to s_3 \to s_0 \to s_2 \to s_3)^\omega \rangle$
s_1	pas récurrent dans	$\langle (s_0 \to s_1 \to s_3)^* \to (s_0 \to s_2 \to s_3)^\omega \rangle$
$s_1 \to s_3$	récurrente dans	$\langle (s_0 \to s_1 \to s_3 \to s_0 \to s_2 \to s_3)^\omega \rangle$
$s_1 \to s_3$	pas récurrente dans	$\langle (s_0 \to s_1 \to s_3)^* \to (s_0 \to s_2 \to s_3)^\omega \rangle$

Plan

Équité simple sur les états ♪♪♪

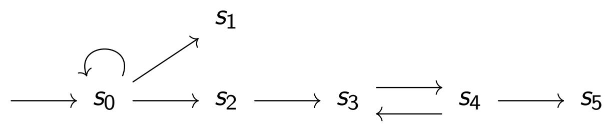
Équité simple

Soit un système de transition $\langle S, I, R\rangle$.
On se donne $F \subseteq S$ un ensemble d'états équitables.
Alors toute exécution σ doit être telle que $Inf_S(F, \sigma)$.

F est récurrent dans σ, i.e. σ contient une infinité d'états dans F (cas σ infini), ou le dernier état de σ est dans F (cas σ fini).

Remarque : l'ensemble F est récurrent, pas nécessairement chaque élément de F. Pour $\mathcal{S} \triangleq i = 0 \wedge \square((i' = i + 1) \vee (i' = i))$, si on se donne $F \triangleq \{i\%2 = 0\}$, les exécutions $0 \to 1 \to 2^\omega$ et $0 \to 1 \to 2 \to 3 \to 4 \ldots$ sont valides.

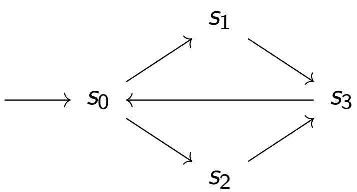
Exemple - équité simple ♪♪♪



$$Exec(S) = \langle s_0{}^\omega\rangle, \langle s_0{}^+ \to s_2 \to (s_3 \to s_4)^\omega\rangle, \langle s_0{}^+ \to s_1\rangle, \langle s_0{}^+ \to s_2 \to (s_3 \to s_4)^+ \to s_5\rangle$$

Équité simple	Exécutions
$\{s_0\}$	$\langle s_0{}^\omega\rangle$
$\{s_1, s_4\}$	$\langle s_0{}^+ \to s_2 \to (s_3 \to s_4)^\omega\rangle, \langle s_0{}^+ \to s_1\rangle$
$\{s_1, s_5\}$	$\langle s_0{}^+ \to s_1\rangle, \langle s_0{}^+ \to s_2 \to (s_3 \to s_4)^+ \to s_5\rangle$

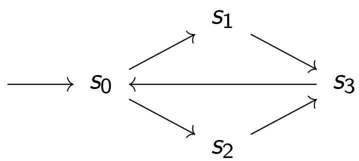
Exemple - équité simple ♪♪♪



On fixe : équité simple sur $\{s_1\}$.

légale	$\langle(s_0 \to s_1 \to s_3)^\omega\rangle$
légale	$\langle(s_0 \to s_1 \to s_3 \to s_0 \to s_2 \to s_3)^\omega\rangle$
illégale	$\langle(s_0 \to s_1 \to s_3)^\star \to (s_0 \to s_2 \to s_3)^\omega\rangle$
légale	$\langle(s_0 \to s_1 \to s_3 \to (s_0 \to s_2 \to s_3)^\star)^\omega\rangle$

Exemple - équité simple



$$Exec(S) =$$

Équité simple	
$\{s_1\}$	
$\{s_1, s_2\}$	

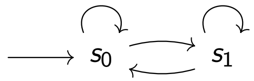
Équité multiple sur les états

Équité multiple

Soit un système de transition $\langle S, I, R\rangle$.
On se donne un ensemble dénombrable, indexable par un ensemble d'entiers $J = \{0, 1, 2, \ldots\}$, d'ensembles équitables $\{F_i\}_{i \in J}$.
Toute exécution σ doit être telle que $\forall i \in J : Inf_S(F_i, \sigma)$.

Exécutions vérifiant l'équité multiple = intersection des exécutions vérifiant l'équité simple sur chacun des F_i.
$\Rightarrow$ l'équité simple est un cas particulier de l'équité multiple.

Exemple - équité multiple



$Exec(S) \quad =$

Équité simple/multiple	
$\{s_0\}$ $\{s_0, s_1\}$ $\{s_0\}\{s_1\}$	

Équivalence équité multiple finie $\leftrightarrow$ simple

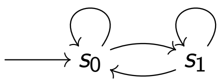
Cas simple : J est fini. $|J|$ est la cardinalité de J.

Le système $\langle S', I', R'\rangle$ à équité simple suivant est équivalent (égalité des exécutions projetées sur S) :

- $S' = S \times J$
- $I' = I \times \{0\}$
- $R' = \quad \{(\langle s, j\rangle, \langle s', j + 1 \bmod |J|\rangle) \mid (s, s') \in R \wedge s \in F_j\}$
  $\cup \{(\langle s, j\rangle, \langle s', j\rangle) \mid (s, s') \in R \wedge s \notin F_j\}$
- Équité simple $F' = F_0 \times \{0\}$

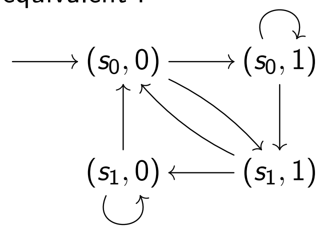
Le premier ensemble de R' est pour le cas où on visite un état de F_j et on cherche donc à visiter l'ensemble suivant ; le deuxième ensemble est pour le cas où on n'est pas en train de visiter un état de F_j, que l'on continue à attendre.

Exemple équité multiple



avec équité multiple : $F_0 = \{s_0\}, F_1 = \{s_1\}$

ST en équité simple équivalent :



avec équité simple sur $\{(s_0, 0)\}$

Équivalence équité multiple $\leftrightarrow$ simple ♪♪♪

Cas général (J potentiellement infini).

Le système $\langle S', I', R' \rangle$ à équité simple suivant est équivalent :

- $S' = S \times J \times J$
- $I' = I \times \{0\} \times \{0\}$
- $R' =$

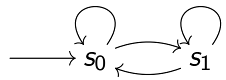
$$\begin{aligned} & \{(\langle s,i,i\rangle, \langle s', i \oplus 1, 0\rangle) \mid (s,s') \in R \wedge s \in F_i\} \\ \cup\ & \{(\langle s,i,j\rangle, \langle s', i, j+1\rangle) \mid j < i \wedge (s,s') \in R \wedge s \in F_j\} \\ \cup\ & \{(\langle s,i,j\rangle, \langle s', i, j\rangle) \mid (s,s') \in R \wedge s \notin F_j\} \end{aligned}$$

- Équité simple $F' = F_0 \times J \times \{0\}$

avec : $i \oplus 1 \triangleq \begin{cases} i+1 & \text{si } J \text{ est infini} \\ i+1 \bmod |J| & \text{sinon} \end{cases}$

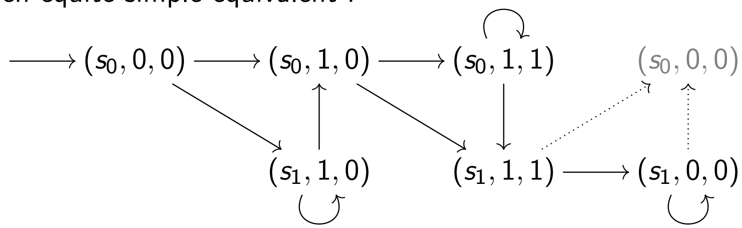
Dans une exécution équitable, les compteurs i, j forment un triangle :
$\langle (0,0) \to (1,0) \to (1,1) \to (2,0) \to (2,1) \to (2,2) \to (3,0) \to \ldots \rangle$

Exemple équité multiple



avec équité multiple : $F_0 = \{s_0\}, F_1 = \{s_1\}$

ST en équité simple équivalent :



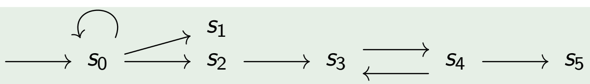
avec équité simple sur $\{(s_0, 0, 0), (s_0, 1, 0)\}$

Équité conditionnelle sur les états ♪♪♪

Équité conditionnelle

Soit un système de transition $\langle S, I, R \rangle$.
On se donne deux ensembles F et G.
Toute exécution σ doit être telle que $Inf_S(F,\sigma) \Rightarrow Inf_S(G,\sigma)$.

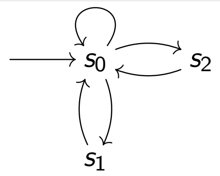
Si F est récurrent dans σ, alors G doit être récurrent dans σ.



$$Exec(S) = \langle {s_0}^\omega \rangle, \langle {s_0}^+ \to s_2 \to (s_3 \to s_4)^\omega \rangle, \langle {s_0}^+ \to s_1 \rangle, \langle {s_0}^+ \to s_2 \to (s_3 \to s_4)^+ \to s_5 \rangle$$

Équité cond.	Exécutions
$\{s_0\} \Rightarrow \{s_5\}$	$\langle {s_0}^+ \to s_2 \to (s_3 \to s_4)^\omega \rangle$, $\langle {s_0}^+ \to s_1 \rangle$, $\langle {s_0}^+ \to s_2 \to (s_3 \to s_4)^+ \to s_5 \rangle$
$\{s_3\} \Rightarrow \{s_4\}$	$\langle {s_0}^\omega \rangle$, $\langle {s_0}^+ \to s_2 \to (s_3 \to s_4)^\omega \rangle$, $\langle {s_0}^+ \to s_1 \rangle$, $\langle {s_0}^+ \to s_2 \to (s_3 \to s_4)^+ \to s_5 \rangle$

Exemple - équité conditionnelle



$Exec(S) =$

Équité cond.	
$\{s_1\} \Rightarrow \{s_2\}$	

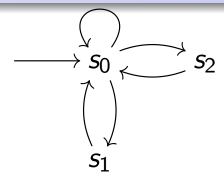
Équivalence équité conditionnelle ↔ simple

Soit un système $\langle S, I, R\rangle$ avec équité conditionnelle $F \Rightarrow G$.
Le système $\langle S', I', R'\rangle$ à équité simple suivant est équivalent :

- $S' = (S \times \{0\}) \cup ((S \setminus F) \times \{1\})$
- $I' = I \times \{0\}$
- $R' = \begin{array}{l} \{(\langle s,0\rangle, \langle s',0\rangle) \mid (s,s') \in R\} \\ \cup\ \{(\langle s,0\rangle, \langle s',1\rangle) \mid (s,s') \in R \land s' \in (S \setminus F)\} \\ \cup\ \{(\langle s,1\rangle, \langle s',1\rangle) \mid (s,s') \in R \land s, s' \in (S \setminus F)\} \end{array}$
- Équité simple $F' = (G \times \{0\}) \cup ((S \setminus F) \times \{1\})$

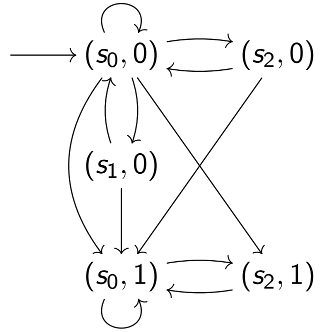
Les états $\langle s, 0\rangle$, identiques au système d'origine, correspondent aux exécutions où G doit être infiniment souvent visité.
Les états $\langle s, 1\rangle$, restreints aux états non dans F, correspondent aux exécutions où F ne doit plus jamais être visité.

Exemple équité conditionnelle



avec équité conditionnelle : $F = \{s_1\}, G = \{s_2\}$

ST en équité simple équivalent :



avec équité simple sur $\{(s_2, 0), (s_0, 1), (s_2, 1)\}$

Plan

Équité sur les transitions

L'équité sur les transitions est plus précise que l'équité sur les états.
Informellement, une exécution infinie est non équitable vis-à-vis d'une transition si :

- la transition n'apparaît qu'un nombre fini de fois,
- et la transition est continûment faisable (équité faible) ou infiniment souvent faisable (équité forte).

