Systèmes de transitions

Philippe Quéinnec, Xavier Thirioux

Département Informatique et Mathématiques appliquées
ENSEEIHT

4 mars 2014

Méthodes formelles ?

Pourquoi ?

- Nécessité de prouver qu'un programme possède bien les propriétés attendues
- C'est dur ⇒ nécessité de cadres formels précis et d'outils

Comment ?

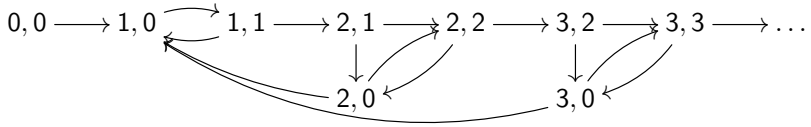
- Langage classique : état = valeurs des variables + *flot de contrôle implicite*
- Système de transitions : état = valeurs des variables. flot de contrôle *explicite*

Exemple

Soit trois processus exécutant concurremment (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) ≈ quelles sont les propriétés attendues
2. Un langage d'actions ≈ un langage de spécification plus opérationnel ≈ un langage de programmation
3. (en fait, langage de spécification = langage d'actions)
4. Cadre formel = système de transitions

Plan du cours

1. Systèmes de transitions
2. TLA+ : les actions
3. Équité dans les systèmes de transitions
4. Logique temporelle linéaire LTL
5. TLA+ : la logique et l'équité
6. Logique temporelle arborescente CTL
7. Vérification de modèles

Première partie

Systèmes de transitions

Introduction

Objectifs

Représenter les exécutions d'un programme 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

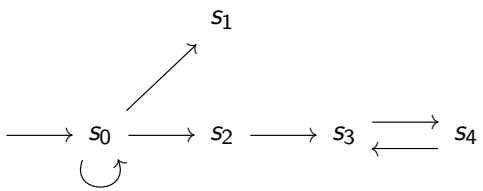
ST

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 : $\langle s_1 \rightarrow s_2 \rightarrow \ldots \rangle$.
- $\langle\rangle$: la séquence vide.
- Pour une séquence finie σ, $\sigma^\star \triangleq$ les séquences finies produites par la répétition arbitraire de σ.
- Pour une séquence finie σ, $\sigma^+ \triangleq \sigma^* \setminus \{\langle\rangle\}$
- Pour une séquence finie σ, $\sigma^\omega \triangleq$ la séquence infinie produite par la répétition infinie de σ.

Traces

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 ssi 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

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

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

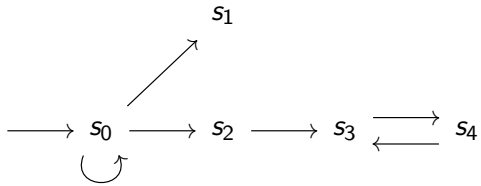
Soit $\mathcal{S} = \langle S, I, R \rangle$ un système de transitions.
Une exécution $\sigma = \langle s_0 \to \ldots \rangle$ est une trace infinie ou finie maximale telle que $s_0 \in I$.
$Exec(\mathcal{S}) \triangleq$ l'ensemble des exécutions de $\mathcal{S}$.

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

Préfixe d'exécution

Pour σ une exécution, un préfixe d'exécution est une trace finie $\sigma_p = \langle s_0 \to \ldots \to s_n \rangle$ telle que $s_0 \in I$ et $\sigma = \sigma_p \to \ldots$.
s_n est appelé l'état final de ce préfixe.

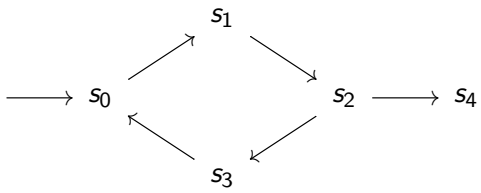
Exemple - traces



$s_0 \to s_0 \to s_2 \to s_3$ 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

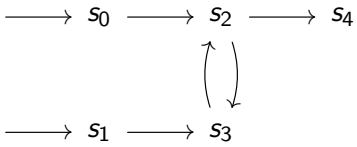


$Traces(s_2) =$

$Traces(s_0) =$

$Exec(\mathcal{S}) =$

Exemple 3 - traces, exécutions



$Traces(s_2) =$

$Traces(s_0) =$

$Traces(s_1) =$

$Exec(\mathcal{S}) =$

États accessibles

État accessible

Soit $\mathcal{S} = \langle S, I, R \rangle$ un système de transitions.
$s \in S$ est un état accessible ssi il existe un préfixe d'exécution dont l'état final est s.
$Acc(\mathcal{S}) \triangleq$ l'ensemble des états accessibles de $\mathcal{S}$.

États récurrents

État récurrent

Soit $\mathcal{S} = \langle S, I, R \rangle$ un système de transitions et $\sigma = \langle s_0 \rightarrow \ldots \rangle$ une exécution.

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

$Inf_S(\sigma) \triangleq$ l'ensemble des états récurrents de σ.

Transitions récurrentes

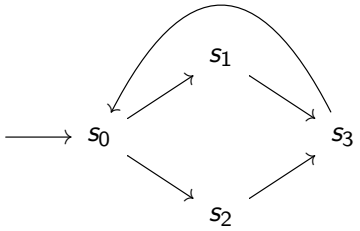
Transition récurrente

Soit $\mathcal{S} = \langle S, I, R \rangle$ un système de transitions et $\sigma = \langle s_0 \to \ldots \rangle$ une exécution.

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

$Inf_T(\sigma) \triangleq$ l'ensemble des transitions récurrentes de 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	$\langle (s_0 \to s_1 \to s_3)^* \to (s_0 \to s_2 \to s_3)^\omega \rangle$

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 : $Etiq \in R \to L$.

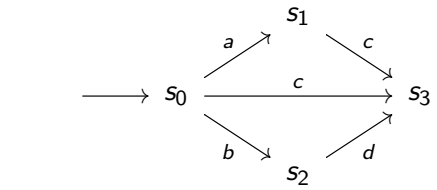
Un ST étiqueté se rapproche beaucoup des automates.
Mais : pas d'état terminal + exécution infinie.

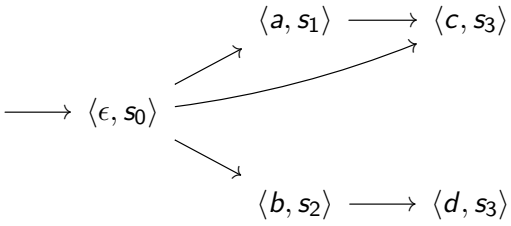
Équivalence aux ST sans étiquette

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 :

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

Exemple - équivalence avec/sans étiquette





Différences avec un automate

Système de transitions ≠ 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

Donnée en extension du graphe des exécutions, par exemple sous forme graphique.
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