Les définitions suivantes sont correctes aussi bien pour les exécutions infinies que pour les exécutions finies maximales. Pour autant, les explications sont plus faciles sur les exécutions infinies. Le bégaiement est présent par défaut dans TLA$^+$et dans la majorité des méthodes outillées s'appuyant sur les systèmes de transition, ce qui justifie cette simplification.

Équité faible sur les transitions

Équité faible

Soit un ST $\langle S, I, R\rangle$ et $F \subseteq R$ un sous-ensemble des transitions.
F est faiblement équitable ssi dans toute exécution σ :

$$Inf_S(S \setminus dom(F), \sigma) \vee Inf_T(F, \sigma)$$

(l'ensemble d'états $S \setminus dom(F)$ est récurrent,
ou l'ensemble de transitions F est récurrent)

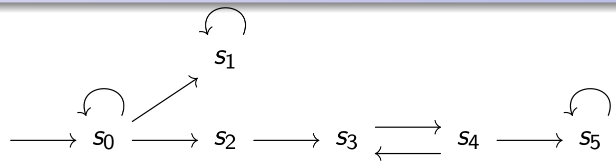
Ou, de manière équivalente :

$$\neg Inf_S(S \setminus dom(F), \sigma) \Rightarrow Inf_T(F, \sigma)$$

(si l'ensemble d'états $S \setminus dom(F)$ n'est pas récurrent,
alors l'ensemble de transitions F est récurrent)

L'équité faible exprime que l'on n'a pas le droit de rester indéfiniment dans un ensemble spécifié d'états alors qu'il existe toujours une transition en équité faible qui est exécutable.

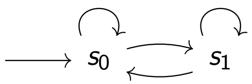
Exemple - équité faible



$$\begin{aligned} Exec(S) \;=\; & \langle s_0{}^{\omega}\rangle, \langle s_0{}^{+} \to s_2 \to (s_3 \to s_4)^{\omega}\rangle, \\ & \langle s_0{}^{+} \to s_1{}^{\omega}\rangle, \langle s_0{}^{+} \to s_2 \to (s_3 \to s_4)^{+} \to s_5{}^{\omega}\rangle \end{aligned}$$

Équité faible	Exécutions
$\{(s_0, s_1)\}$	$\langle s_0{}^{+} \to s_2 \to (s_3 \to s_4)^{\omega}\rangle$, $\langle s_0{}^{+} \to s_1{}^{\omega}\rangle, \langle s_0{}^{+} \to s_2 \to (s_3 \to s_4)^{+} \to s_5{}^{\omega}\rangle$
$\{(s_0, s_1), (s_0, s_0)\}$	*toutes*
$\{(s_4, s_5)\}$	*toutes*

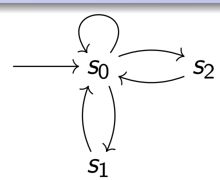
Exemple - équité faible



$$\begin{aligned} Exec(S) \;=\; & \langle (s_0{}^{+} \to s_1{}^{+})^{\omega}\rangle, \\ & \langle (s_0{}^{+} \to s_1{}^{+})^{*} \to s_0{}^{\omega}\rangle, \\ & \langle (s_0{}^{+} \to s_1{}^{+})^{*} \to s_0{}^{+} \to s_1{}^{\omega}\rangle \end{aligned}$$

Équité faible	Exécutions
$\{(s_0, s_1)\}$	$\langle (s_0{}^{+} \to s_1{}^{+})^{\omega}\rangle$, $\langle (s_0{}^{+} \to s_1{}^{+})^{*} \to s_0{}^{+} \to s_1{}^{\omega}\rangle$
$\{(s_0, s_0)\}$	*toutes*
$\{(s_0, s_0), (s_0, s_1)\}$	*toutes*

Exemple - équité faible

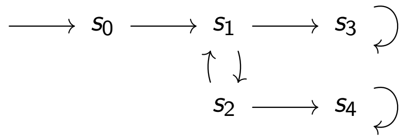


On note $T \triangleq s_0{}^{*} \to (s_0 \to s_1)^{*} \to (s_0 \to s_2)^{*} \setminus \langle\rangle$.
(le $\setminus\langle\rangle$ garantit que T ne contient pas la séquence vide)

$$Exec(S) = \langle T^{\omega}\rangle$$

Équité faible	Exécutions
$\{(s_0, s_2)\}$	$\langle T^{*} \to (s_0{}^{*} \to (s_0 \to s_1)^{*} \to s_0{}^{*} \to (s_0 \to s_2)^{+})^{\omega}\rangle$, $\langle T^{*} \to (s_0{}^{*} \to (s_0 \to s_1)^{+})^{\omega}\rangle$

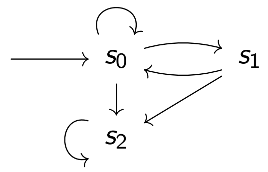
Exemple - équité faible



$Exec(S) \quad =$

Équité faible	
$\{(s_2, s_4)\}$ $\{(s_2, s_4), (s_1, s_3)\}$	

Exemple - équité faible



$Exec(S) \quad =$

Équité faible	
$\{(s_0, s_1)\}$ $\{(s_1, s_2)\}$ $\{(s_0, s_2), (s_1, s_2)\}$	

Équité faible → équité simple sur les états ♪♪♪

Soit un système $\langle S, I, R\rangle$ avec équité faible sur F.
Le système $\langle S', I', R'\rangle$ à équité simple suivant est équivalent :

- $S' = S \times \{0, 1\}$
- $I' = I \times \{0\}$
- $R' = \{(\langle s, \_\rangle, \langle s', 1\rangle) \mid (s, s') \in R \cap F\}$
  $\cup \{(\langle s, \_\rangle, \langle s', 0\rangle) \mid (s, s') \in R \setminus F\}$
- Équité simple $F' = S \setminus dom(F) \times \{0, 1\} \;\cup\; S \times \{1\}$

Les états $\langle s, 1\rangle$ correspondent aux états où l'on vient d'exécuter une transition de F, les états $\langle s, 0\rangle$ correspondent aux états où l'on vient d'exécuter une transition qui n'est pas dans F.

Équité forte sur les transitions ♪♪♪

Équité forte

Soit un ST $\langle S, I, R\rangle$ et $F \subseteq R$ un sous-ensemble des transitions.
F est fortement équitable ssi dans toute exécution σ :

$$\neg Inf_S(dom(F), \sigma) \vee Inf_T(F, \sigma)$$

l'ensemble d'états $dom(F)$ n'est pas récurrent,
ou l'ensemble de transitions F est récurrent.

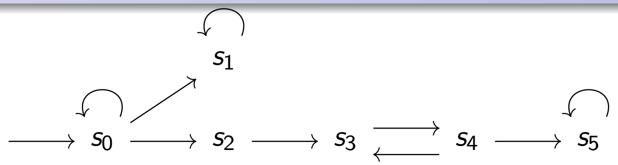
Ou, de manière équivalente :

$$Inf_S(dom(F), \sigma) \Rightarrow Inf_T(F, \sigma)$$

si l'ensemble d'états $dom(F)$ est récurrent,
alors l'ensemble de transitions F est récurrent.

L'équité forte exprime que si l'on passe infiniment souvent dans un ensemble d'états où des transitions de r sont exécutables, alors une transition de r finit par être exécutée.

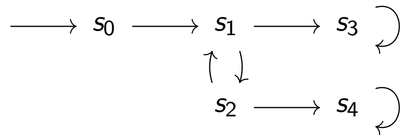
Exemple - équité forte



$$
\begin{aligned}
Exec(S) \;=\;& \langle {s_0}^{\omega}\rangle, \langle {s_0}^{+} \to s_2 \to (s_3 \to s_4)^{\omega}\rangle,\\
& \langle {s_0}^{+} \to {s_1}^{\omega}\rangle, \langle {s_0}^{+} \to s_2 \to (s_3 \to s_4)^{+} \to {s_5}^{\omega}\rangle
\end{aligned}
$$

Équité forte	Exécutions
$\{(s_0, s_1)\}$	$\langle {s_0}^{+} \to s_2 \to (s_3 \to s_4)^{\omega}\rangle$, $\langle {s_0}^{+} \to {s_1}^{\omega}\rangle, \langle {s_0}^{+} \to s_2 \to (s_3 \to s_4)^{+} \to {s_5}^{\omega}\rangle$
$\{(s_4, s_5)\}$	$\langle {s_0}^{\omega}\rangle, \langle {s_0}^{+} \to {s_1}^{\omega}\rangle, \langle {s_0}^{+} \to s_2 \to (s_3 \to s_4)^{+} \to {s_5}^{\omega}\rangle$
$\{(s_3, s_4), (s_4, s_5)\}$	*toutes*

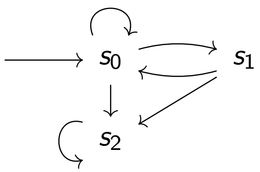
Exemple - équité forte



$$Exec(S) \;=$$

Équité forte	
$\{(s_2, s_4)\}$	

Exemple - équité forte



$$Exec(S) \;=$$

Équité forte	
$\{(s_0, s_1)\}$ $\{(s_1, s_2)\}$ $\{(s_0, s_1), (s_1, s_2)\}$	

Équité forte $\to$ équité conditionnelle sur les états

Soit un système $\langle S, I, R\rangle$ avec équité forte sur F.
Le système $\langle S', I', R'\rangle$ à équité conditionnelle suivant est équivalent :

- $S' = S \times \{0, 1\}$
- $I' = I \times \{0\}$
- $R' = \;\; \{(\langle s, \_\rangle, \langle s', 1\rangle) \mid (s, s') \in R \cap F\} \;\cup\; \{(\langle s, \_\rangle, \langle s', 0\rangle) \mid (s, s') \in R \setminus F\}$
- Équité conditionnelle $\;\; F' = dom(F) \times \{0, 1\}$, $\;\; G' = S \times \{1\}$

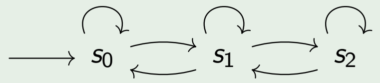
Les états $\langle s, 1\rangle$ correspondent aux états où l'on vient d'exécuter une transition de F, les états $\langle s, 0\rangle$ correspondent aux états où l'on vient d'exécuter une transition qui n'est pas dans F.

Combinaisons d'équités faibles/fortes

En pratique, on se donne

- plusieurs ensembles de transitions en équité faible,
- plusieurs ensembles de transitions en équité forte.

Le système doit respecter toutes ces contraintes (la conjonction).



Équité faible sur $\{(s_0, s_1)\}$ (interdit le bégaiement à l'infini)
Équité faible sur $\{(s_1, s_2)\}$ (idem)
Équité faible sur $\{(s_2, s_1)\}$ (idem)
Équité forte sur $\{(s_1, s_2)\}$ (interdit de ne jamais aller en s_2)

Ici, équivalent à équité multiple sur $\{\{s_1\}, \{s_2\}\}$: toute exécution où s_1 et s_2 apparaissent infiniment souvent.

Équité sur les étiquettes

Dans le cas d'un système de transitions étiqueté, on peut également définir l'équité (faible ou forte) sur un ensemble d'étiquettes $F \subseteq L$. Cela revient à l'équité sur les transitions $Etiq^{-1}(F)$.

Conclusion

- L'équité contraint des états / des transitions à être visité(e)s infiniment souvent.
- Les contraintes d'équité éliminent les exécutions non équitables, jugées sans intérêt.
- On utilise plutôt l'équité sur les transitions qui traduit que, si une action est toujours / infiniment souvent faisable, elle aura lieu : le système n'est pas injuste vis-à-vis de ces actions.

Quatrième partie

LTL – logique temporelle linéaire

Plan

Logiques temporelles ♪♪♪

Objectif

Exprimer des propriétés portant sur les exécutions des systèmes.

Spécification non opérationnelle : pas de relation de transition explicite, pas de notion d'états initiaux.
Une logique est définie par :

- une syntaxe : opérateurs de logique classique plus des opérateurs temporels pour parler du futur et du passé.
- une sémantique : domaine des objets (appelés modèles) sur lesquels on va tester la validité des formules, plus l'interprétation des opérateurs.

Plan

Linear Temporal Logic

Modèles

Une formule LTL se rapporte toujours à une trace donnée σ d'un système : les traces constituent les modèles de cette logique.

Note : plutôt que d'*état*, on parle souvent d'*instant* pour désigner les éléments d'une trace.

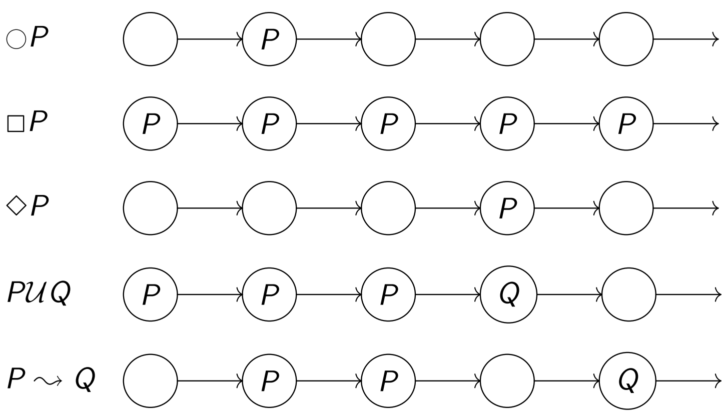
Rappel : pour un ST $\langle S, I, R\rangle$, une trace est une séquence $\sigma \in S^\star \cup S^\omega$, tel que pour tout s_i, s_{i+1} consécutifs, $(s_i, s_{i+1}) \in R$.

Syntaxe de la LTL

formule	nom	interprétation
s		le premier état de la trace est s
$\neg P$		
$P \vee Q$		
$P \wedge Q$		
$\bigcirc P$	*next*	P est vrai à l'instant suivant
$\Box P$	*always*	P est toujours vrai i.e. à tout instant à partir de l'instant courant
$\Diamond P$	*eventually*	P sera vrai (dans le futur)
$P \mathcal{U} Q$	*until*	Q sera vrai, et en attendant P reste vrai
$P \leadsto Q$	*leadsto*	quand P est vrai, alors Q est vrai plus tard

Dans les approches symboliques, l'opérateur $\bigcirc$ représentant l'instant suivant peut être remplacé par des variables primées qui représentent la valeur des variables du système dans l'état suivant.

Intuition sémantique



Opérateurs minimaux

Les opérateurs minimaux sont $\bigcirc P$ et $P \mathcal{U} Q$:

- $\Diamond P \triangleq True\, \mathcal{U} P$
- $\Box P \triangleq \neg\Diamond\neg P$
- $P \leadsto Q \triangleq \Box(P \Rightarrow \Diamond Q)$

Syntaxe alternative

Syntaxe alternative

On trouve fréquemment une autre syntaxe :

- $\Box \leftrightarrow$ G (*globally*)
- $\Diamond \leftrightarrow$ F (*finally*)
- $\bigcirc \leftrightarrow$ X (*next*)

Opérateurs complémentaires

- Opérateur *waiting-for* (ou *unless* ou *weak-until*)
  $P \mathcal{W} Q \triangleq \Box P \vee P \mathcal{U} Q$
  Q finit peut-être par être vrai et en attendant P reste vrai
- Opérateur *release*
  $P \mathcal{R} Q \triangleq Q \mathcal{U}(P \wedge Q)$
  Q reste vrai jusqu'à ce que P le devienne.

Opérateurs du passé

formule	nom	interprétation
$\ominus P$	*previously*	P est vrai dans l'instant précédent
$\boxminus P$	*has-always-been*	P a toujours été vrai jusqu'à l'instant courant
$\diamondminus P$	*once*	P a été vrai dans le passé
$P \mathcal{S} Q$	*since*	Q a été vrai dans le passé et P est resté vrai depuis la dernière occurrence de Q
$P \mathcal{B} Q$	*back-to*	P est vrai depuis la dernière occurrence de Q, ou depuis l'instant initial si Q n'a jamais été vrai

Peu utilisés en pratique.

Sémantique (système)

On note (σ, i) pour le suffixe $\langle s_i \to s_{i+1} \to \ldots \rangle$ d'une trace $\sigma = \langle s_0 \to s_1 \to \ldots \rangle$.

Vérification par un système

Un système $\mathcal{S}$ vérifie (valide) la formule F ssi toutes les exécutions de $\mathcal{S}$ la valident à partir de l'instant initial :

$$\frac{\forall \sigma \in Exec(\mathcal{S}) : (\sigma, 0) \models F}{\mathcal{S} \models F}$$

Rappel : les exécutions d'un système sont ses traces finies maximales et infinies, et qui débutent par un état initial.

Sémantique (opérateurs logiques)

Sémantique standard des opérateurs logiques

$$\frac{(\sigma, i) \models P \quad (\sigma, i) \models Q}{(\sigma, i) \models P \wedge Q}$$

$$\frac{(\sigma, i) \models P}{(\sigma, i) \models P \vee Q} \qquad \frac{(\sigma, i) \models Q}{(\sigma, i) \models P \vee Q}$$

$$\frac{\neg\, (\sigma, i) \models P}{(\sigma, i) \models \neg P}$$

Sémantique (opérateurs temporels)

$$\frac{\sigma_i = s}{(\sigma, i) \models s}$$

$$\frac{(\sigma, i+1) \models P}{(\sigma, i) \models \bigcirc P}$$

$$\frac{\exists k \geq 0 : (\sigma, i+k) \models Q \land \forall k', 0 \leq k' < k : (\sigma, i+k') \models P}{(\sigma, i) \models P \mathcal{U} Q}$$

Sémantique (opérateurs temporels dérivés)

$$\frac{\exists k \geq 0 : (\sigma, i+k) \models P}{(\sigma, i) \models \Diamond P}$$

$$\frac{\forall k \geq 0 : (\sigma, i+k) \models P}{(\sigma, i) \models \Box P}$$

$$\frac{\forall k \geq 0 : ((\sigma, i+k) \models P \Rightarrow \exists k' \geq k : (\sigma, i+k') \models Q)}{(\sigma, i) \models P \leadsto Q}$$

Réduction à la logique pure

- La logique temporelle linéaire possède une expressivité telle qu'elle peut représenter exactement n'importe quelle spécification opérationnelle décrite en termes de système de transitions, d'où :
- vérifier qu'un système de transitions $\mathcal{M}$ possède la propriété temporelle $F_{\mathcal{S}pec}$:

$$\mathcal{M} \models F_{\mathcal{S}pec}$$

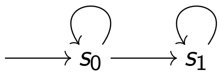
- revient à déterminer la validité de :

$$F_{\mathcal{M}} \Rightarrow F_{\mathcal{S}pec}$$

où $F_{\mathcal{M}}$ est une formule représentant exactement les exécutions du modèle $\mathcal{M}$ (i.e. ses états initiaux, ses transitions, ses contraintes d'équité).

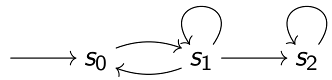
Plan

Exemple 1



	pas d'équité	équité faible (s_0, s_1)
$s_0 \wedge \bigcirc s_0$		
$s_0 \wedge \bigcirc(s_0 \vee s_1)$		
$\Box(s_0 \Rightarrow \bigcirc s_0)$		
$\Box(s_0 \Rightarrow \bigcirc(s_0 \vee s_1))$		
$\Box(s_1 \Rightarrow \bigcirc s_1)$		
$\Diamond(s_0 \wedge \bigcirc s_1)$		
$\Box s_0$		
$\Diamond\neg s_0$		
$\Diamond\Box s_1$		
$s_0 \mathcal{W} s_1$		
$s_0 \mathcal{U} s_1$		

Exemple 2



	pas d'équité	faible (s_1, s_2)	forte (s_1, s_2)
$\Box\Diamond\neg s_1$			
$\Box(s_1 \Rightarrow \Diamond s_2)$			
$\Diamond\Box(s_1 \vee s_2)$			
$\Box(s_1 \mathcal{U} s_2)$			
$\Box(s_0 \Rightarrow s_0 \mathcal{U} s_1)$			
$\Box(s_0 \mathcal{U}(s_1 \vee s_2))$			
$\Box(s_1 \Rightarrow s_1 \mathcal{U} s_2)$			
$\Diamond(s_1 \mathcal{U} s_2)$			
$\Diamond(s_1 \mathcal{W} s_2)$			
$\Box\Diamond(s_1 \mathcal{U}(s_0 \vee s_2))$			

Sûreté/vivacité – *Safety/Liveness*

On qualifie de

- Sûreté : rien de mauvais ne se produit
  = propriété qui s'invalide sur un préfixe fini d'une exécution :
  $\Box P$, $\Box(P \Rightarrow \Box P)$, $P\mathcal{W}Q$...
- Vivacité : quelque chose de bon finit par se produire
  = propriété qui peut toujours être validée en étendant le préfixe d'une exécution :
  $\Diamond P$, $P \leadsto Q$...
- Certaines propriétés combinent vivacité et sûreté :
  $P\mathcal{U}Q$, $\Box P \wedge \Diamond Q$...
  - Réponse : $\Box\Diamond P$
  - Persistance : $\Diamond\Box P$

Invariance, stabilité

Invariance

Spécifier un sur-ensemble des états accessibles d'un système :

$$\mathcal{S} \models \Box P$$

où P est un prédicat d'état.

Stabilité

Spécifier la stabilité d'une situation si elle survient :

$$\mathcal{S} \models \Box(P \Rightarrow \Box P)$$

où P est un prédicat d'état.

Possibilité

Possibilité

Spécifier qu'il est possible d'atteindre un certain état vérifiant P dans une certaine exécution :
Impossible pour P arbitraire, mais pour P un prédicat d'état :

$$\mathcal{S} \not\models \Box\neg P$$

Attention à la négation : $\neg\Box P = \Diamond\neg P$ mais $\mathcal{S} \not\models \Box P \not\Rightarrow \mathcal{S} \models \Diamond\neg P$

Négation

Négation : danger !

Pour σ exécution : $\sigma \models \neg P \equiv \sigma \not\models P$

Pour $\mathcal{S}$ système : $\mathcal{S} \models \neg P \Rightarrow \mathcal{S} \not\models P$ mais pas l'inverse !

$\mathcal{S} \not\models Q$ signifie qu'il existe au moins une exécution qui invalide Q ($=$ qui valide $\neg Q$), mais pas que toutes les exécutions le font.
En LTL, on peut avoir $\mathcal{S} \not\models Q \wedge \mathcal{S} \not\models \neg Q$:

$$\begin{tikzcd} {} \arrow[r] & s_0 \arrow[loop above] \arrow[r] & s_1 \arrow[loop above] \end{tikzcd}$$

$$\frac{s_0^{+} \to s_1^{\omega} \not\models \Box s_0}{\mathcal{S} \not\models \Box s_0} \qquad \frac{s_0^{\omega} \not\models \Diamond\neg s_0}{\mathcal{S} \not\models \Diamond\neg s_0}$$

Combinaisons

Infiniment souvent – Réponse

Spécifier que P est infiniment souvent vrai dans toute exécution :

$$\mathcal{S} \models \Box\Diamond P$$

Finalement toujours – Persistance

Spécifier que P finit par rester définitivement vrai :

$$\mathcal{S} \models \Diamond\Box P$$

Note : $\Box\Box P = \Box P$ et $\Diamond\Diamond P = \Diamond P$

Client/serveur

Réponse

Spécifier qu'un système (jouant le rôle d'un serveur) répond toujours (Q) à un requête donnée (P) :

$$\mathcal{S} \models \Box(P \Rightarrow \Diamond Q)$$

Souvent nommé leads-to :

$$\mathcal{S} \models P \leadsto Q$$

Stabilité d'une requête

Spécifier que la requête P d'un système (jouant le rôle d'un client) est stable tant qu'il n'y a pas de réponse favorable Q :

$$\mathcal{S} \models \Box(P \Rightarrow P \mathcal{W} Q)$$

Équité des transitions – *Fairness*

Rappel informel :
faible : continûment faisable → infiniment souvent fait
forte : infiniment souvent faisable → infiniment souvent fait

Équité faible des transitions

Soit $r \subseteq R$. Les transitions r sont en équité faible dans $\mathcal{S}$:

$$\mathcal{S} \models \Diamond\Box dom(r) \Rightarrow \Box\Diamond r$$
$$\mathcal{S} \models \Box\Diamond\neg dom(r) \vee \Box\Diamond r$$

Équité forte des transitions

Soit $r \subseteq R$. Les transitions r sont en équité forte dans $\mathcal{S}$:

$$\mathcal{S} \models \Box\Diamond dom(r) \Rightarrow \Box\Diamond r$$
$$\mathcal{S} \models \Diamond\Box\neg dom(r) \vee \Box\Diamond r$$

(une transition $s \rightarrow s'$ est équivalente à $s \wedge \bigcirc s'$, et un ensemble de transition $\{t_1, t_2, \ldots\}$ est équivalent à $t_1 \vee t_2 \vee \ldots$)

Spécification d'un système de transitions

Si on utilise une description en intention, et si l'on remplace l'utilisation de l'opérateur $\bigcirc$ par les variables primées, alors on peut spécifier toutes les exécutions permises par un système $\langle S, I, R\rangle$:

$$\mathcal{S} \models I \wedge \Box R$$

L'utilisation de variables primées n'est pas nécessaire mais simplifie les formules.
Par exemple $P(x, x')$ est équivalent à la formule :
$\forall v : x = v \Rightarrow \bigcirc P(v, x)$
qui nécessite une quantification sur une variable.

Limites de l'expressivité

Tout n'est pas exprimable en LTL :

- Possibilité arbitraire : si P devient vrai, il est toujours possible (mais pas nécessaire) que Q le devienne après.
- Accessibilité d'un état : depuis l'état initial, il est possible d'atteindre cet état.
- Réinitialisabilité : quelque soit l'état, il est possible de revenir dans un des états initiaux.

(ces propriétés sont exprimables en Computational Tree Logic (CTL), à venir)

Conclusion

La logique temporelle linéaire (LTL) permet d'exprimer, abstraitement, des propriétés sur les exécutions d'un système

Logiques modales

La LTL est un cas particulier de logique modale.
Autres interprétations :

- $\Box$ = nécessité, $\Diamond$ = possibilité
- logique de la croyance : « je crois que P est vrai »
- logique épistémique : « X sait que P »
- logique déontique : « P est obligatoire/interdit/permis »
- ...

Cinquième partie

TLA+– la logique

Objectifs

- Exprimer des propriétés vérifiées par une spécification TLA+
- Exprimer l'équité garantissant la progression
- Démontrer des liens entre spécifications (équivalence, raffinage)
- Pouvoir vérifier ces propriétés de manière mécanisée (automatiquement – vérification de modèles –, ou manuellement – preuve axiomatique –)

Plan

La logique TLA+

Expressions logiques

Expressions de LTL avec $\Box$, $\Diamond$, $\leadsto$ (leads-to) et variables primées + quantificateurs $\forall, \exists$.

Pas de $\mathcal{U}$, ni de $\mathcal{W}$, mais :

$$\begin{array}{lcl}\Box(p \Rightarrow (p\mathcal{W}q)) & = & \Box(p \Rightarrow (p' \vee q)) \\ \Box(p \Rightarrow (p\mathcal{U}q)) & = & \Box(p \Rightarrow (p' \vee q)) \wedge \Box(p \Rightarrow \Diamond q)\end{array}$$

Équité / *Fairness*

ENABLED

ENABLED $\mathcal{A}$ est la fonction d'état qui est vraie dans l'état s ssi il existe un état t accessible depuis s par l'action $\mathcal{A}$.

Weak/Strong Fairness

- $\mathrm{WF}_e(\mathcal{A}) \triangleq \Box\Diamond\neg(\textsc{enabled}\ \langle\mathcal{A}\rangle_e) \vee \Box\Diamond\langle\mathcal{A}\rangle_e$
  si $\mathcal{A}$ est constamment déclenchable, elle sera déclenchée.
- $\mathrm{SF}_e(\mathcal{A}) \triangleq \Diamond\Box\neg(\textsc{enabled}\ \langle\mathcal{A}\rangle_e) \vee \Box\Diamond\langle\mathcal{A}\rangle_e$
  si $\mathcal{A}$ est infiniment souvent déclenchable, elle sera déclenchée.

Forme d'une spécification TLA+

En général, une spécification TLA+ est une conjonction

$$\mathcal{I} \wedge \Box[\mathcal{N}]_v \wedge \mathcal{E}$$

- $\mathcal{I}$ = prédicat d'état décrivant les états initiaux
- $\mathcal{N}$ = disjonction d'actions $\mathcal{A}_1 \vee \mathcal{A}_2 \vee \mathcal{A}_3 \vee \ldots$
- $\mathcal{E}$ = conjonction de contraintes d'équité portant sur les actions : $\mathrm{WF}_v(A_1) \wedge \mathrm{SF}_v(A_3) \wedge \ldots$

Raffinage de spécification

Raffinage simple

Une spécification (concrète) *Pc* raffine une spécification (abstraite) *Pa* si $Pc \Rightarrow Pa$: tout ce que fait *Pc* est possible dans *Pa*.

Cela signifie que si $Pa \models P$ pour une propriété LTL quelconque, alors $Pc \models P$.

Raffinage – exemple

Somme abstraite

MODULE *somme1*

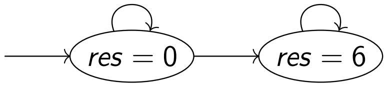
EXTENDS *Naturals*
CONSTANT *N*
VARIABLE *res*

$TypeInvariant \triangleq res \in Nat$

$Init \triangleq res = 0$
$Next \triangleq res' = ((N+1) * N) \div 2$
$Spec \triangleq Init \wedge \Box[Next]_{res} \wedge \mathrm{WF}_{res}(Next)$

Raffinage – exemple

Graphe des exécutions pour N = 3



Raffinage – exemple

Somme plus concrète

MODULE *somme2*

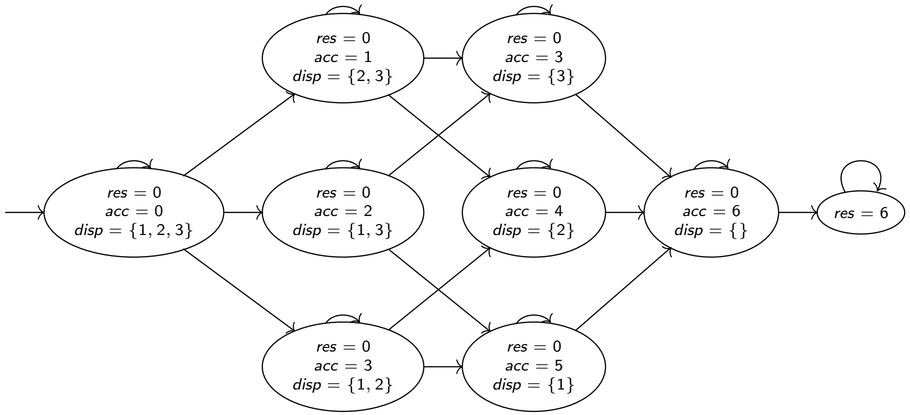
EXTENDS *Naturals*
CONSTANT N
VARIABLE $res, acc, disp$

$$TypeInvariant \triangleq res \in Nat \land acc \in Nat \land disp \in \text{SUBSET } 1 \,..\, N$$

$$Init \triangleq res = 0 \land acc = 0 \land disp = 1 \,..\, N$$
$$\begin{aligned} Next \triangleq\ & \lor \exists i \in disp : acc' = acc + i \land disp' = disp \setminus \{i\} \\ & \qquad \land \text{UNCHANGED } res \\ & \lor disp = \{\} \land res' = acc \land \text{UNCHANGED } \langle disp, acc \rangle \end{aligned}$$
$$Spec \triangleq Init \land \Box[Next]_{\langle res, disp, acc \rangle} \land \mathrm{WF}_{\langle res, disp, acc \rangle}(Next)$$

Raffinage – exemple

Graphe des exécutions pour N = 3



Décomposition : introduction de transitions intermédiaires.

Raffinage – exemple

Somme2 raffine somme1

MODULE *somme2_raffine_somme1*

EXTENDS *somme2*
$Orig \triangleq$ INSTANCE *somme1*
$Raffinement \triangleq Orig!Spec$
THEOREM $Spec \Rightarrow Orig!Spec$

Raffinage – exemple

Somme concrète

MODULE *somme3*

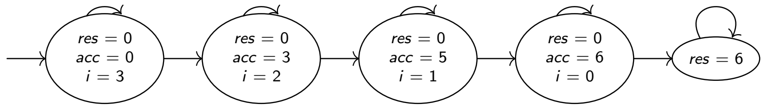
EXTENDS *Naturals*
CONSTANT N
VARIABLE res, acc, i

$$TypeInvariant \triangleq res \in Nat \land acc \in Nat \land i \in 1 \,..\, N$$

$$Init \triangleq res = 0 \land acc = 0 \land i = N$$
$$Next \triangleq \lor\ i > 0 \land acc' = acc + i \land i' = i - 1 \land \text{UNCHANGED } res$$
$$\qquad\qquad \lor\ i = 0 \land res' = acc \land \text{UNCHANGED } \langle i, acc \rangle$$
$$Spec \triangleq Init \land \Box[Next]_{\langle res, i, acc \rangle} \land \mathrm{WF}_{\langle res, i, acc \rangle}(Next)$$

Raffinage – exemple

Graphe des exécutions pour N = 3



Réduction du non-déterminisme + changement de représentation (raffinement de données) $disp = 1..i$

Raffinage – exemple

Somme3 raffine somme2

MODULE *somme3_raffine_somme2*

EXTENDS *somme3*
$$dispMapping \triangleq 1 \,..\, i$$
$$Orig \triangleq \text{INSTANCE } somme2 \text{ WITH } disp \leftarrow dispMapping$$
$$Raffinement \triangleq Orig!Spec$$
$$\text{THEOREM } Spec \Rightarrow Orig!Spec$$

Plan

Règles de preuve – simple temporal logic

$$\frac{F \text{ prouvable en logique propositionnelle}}{\Box F}\,\textsc{stl1} \qquad \frac{F \Rightarrow G}{\Box F \Rightarrow \Box G}\,\textsc{stl4}$$

$$\frac{}{\Box F \Rightarrow F}\,\textsc{stl2} \qquad \frac{}{\Box\Box F = \Box F}\,\textsc{stl3}$$

$$\frac{}{\Box(F \land G) = (\Box F) \land (\Box G)}\,\textsc{stl5} \qquad \frac{}{\Diamond\Box F \land \Diamond\Box G = \Diamond\Box(F \land G)}\,\textsc{stl6}$$

Règles de preuve – TLA$^+$ invariant

$$\frac{P \land (v' = v) \Rightarrow P'}{\Box P = (P \land \Box[P \Rightarrow P']_v)}\,\textsc{tla1} \qquad \frac{P \land [\mathcal{A}]_{v_1} \Rightarrow Q \land [\mathcal{B}]_{v_2}}{\Box P \land \Box[\mathcal{A}]_{v_1} \Rightarrow \Box Q \land \Box[\mathcal{B}]_{v_2}}\,\textsc{tla2}$$

TLA1 : principe d'induction pour prouver $\Box P$ à partir de l'état initial et de la conservation de P. TLA2 : le raffinage de spécifications se ramène au raffinage des actions.

$$\frac{I \land [\mathcal{N}]_v \Rightarrow I'}{I \land \Box[\mathcal{N}]_v \Rightarrow \Box I}\,\textsc{inv1} \qquad \frac{}{\Box I \Rightarrow (\Box[\mathcal{N}]_v = \Box[\mathcal{N} \land I \land I']_v)}\,\textsc{inv2}$$

INV1 : preuve par induction d'un invariant
Hypothèse : I est préservé par $\mathcal{N}$ et le bégaiement.
Conclusion : si I est initialement vrai et toute transition vérifie $\mathcal{N}$ ou bégaiement ($\Box[\mathcal{N}]$), alors I est un invariant ($\Box I$).
INV2 : injection d'un invariant dans la spécification.

Règles de preuve – TLA$^+$ vivacité avec équité faible

$$\frac{\begin{array}{c} P \land [\mathcal{N}]_v \Rightarrow (P' \lor Q') \\ P \land \langle \mathcal{N} \land \mathcal{A} \rangle_v \Rightarrow Q' \\ P \Rightarrow \text{ENABLED } \langle \mathcal{A} \rangle_v \end{array}}{\Box[\mathcal{N}]_v \land WF_v(\mathcal{A}) \Rightarrow (P \leadsto Q)}\,\textsc{wf1}$$

Hypothèses :

1. si on a P, en faisant une transition quelconque ($[\mathcal{N}]$), on conserve P ou on établit Q ;
2. Il y a une action $\mathcal{A}$ qui établit Q ;
3. Quand P est vrai, l'action $\mathcal{A}$ est faisable.

Alors, sous contrainte d'équité faible sur $\mathcal{A}$, si P est vrai, $\mathcal{A}$ doit finir par avoir lieu (car P reste constamment vrai au moins jusqu'à établir Q et P garantit que $\mathcal{A}$ est faisable), et donc Q finira par être vrai.

Règles de preuve – TLA$^+$ vivacité avec équité forte

$$\frac{\begin{array}{c} P \land [\mathcal{N}]_v \Rightarrow (P' \lor Q') \\ P \land \langle \mathcal{N} \land \mathcal{A} \rangle_v \Rightarrow Q' \\ \Box P \land \Box[\mathcal{N}]_v \land \Box F \Rightarrow \Diamond\text{ENABLED } \langle \mathcal{A} \rangle_v \end{array}}{\Box[\mathcal{N}]_v \land SF_v(\mathcal{A}) \land \Box F \Rightarrow (P \leadsto Q)}\,\textsc{sf1}$$

Hypothèses :

1. si on a P, en faisant une transition quelconque ($[\mathcal{N}]$), on conserve P ou on établit Q ;
2. Il y a une action $\mathcal{A}$ qui établit Q ;
3. Si P est constamment vrai, l'action $\mathcal{A}$ finira par être faisable.

Alors, sous contrainte d'équité forte sur $\mathcal{A}$, si P est vrai, $\mathcal{A}$ doit finir par avoir lieu (car P reste constamment vrai au moins jusqu'à établir Q et P garantit que $\mathcal{A}$ sera faisable, donc est infiniment souvent faisable), et donc Q finira par être vrai.
($\Box F$ est un invariant qui facilite en général la preuve du 3)

Règles de preuve dérivées

$$\frac{\Box(P \Rightarrow \Box P) \land \Diamond P}{\Diamond\Box P}\text{LDSTBL}$$

$\Box(P \Rightarrow \Box P)$ signifie que P est stable : une fois vrai, il le reste. Combiné avec $\Diamond P$ (un jour P sera vrai), on obtient que P est finalement toujours vrai.

$$\frac{P \leadsto Q \land Q \leadsto R}{P \leadsto R}\text{TRANS}$$

Transitivité du $\leadsto$.

Plan

Vérification de modèles

Principe

Construire le graphe des exécutions et étudier la propriété.

- $\Box P$, où P est un prédicat d'état (sans variable primée) : au fur et à mesure de la construction des états.
- $\Box P(v, v')$, où $P(v, v')$ est un prédicat de transition (prédicat non temporel avec variables primées et non primées) : au fur et à mesure du calcul des transitions.
- Vivacité $\Diamond P$, $P \leadsto Q\ldots$: une fois le graphe construit, chercher un cycle qui respecte les contraintes d'équité et qui invalide la propriété.

Uniquement sur des modèles finis, et, pratiquement, de petites tailles.

Complexité

Soit $|S|$ le nombre d'états d'un système $\mathcal{S} = \langle S, I, R\rangle$ et $|F|$ la taille (le nombre d'opérateurs temporels) d'une formule LTL F. La complexité en temps (et espace) pour vérifier $\mathcal{S} \models F$ est $O(|S| \times 2^{|F|})$.

Vérificateur TLC

Le vérificateur de modèles TLC sait vérifier :

- les spécifications avec des actions gardées ;
- (efficacement) les invariants sans variables primées : $\Box P$ où P est un prédicat d'état ;
- les formules de sûreté pure avec variables primées et bégaiement : $\Box[P]_v$ où P est un prédicat de transition ;
- $P \leadsto Q$ où P et Q sont des prédicats d'état (*sans* variables primées) ;
- les formules combinant $\Box, \Diamond$ *sans* variables primées.

Note : l'espace d'états du système et des formules doit être fini : toute quantification bornée par exemple.

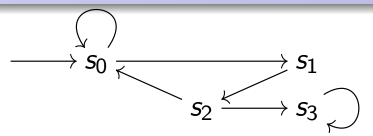
Sixième partie

CTL – logique temporelle arborescente

Plan

Ensemble des exécutions vs arbre des exécutions ♪♪♪

Soit le système de transitions :

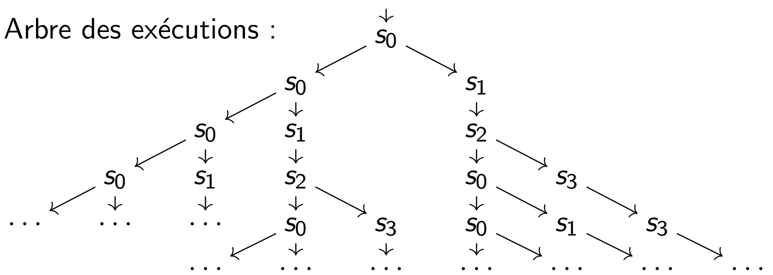


Ensemble des exécutions :

$\langle((s_0{}^+ \to s_1 \to s_2)^* \to s_0{}^\omega\rangle, \langle((s_0{}^+ \to s_1 \to s_2)^\omega\rangle, \langle((s_0{}^+ \to s_1 \to s_2)^+ \to s_3{}^\omega\rangle$

$$\text{ou}\left\{\begin{array}{l} s_0 \to s_0 \to \cdots, s_0 \to s_1 \to s_2 \to s_0 \to s_0 \to \cdots, \\ s_0 \to s_1 \to s_2 \to s_0 \to \cdots, s_0 \to s_0 \to s_1 \to s_2 \to s_0 \to \cdots, \ldots \\ s_0 \to s_1 \to s_2 \to s_3 \to s_3 \to \cdots, \ldots \end{array}\right\}$$

Arbre des exécutions :



Computational Tree Logic – logique temporelle arborescente ♪♪♪

Modèles

Une formule CTL se rapporte toujours à un état donné s d'un système, duquel partent des traces $Traces(s)$.
Les états de S constituent les modèles de cette logique.

La différence (syntaxiquement parlant) avec LTL réside dans l'apparition dans les opérateurs temporels de quantificateurs de traces.

Syntaxe de la CTL ♪♪♪

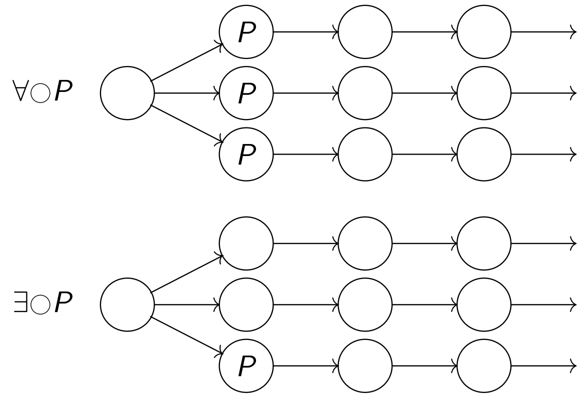
Quantification universelle

formule	interprétation (pour s un état) pour toute trace partant de s
$\forall\bigcirc P$	P est vrai à l'instant suivant
$\forall\Box P$	P est toujours vrai à chaque état
$\forall\Diamond P$	P finit par être vrai (dans le futur)
$P \,\forall\mathcal{U}\, Q$	Q finit par être vrai, et en attendant P reste vrai

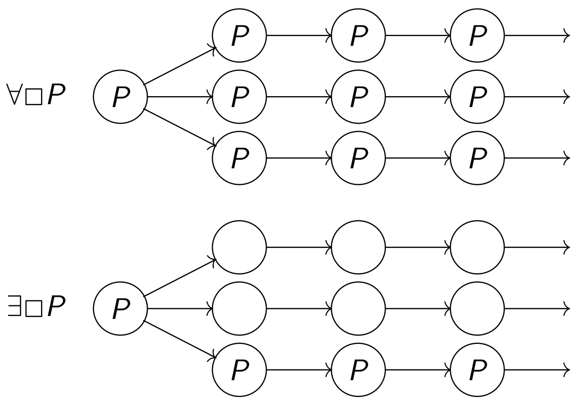
Quantification existentielle

formule	interprétation (pour s un état) pour au moins une trace partant de s
$\exists\bigcirc P$	P est vrai à l'instant suivant
$\exists\Box P$	P est toujours vrai à chaque état
$\exists\Diamond P$	P finit par être vrai (dans le futur)
$P \,\exists\mathcal{U}\, Q$	Q finit par être vrai, et en attendant P reste vrai

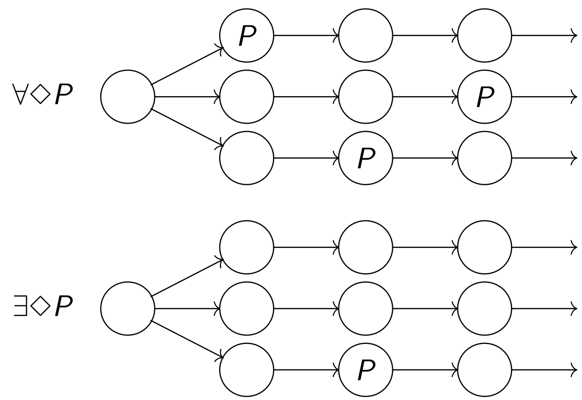
Intuition sémantique $\forall\bigcirc$, $\exists\bigcirc$



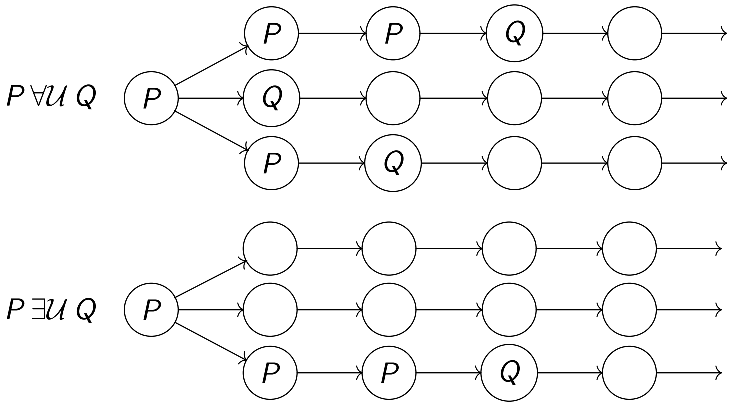
Intuition sémantique $\forall\Box$, $\exists\Box$



Intuition sémantique $\forall\Diamond$, $\exists\Diamond$



Intuition sémantique $\forall\mathcal{U}$, $\exists\mathcal{U}$



Opérateurs minimaux ♪♪♪

Un ensemble d'opérateurs minimaux est $\forall\bigcirc$, $\forall\mathcal{U}$ et $\exists\mathcal{U}$:

- $\exists\bigcirc P \triangleq \neg\forall\bigcirc\neg P$
- $\forall\Diamond P \triangleq \mathit{True}\, \forall\mathcal{U}\, P$
- $\exists\Diamond P \triangleq \mathit{True}\, \exists\mathcal{U}\, P$
- $\forall\Box P \triangleq \neg\exists\Diamond\neg P$
- $\exists\Box P \triangleq \neg\forall\Diamond\neg P$

(autres ensembles minimaux : $\{\exists\bigcirc, \exists\Box, \exists\mathcal{U}\}$ ou $\{\forall\Diamond, \exists\mathcal{U}, \exists\bigcirc\}$)

Syntaxe alternative

Syntaxe alternative

On trouve très fréquemment une autre syntaxe :

$\forall$	$\leftrightarrow$	A (all)
$\exists$	$\leftrightarrow$	E (exists)
$\Box$	$\leftrightarrow$	G (globally)
$\Diamond$	$\leftrightarrow$	F (finally)
$\bigcirc$	$\leftrightarrow$	X (next)

Par exemple :

$\forall\Box\exists\Diamond P \leftrightarrow$ AG EF P

$f \,\forall\mathcal{U}\, g \leftrightarrow$ A(f U g)

Opérateur complémentaire waiting-for ♪♪♪

$P \exists\mathcal{W}\, Q \triangleq \exists\Box P \vee P \exists\mathcal{U}\, Q$

$P \forall\mathcal{W}\, Q \not\triangleq \forall\Box P \vee P \forall\mathcal{U}\, Q$ – *trop fort*

$\triangleq \neg(\neg Q \,\exists\mathcal{U} (\neg P \wedge \neg Q))$

Sémantique (système) ♪♪♪

La relation de validation sémantique ne fait intervenir que l'état courant.

Vérification par un système

Un système $\mathcal{S} = \langle S, \{s_0\}, R\rangle$ vérifie (valide) la formule F ssi l'état initial de $\mathcal{S}$ la valide :

$$\frac{s_0 \models F}{\mathcal{S} \models F}$$

(la sémantique est moins claire s'il y a plusieurs états initiaux, du fait de l'opérateur $\exists$: pour tous les états initiaux, ou pour au moins un ? En pratique, on peut toujours se ramener à un seul état initial, donc on évite la difficulté)

Sémantique (opérateurs logiques)

$$\frac{}{s \models s}$$

$$\frac{s \models P \quad s \models Q}{s \models P \wedge Q}$$

$$\frac{s \models P}{s \models P \vee Q} \qquad \frac{s \models Q}{s \models P \vee Q}$$

$$\frac{s \models P}{s \not\models \neg P}$$

Sémantique (opérateurs temporels) ♪♪♪

(rappel : pour une trace σ, σ_i est le i-ième élément de σ en commençant à 0, et pour un état s, $Traces(s)$ est l'ensemble des traces issues de s)

$$\frac{\forall \sigma \in Traces(s) : \sigma_1 \models P}{s \models \forall \bigcirc P}$$

$$\frac{\forall \sigma \in Traces(s) : \exists j \geq 0 : \sigma_j \models Q \wedge \forall i < j : \sigma_i \models P}{s \models P \, \forall \mathcal{U} \, Q}$$

$$\frac{\exists \sigma \in Traces(s) : \exists j \geq 0 : \sigma_j \models Q \wedge \forall i < j : \sigma_i \models P}{s \models P \, \exists \mathcal{U} \, Q}$$

Sémantique (opérateurs temporels dérivés)

$$\frac{\exists \sigma \in Traces(s) : \sigma_1 \models P}{s \models \exists \bigcirc P}$$

$$\frac{\forall \sigma \in Traces(s) : \forall i \geq 0 : \sigma_i \models P}{s \models \forall \Box P}$$

$$\frac{\exists \sigma \in Traces(s) : \forall i \geq 0 : \sigma_i \models P}{s \models \exists \Box P}$$

$$\frac{\forall \sigma \in Traces(s) : \exists i \geq 0 : \sigma_i \models P}{s \models \forall \Diamond P}$$

$$\frac{\exists \sigma \in Traces(s) : \exists i \geq 0 : \sigma_i \models P}{s \models \exists \Diamond P}$$

Négation ♪♪♪

Négation

Contrairement à LTL, pour toute propriété CTL, on a :
soit $\mathcal{S} \models F$, soit $\mathcal{S} \models \neg F$,
et $\mathcal{S} \not\models F \equiv \mathcal{S} \models \neg F$.

Négation des formules $\forall, \exists, \Box, \Diamond$

La négation d'une formule à base de $\forall, \exists, \Box, \Diamond$ se fait simplement en inversant chaque opérateur pour son dual.

exemples :
$\neg(\forall \Diamond \exists \Box p) = \exists \Box \forall \Diamond \neg p$
$(\forall \Diamond \neg s_0 \Rightarrow \forall \Diamond s_3) = (\exists \Box s_0 \vee \forall \Diamond s_3)$ car $(p \Rightarrow q) = (\neg p \vee q)$

Définition par point-fixe

Une fois définis $\exists\bigcirc$ et $\forall\bigcirc$, chaque opérateur est le plus petit point fixe de sa définition inductive :

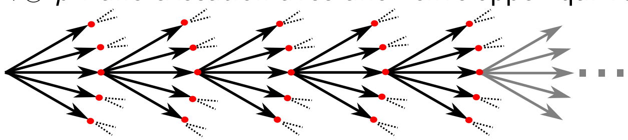
$$
\begin{array}{lcl}
\forall\Box f & = & f \wedge \forall\bigcirc\forall\Box f \\
\exists\Box f & = & f \wedge \exists\bigcirc\exists\Box f \\
\forall\Diamond f & = & f \vee \forall\bigcirc\forall\Diamond f \\
\exists\Diamond f & = & f \vee \exists\bigcirc\exists\Diamond f \\
f\,\forall\mathcal{U}\,g & = & g \vee (f \wedge \forall\bigcirc(f\,\forall\mathcal{U}\,g)) \\
f\,\exists\mathcal{U}\,g & = & g \vee (f \wedge \exists\bigcirc(f\,\exists\mathcal{U}\,g))
\end{array}
$$

(surtout utile pour l'implantation d'un vérificateur de modèles)

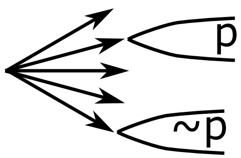
Plan

Exemples amusants ♪♪♪

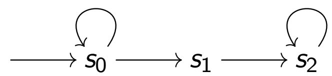
- $\exists\Box\ \forall\bigcirc\ p$: une exécution avec une “enveloppe” qui vérifie p



- $\exists\bigcirc\ \forall\Box p\ \wedge\ \exists\bigcirc\ \forall\Box\neg p$
  un état successeur à partir duquel p est toujours et partout vrai,
  et un état successeur à partir duquel $\neg p$ est toujours et partout vrai



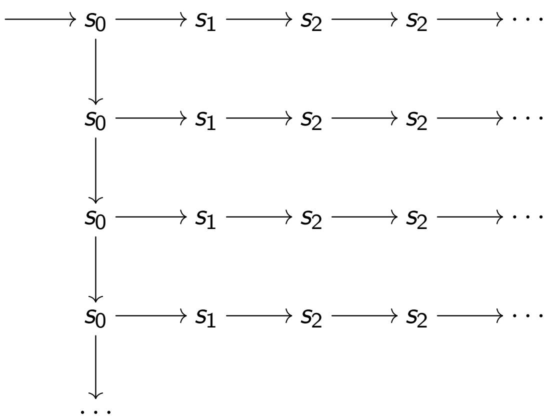
Exemple ♪♪♪



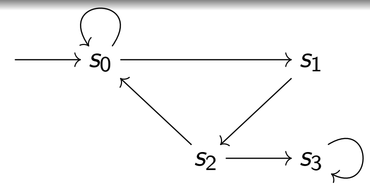
	pas d'équité	équité faible (s_0, s_1)
$s_0 \wedge \forall\bigcirc s_0$		
$s_0 \wedge \exists\bigcirc s_0$		
$\forall\Box(s_0 \Rightarrow \exists\bigcirc s_0)$		
$\forall\Box(s_0 \Rightarrow \exists\Diamond s_2)$		
$\forall\Box(s_0 \Rightarrow \forall\Diamond s_2)$		
$\exists\Diamond\neg s_0$		
$\forall\Diamond\neg s_0$		
$\forall\Box\exists\Diamond s_2$		
$\forall\Box\forall\Diamond s_2$		
$\forall\Diamond\exists\Diamond s_1$		
$\forall\Box\exists\Diamond s_1$		

Exemple - Arbre des exécutions

Arbre des exécutions du système de transition précédent



Exemple 2 ♪♪♪



	pas d'équité	faible (s_0, s_1)	forte (s_2, s_3)	forte (s_2, s_3) faible (s_0, s_1)
$\exists\Box s_0$				
$\forall\Box\exists\Diamond s_3$				
$\forall\Box\forall\Diamond s_3$				
$\forall\Diamond\forall\Box s_3$				
$\exists\Box s_0 \vee \forall\Diamond s_3$				
$\forall\Diamond\neg s_0 \Rightarrow \forall\Diamond s_3$				

Invariance, Possibilité

Invariance

Spécifier un sur-ensemble des états accessibles d'un système :

$$\mathcal{S} \models \forall\Box P$$

où P est un prédicat d'état.

Stabilité

Spécifier la stabilité d'une situation si elle survient :

$$\mathcal{S} \models \forall\Box(P \Rightarrow \forall\Box P)$$

où P est un prédicat d'état.

Possibilité

Spécifier qu'il est possible d'atteindre un état vérifiant P :

$$\mathcal{S} \models \exists\Diamond P$$

Possibilité complexe

Séquence

Spécifier qu'un scénario d'exécution $\langle s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_n\rangle$ est possible.

$$\mathcal{S} \models s_1 \wedge \exists\bigcirc(s_2 \wedge \ldots \wedge \exists\bigcirc(s_{n-1} \wedge \exists\bigcirc s_n)\ldots)$$

Réinitialisabilité

Spécifier que quelque soit l'état courant, il est possible de revenir dans un des états initiaux (définis par le prédicat I).

$$\mathcal{S} \models \forall\Box\exists\Diamond I$$

Possibilité arbitraire

Spécifier que si P devient vrai, il est toujours possible (mais pas nécessaire) que Q le devienne après.

$$\mathcal{S} \models \forall\Box(P \Rightarrow \exists\Diamond Q)$$

Client/serveur

Réponse

Spécifier qu'un système (jouant le rôle d'un serveur) répond toujours (Q) à une requête donnée (P) :

$$\mathcal{S} \models \forall\Box(P \Rightarrow \forall\Diamond Q)$$

Stabilité d'une requête

Spécifier que la requête P d'un système (jouant le rôle d'un client) est stable tant qu'il n'y a pas de réponse favorable Q :

$$\mathcal{S} \models \forall\Box(P \Rightarrow P \forall\mathcal{W}\, Q)$$

Combinaisons ♪♪♪

Infiniment souvent

Spécifier que P est infiniment souvent vrai dans toute exécution :

$$\mathcal{S} \models \forall\Box\forall\Diamond P$$

Finalement toujours

Spécifier que P finit par rester définitivement vrai :

impossible ! $\mathcal{S} \models \forall\Diamond\forall\Box P$ ne convient pas (trop fort)

Soit $S =$

$$\begin{tikzcd} {} \arrow[r] & s_0 \arrow[loop above] \arrow[r] & s_1 \arrow[r] & s_2 \arrow[loop above] \end{tikzcd}$$

en LTL : $S \models \Diamond\Box(s_0 \vee s_2)$

mais CTL : $S \not\models \forall\Diamond\forall\Box(s_0 \vee s_2)$

(tant qu'on est en s_0, on *peut* passer en s_1 : $S \models \forall\Diamond\exists\Diamond s_1$)

Note : $\mathcal{X}\mathcal{X}P = \mathcal{X}P$ pour $\mathcal{X} \in \{\forall\Box, \exists\Box, \forall\Diamond, \exists\Diamond\}$

Spécification d'un ST

Si on utilise une description en intention, et si l'on remplace l'utilisation de l'opérateur $\forall\bigcirc$ par les variables primées, alors on peut spécifier toutes les exécutions permises par un système $\langle S, I, R\rangle$:

$$\mathcal{S} \models I \wedge \forall\Box R$$

L'utilisation de variables primées n'est pas nécessaire mais simplifie les formules.

Par exemple $P(x, x')$ est équivalent à la formule :

$\forall v : x = v \Rightarrow \forall\bigcirc P(v, x)$

qui nécessite une quantification sur une variable.

Comparaison CTL vs. LTL ♪♪♪

Contrairement à CTL, les opérateurs temporels LTL parlent tous de la même trace. Les combinaisons de connecteurs temporels ont parfois des sens (subtilement) différents.

	CTL	LTL
$\forall P$, nécessairement P ou $\neg P$	$\mathcal{S} \models P \vee \mathcal{S} \models \neg P$	$\cancel{\mathcal{S} \models P \vee \mathcal{S} \models \neg P}$
négation	$S \models \neg P \equiv S \not\models P$	$\cancel{S \models \neg P \equiv S \not\models P}$
l'un de P ou Q inévitable	$\cancel{\mathcal{S} \models \forall\Diamond P \vee \forall\Diamond Q}$ $\mathcal{S} \models \forall\Diamond(P \vee Q)$	$\mathcal{S} \models \Diamond P \vee \Diamond Q$
l'un de P ou Q continu	$\cancel{\mathcal{S} \models \forall\Box(P \vee Q)}$ $\cancel{\mathcal{S} \models \forall\Box P \vee \forall\Box Q}$	$\mathcal{S} \models \Box P \vee \Box Q$
$\neg P$ transitoire	$\cancel{\mathcal{S} \models \forall\Diamond\forall\Box P}$	$\mathcal{S} \models \Diamond\Box P$
possibilité	$S \models \exists\Diamond P$	$\cancel{S \models \Diamond P}$

Conséquence : l'équité n'est pas exprimable en CTL.

Néanmoins, on peut vérifier des propriétés CTL sur un ST comportant des contraintes d'équité.

Comparaison LTL vs. CTL

Linear Time Logic

+ Intuitive
– ...sauf la négation
+ Suffisante pour décrire un système de transition
+ y compris l'équité
– Vérification exponentielle en le nombre d'opérateurs temporels

Computational Tree Logic

– Expressivité parfois déroutante
+ Propriétés de possibilité (p.e. réinitialisabilité)
+ Suffisante pour décrire un système de transition
– ...sauf l'équité non exprimable (mais utilisable)
+ Vérification linéaire en le nombre d'opérateurs temporels

Au-delà : CTL*

CTL* autorise tout mélange des quantificateurs de traces $\forall, \exists$ et d'états $\square, \Diamond, \bigcirc, \mathcal{U}$.

Exemple : $\exists((\square\Diamond P) \wedge (\Diamond Q))$ = il existe une exécution où P est infiniment souvent vrai, et où Q sera vrai.

CTL* est strictement plus expressif que CTL et LTL. L'usage pratique est rare (hors les fragments correspondant à CTL et LTL).

Septième partie

Vérification de propriétés

Objectif

Vérifier si un système possède une propriété donnée : $\mathcal{M} \models P$.
Si le système et la propriété sont dans le même formalisme (cas de TLA$^+$), cela revient à vérifier si $\mathcal{M} \Rightarrow P$.

Plan

Principe

Le principe de la vérification est le parcours systématique et éventuellement exhaustif de l'espace d'états du système, afin de vérifier la compatibilité avec les propriétés attendues.
$\rightarrow$ système de transitions fini.

- Génération en avant : depuis les états initiaux, pour construire le graphe d'états et vérifier tout type de propriété temporelle (cas de TLA$^+$)
- Génération en arrière :
  - Depuis un (ensemble d')état(s) dont on veut vérifier l'accessibilité depuis un état initial
  - depuis les états interdits (illégaux), pour vérifier qu'ils sont inaccessibles depuis un état initial (propriété de sûreté)

Analyse du problème

Pour construire le graphe d'états, on a besoin de :

- mémoriser les états déjà visités ($\mathcal{A}$*nciens*) ;
- mémoriser les nouveaux états à explorer ($\mathcal{N}$*ouveaux*) : états successeurs ou prédécesseurs des états visités (selon que l'on construit le graphe en avant depuis les états initiaux, ou en arrière depuis des états dit terminaux, par exemple les états interdits).

L'ensemble des états visités finit par devenir beaucoup plus gros que l'ensemble des états à explorer, par simple accumulation : les ensembles $\mathcal{A}$ et $\mathcal{N}$ n'auront pas nécessairement la même représentation mémoire pour des raisons d'efficacité.

Génération de modèle – TLA$^+$

Système $=$ *Init* $\wedge \Box$[*Next*]

```
procedure MC(Init, Next)
begin
  A := {};   -- états explorés
  N := Init; -- états nouveaux, à explorer
  while (N ≠ {})
  begin
   state  := pop(N);       -- choix d'un état à explorer
   succ := state ∧ Next; -- états successeurs
   A := A ∪ {state};      -- state a été traité
   N := N ∪ (succ \ A);  -- images nouvelles à explorer
  end;
  return A;               -- tout a été exploré
end
```


Algorithme générique

```
procedure exploration(Init, Image, Pop, Push, Minus, Union)
begin
  A := {}; -- états explorés
  N := Init; -- états non explorés
  while (N ≠ {})
  begin
   choix  := Pop(N);               -- choix d'état(s) à explorer
   images := Image(choix) Minus A; -- Images non déjà explorées
   A := A Union choix;             -- choix a été traité
   N := Push(N, images);           -- ses images sont à explorer
  end;
  return A;                        -- tout a été exploré
end
```


Prérequis pour l'efficacité

États déjà visités

- Représentation concise en mémoire (très gros ensemble).
- Opérations pour l'ajout d'états nouveaux (`Union`).

États nouveaux

- Opération pour la différence ensembliste avec les états déjà visités (`Minus`) ;
- Opération pour le calcul d'image directe ou inverse (`Image`) ;
- Représentation mémoire permettant de construire un ordre global (`Pop` et `Push`).

Quelques solutions possibles

Représentation des états déjà visités

- explicite : bit-state encoding, table de hachage (risque de collision).
- implicite ou symbolique : Binary Decision Diagrams, formules SAT, polyhèdres. . .
- mixte : explicite + fingerprint (TLA$^+$)

Ordre global sur les états nouveaux

- parcours en profondeur (occupation mémoire limitée)
- parcours en largeur (petits contre-exemples)

Plan

Syntaxe alternative (rappel)

Syntaxe alternative

A (all)	$\leftrightarrow$	$\forall$
E (exists)	$\leftrightarrow$	$\exists$
G (globally)	$\leftrightarrow$	$\Box$
F (finally)	$\leftrightarrow$	$\Diamond$
X (next)	$\leftrightarrow$	$\bigcirc$
A(f U g) ou f AU g	$\leftrightarrow$	$f \forall \mathcal{U} g$
E(f U g) ou f EU g	$\leftrightarrow$	$f \exists \mathcal{U} g$

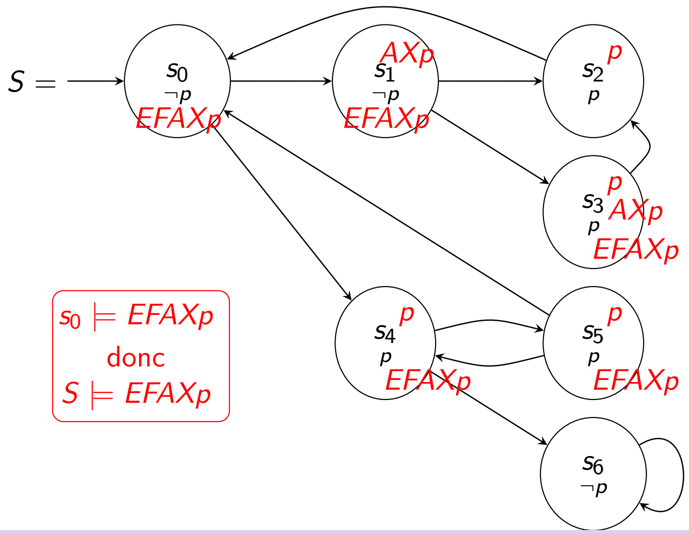
Par exemple : AG EF f $\leftrightarrow$ $\forall\Box\exists\Diamond f$

Principe

- Les modèles en CTL sont les états (et non les exécutions).
- Calcul de l'ensemble des états qui satisfont une formule donnée, de façon inductive sur la structure de la formule.
- On procède par marquage des états : $s \mapsto \{F \mid s \models F\}$ (plutôt que $F \mapsto \{s \mid s \models F\}$).
- Chaque opérateur d'une formule F peut amener un parcours complet de S.

Marquage : un exemple

$EF\ AX\ p$ = existence d'un état dont tous les successeurs vérifient p.



$s_0 \models EFAXp$
donc
$S \models EFAXp$

marquer p

marquer AXp

marquer $EFAXp$

en remontant

les prédécesseurs

Marquage CTL : $mark(F)$

- Si $F = p$ prédicat d'état, marquer les états qui vérifient le prédicat :

  ```
  pourtout s ∈ S : s.F := eval(s,p)
  ```

  (où eval(s,p) permet d'évaluer un prédicat dans un état)
- Si $F = \neg F'$, marquer les états non marqués par F' :

  ```
  mark(F');
  pourtout s ∈ S : s.F := not s.F';
  ```
- Si $F = F_1 \land F_2$, marquer les états marqués par les deux formules :

  ```
  mark(F1);
  mark(F2);
  pourtout s ∈ S : s.F := s.F1 and s.F2;
  ```


Marquage CTL : $mark(F)$

- Si $F = EX\ H$, marquer les états qui peuvent atteindre H en une transition :

  ```
  mark(H);
  pourtout s ∈ S : s.F := false;
  pourtout s ∈ S, pourtout t ∈ succ(s) :
      si t.H alors s.F := true;
  ```
- Si $F = AX\ H$, marquer les états qui atteignent H par toutes les transitions :

  ```
  mark(H);
  pourtout s ∈ S : s.F := false;
  pourtout s ∈ S :
      si pourtout t ∈ succ(s), t.H alors
          s.F := true;
  ```


Marquage CTL : $mark(F)$

- Si $F = H\ EU\ K$, idée : $H\ EU\ K \equiv K \lor (H \land EX(H\ EU\ K))$

```
mark(H);
mark(K);
L := ∅;
pourtout s ∈ S :
    s.F := s.K;
    si s.F alors L := L ∪ {s};
tantque L≠∅ faire:
    s := choose any in L;  // ordre sans importance
    L := L \ {s};
    pourtout t ∈ pred(s) :
        si t.H et ¬t.F alors   // pas déjà marqué
            t.F = true;
            L := L ∪ {t};
        finsi
    finpour
fintq
```

Marquage CTL : *mark*(F)

- Si $F = H\ AU\ K$, idée : $H\ AU\ K \equiv K \vee (H \wedge AX(H\ AU\ K))$

```
mark(H); mark(K);
L := ∅;
pourtout s ∈ S :
    s.F := s.K; s.nb := card(succ(s));
    si s.F alors L := L ∪ {s};
tantque L≠∅ faire:
    s := choose any in L;  // ordre sans importance
    L := L \ {s};
    pourtout t ∈ pred(s) :
        t.nb := t.nb - 1;
        si t.nb = 0 et t.H et ¬t.F alors
            t.F := true;
            L := L ∪ {t};
        finsi
    finpour
fintq
```


CTL équitable

Fair CTL = CTL + des contraintes d'équité multiple sur les états.

- Contraintes d'équité : un ensemble d'ensemble d'états $F_1, \ldots, F_n$
- Trace équitable : trace qui visite infiniment souvent tous les F_i
- État équitable : état à partir duquel il existe au moins une trace équitable
- Avoir une propriété en CTL équitable :

$$M \models_f \phi \triangleq M \models \mathit{fair} \Rightarrow \phi \text{ avec } \mathit{fair} = \bigwedge_{1 \le i \le n} \Box\Diamond F_i$$

Ce n'est pas une formule de CTL (ni de LTL) !

CTL équitable – définition

Pour un état s, $s \models_f \phi$ signifie que s vérifie ϕ avec les contraintes d'équité :

- $s \models_f EX\phi$ s'il existe une trace équitable $s_0 \cdot s_1 \cdots$ avec $s_0 = s$ et tel que $s_1 \models_f \phi$.
- $s \models_f AX\phi$ si pour toute trace équitable $s_0 \cdot s_1 \cdots$ avec $s_0 = s$, $s_1 \models_f \phi$.
- $s \models_f EG\phi$ s'il existe une trace équitable $s_0 \cdot s_1 \cdots$ avec $s_0 = s$ et tel que $\forall i, s_i \models_f \phi$.
- $s \models_f AG\phi$ si pour toute trace équitable $s_0 \cdot s_1 \cdots$ avec $s_0 = s$, $\forall i, s_i \models_f \phi$.
- $s \models_f \phi EU\psi$ s'il existe une trace équitable $s_0 \cdots s_k \cdots$ avec $s_0 = s$ et tel que $s_k \models_f \psi$ et $\forall 0 \le i < k, s_i \models_f \phi$
- $s \models_f \phi AU\psi$ si pour toute trace équitable $\exists k, s_0 \cdots s_k \cdots$ avec $s_0 = s$, et $s_k \models_f \psi$ et $\forall 0 \le i < k, s_i \models_f \phi$

Réduction à CTL classique

On suppose qu'on a marqué les états avec un prédicat *fair* qui dit qu'il existe une trace équitable issue de l'état.

- $s \models_f EX\phi$ ssi $s \models EX(\phi \wedge \mathit{fair})$
- $s \models_f AX\phi$ ssi $s \models AX(\mathit{fair} \Rightarrow \phi)$
- $s \models_f EG\phi$ est compliqué
- $s \models_f AG\phi$ ssi $s \models AG(\mathit{fair} \Rightarrow \phi)$
- $s \models_f \phi EU\psi$ ssi $s \models \phi EU(\mathit{fair} \wedge \psi)$
- $s \models_f \phi AU\psi$ ssi $s \models_f \neg EG\neg\psi$ et $s \models_f \neg((\neg\psi)EU(\neg\phi \wedge \neg\psi))$

Calcul des états équitables (*fair*)

Soit un graphe d'états S et un ensemble de contraintes d'équité F_i.

1. Calculer $SCC(S)$, l'ensemble des *strongly connected components* (composantes fortement connexes) du graphe S. (= les sous-graphes où l'on peut rester boucler à l'infini)
2. Calculer l'union des SCC qui intersectent tous les F_i :
$$S' = \bigcup_{C \in SCC(S)} (C \text{ si } \forall i : C \cap F_i \neq \emptyset)$$
(= le sous-graphe où l'on peut boucler en visitant infiniment tous les F_i)
3. États équitables : $S'' = \{u \in S : u \to^* v \text{ où } v \in S'\}$ (fermeture réflexive transitive des prédécesseurs des états de $S' = S'$ et les états qui peuvent y conduire)
4. Chaque état de S'' est alors marqué *fair*.

Vérification de $S \models_f EG\phi$

On vérifie l'existence d'une trace vérifiant ϕ et conduisant à une (ou plusieurs) composante fortement connexe où ϕ est toujours vrai :

- Marquer les états avec ϕ
- Soit $S(\phi)$ la restriction du système aux états vérifiant ϕ
- Calculer $SCC(S(\phi))$ (les sous-graphes où l'on boucle infiniment en conservant ϕ)
- Calculer S_f l'union des SCC qui intersectent tous les F_i (= le sous-graphe vérifiant toujours ϕ et où l'on peut visiter infiniment souvent tous les F_i)
- $s \models_f EG\phi$ ssi $s \models \phi\ EU\ S_f$ (un préfixe vérifiant ϕ, suivi d'un suffixe infini vérifiant ϕ et visitant infiniment les F_i)

Complexité

- CTL classique : complexité linéaire en le nombre d'opérateurs $= O(|S| \times |\phi|)$.
- CTL équitable :
  - Calcul des SCC linéaire (algorithme de Tarjan)
  - Calcul d'accessibilité linéaire
  - Transformation linéaire en CTL classique
  - Complexité $= O(|S| \times |\phi|)$.

(rappel : on peut ramener l'équité sur les transitions WF(A) et SF(A) à l'équité multiple sur les états)

Plan

Réduction du problème

- Les traces sont en nombre infini en général.
- On se ramène à un problème de traces ω-régulières : un préfixe puis un cycle (un lasso).

Traces ω-régulières

Soit $\mathcal{S} = \langle S, I, R \rangle$ un système de transition fini, L une formule LTL

$$\begin{aligned} & \exists \sigma \in S^\omega : \sigma \in Exec(\mathcal{S}) \wedge \sigma \models L \\ \Leftrightarrow \quad & \exists \sigma_p, \sigma_c \in S^\star : \langle \sigma_p \to {\sigma_c}^\omega \rangle \in Exec(\mathcal{S}) \wedge \langle \sigma_p \to {\sigma_c}^\omega \rangle \models L \end{aligned}$$

(sur un système ayant un nombre fini d'états, la seule solution pour obtenir une exécution infinie est de boucler sur un cycle)

Vérification de modèles
Vérification par preuve
Construction de l'espace d'états
CTL logique temporelle arborescente
LTL logique temporelle linéaire

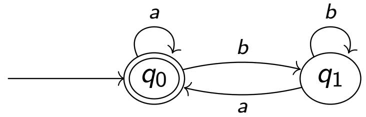
Automate de Büchi

Automate de Büchi

$A = (Q, \Sigma, \delta, q_0, F)$, où

- Q : ensemble fini d'états
- Σ : alphabet fini
- $q_0 \in Q$: l'état initial de l'automate
- $F \subseteq Q$: les états acceptants
- $\delta \in Q \times \Sigma \mapsto Q$: fonction de transition de l'automate.

Un mot infini est accepté si sa reconnaissance visite infiniment F.



reconnaît les mots infinis sur $\{a, b\}$ ayant une infinité de a.

Vérification de modèles
Vérification par preuve
Construction de l'espace d'états
CTL logique temporelle arborescente
LTL logique temporelle linéaire

Intérêt des automates de Büchi

Ensemble infini de traces

- Permet de représenter un ensemble infini de traces ω-régulières
- Suffisant pour représenter tout système de transition fini
- Suffisant pour représenter toute formule de LTL

Manipulation d'ensembles infinis de traces

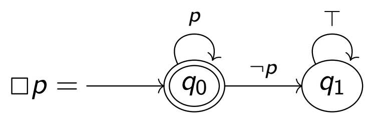
- Opérations faciles (polynomial en le nombre d'états) : $\cup$, $\cap$, concaténation
- Opération facile : tester le vide (aucun mot accepté)
- Opération coûteuse (exponentielle) : le complémentaire

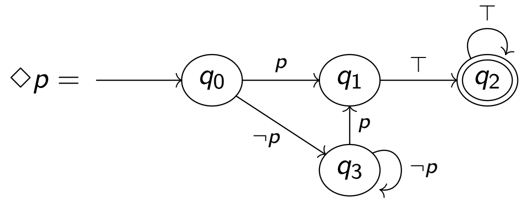
(Note : les automates de Büchi non déterministes sont strictement plus expressifs que les automates de Büchi déterministes. Par exemple : $\{a, b\}^* b^\omega$ n'est pas reconnaissable avec un automate déterministe.)

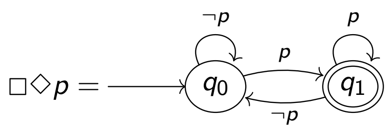
Vérification de modèles
Vérification par preuve
Construction de l'espace d'états
CTL logique temporelle arborescente
LTL logique temporelle linéaire

Transformation de φ en automate

On peut transformer toute formule LTL en un automate de Büchi reconnaissant le même langage (le même ensemble de traces).







Principe de la vérification LTL

Vérifier $\mathcal{M} \models \varphi$

1. Transformer $\mathcal{M}$ en un automate $B_{\mathcal{M}}$ (trivial)
2. Transformer φ en $B_{\neg\varphi}$
3. Construire $B_{\otimes}$ reconnaissant $L(B_{\mathcal{M}}) \cap L(B_{\neg\varphi})$
4. Tester si $L(B_{\otimes})$ est vide : si oui alors $M \models \varphi$; si non, on a un contre-exemple

Complexité en temps : $O(|M| \times 2^{|\varphi|})$
Complexité en espace : **PSPACE**-complet

Équité

L'équité est une formule de LTL (p.e. $\Box\Diamond F$ ou $\Diamond\Box(\text{ENABLED } A) \Rightarrow \Box\Diamond A$) $\rightarrow$ directement pris en compte

Plan

TLA$^+$ Proof System (TLAPS)

TLAPS

- Un assistant de preuve pour TLA$^+$
- Écriture manuelle de la preuve selon une structure hiérarchique
- Chaque étape est mécaniquement vérifiée, soit directement, soit par des démonstrateurs externes :
  - SMT (logique du premier ordre, arithmétique)
  - Zenon (logique du premier ordre)
  - Isabelle (théorie des ensembles, induction, second ordre)
  - PTL (propositional temporal logic)
- Obtention d'un certificat formel vérifiable

Exemple : `xyplus1` (spécification)

MODULE *xyplus1*

EXTENDS *Naturals*, *FiniteSetTheorems*, *TLAPS*
VARIABLES x, y

$TypeInv \triangleq (x \in Nat \land y \in Nat)$

$Ecart \triangleq (0 \leq x - y \land x - y \leq 1)$

$Init \triangleq x = 0 \land y = 0$

$Act1 \triangleq x' = y + 1 \land \text{UNCHANGED } \langle y \rangle$

$Act2 \triangleq y' = x \land \text{UNCHANGED } \langle x \rangle$

$Spec \triangleq Init \land \Box[Act1 \lor Act2]_{\langle x, y \rangle}$

Exemple : `xyplus1` (preuves)

MODULE *xyplus1*

THEOREM $TypeInvOK \triangleq Spec \Rightarrow \Box TypeInv$
$\langle 1 \rangle 1$ $Init \Rightarrow TypeInv$ BY DEF $Init$, $TypeInv$
$\langle 1 \rangle 2$ $(Act1 \lor Act2) \land TypeInv \Rightarrow TypeInv'$ BY DEF $Act1$, $Act2$, $TypeInv$
$\langle 1 \rangle 3$ UNCHANGED $\langle x, y \rangle \land TypeInv \Rightarrow TypeInv'$ BY DEF $TypeInv$
$\langle 1 \rangle 4$ QED BY PTL, $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, $\langle 1 \rangle 3$ DEF $TypeInv$, $Spec$

THEOREM $EcartOK \triangleq Spec \Rightarrow \Box Ecart$
$\langle 1 \rangle 1$ $Init \Rightarrow Ecart$ BY DEF $Init$, $Ecart$
$\langle 1 \rangle 2$ $(Act1 \lor Act2) \land TypeInv \land Ecart \Rightarrow Ecart'$
$\langle 2 \rangle 1$ $Act1 \land TypeInv \land Ecart \Rightarrow Ecart'$ BY DEF $Act1$, $TypeInv$, $Ecart$
$\langle 2 \rangle 2$ $Act2 \land TypeInv \land Ecart \Rightarrow Ecart'$ BY DEF $Act2$, $TypeInv$, $Ecart$
$\langle 2 \rangle 3$ QED BY $\langle 2 \rangle 2$, $\langle 2 \rangle 1$
$\langle 1 \rangle 3$ UNCHANGED $\langle x, y \rangle \land$ $Ecart \Rightarrow Ecart'$ BY DEF $Ecart$
$\langle 1 \rangle 4$ QED BY PTL, $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, $\langle 1 \rangle 3$, $TypeInvOK$ DEF $Ecart$, $Spec$, $TypeInv$

THEOREM $Spec \Rightarrow \Box(TypeInv \land Ecart)$ BY $TypeInvOK$, $EcartOK$

Conclusion

Vérification par preuve

Preuve manuelle assistée :

- Expertise nécessaire à la fois dans le domaine et le vérificateur
- Système infini, de grande taille, paramétré
- De gros progrès récents avec la décharge partielle sur des vérificateurs externes automatiques

Vérification de modèles

- Vérification automatique
- Systèmes finis de petite taille (quelques millions d'états)
- Nécessite la construction de l'espace d'état
- LTL : $O(|M| \times 2^{|\varphi|})$
- CTL : $O(|M| \times |\varphi|)$, mais l'équité complexifie les formules

Pour aller plus loin

- *Symbolic model checking* : représentation d'un ensemble d'états par une formule
  Intéressant quand la spécification est elle-même décrite par des formules (cas de TLA$^+$)
- Model checking à la volée : vérification au fur et à mesure de la construction de l'espace d'état, permet de trouver plus vite un contre-exemple s'il existe.
- *Bounded model checking* : borner la profondeur explorée. Efficace à faible profondeur (dépliage du modèle sous la forme d'une formule SAT) mais ne garantit pas la correction totale.
- Collaboration vérification par preuve $\leftrightarrow$ vérification de modèles

Huitième partie

Conclusion

Résumé

Motivations pour la vérification de logiciels

- Les implantations sont souvent erronées
- Les spécifications sont souvent incomplètes $\Rightarrow$ comportements inattendus
- Systèmes critiques (avionique, médecine...) : les erreurs peuvent avoir des conséquences dramatiques

Vérification de systèmes réels

- Difficile sur l'intégralité
- Envisageable pour certaines propriétés/parties

Fondations pour la vérification

- Logique propositionnelle et logique des prédicats
  - Formules bien formées
  - Sémantique
  - Preuves
- Logique temporelle
  - Temps logique : LTL, CTL
  - Temps réel : automates temporisés

Approches pour la vérification

- Les systèmes de transitions forment la base de la plupart des méthodes de vérification
- Vérification de modèles (*model checking*) :
  - Automatique
  - Expertise nécessaire dans la modélisation, pas dans la vérification
  - Explosion combinatoire du nombre d'états/transitions
- Vérification par preuve :
  - Semi-automatique
  - Expertise nécessaire dans la modélisation et dans la preuve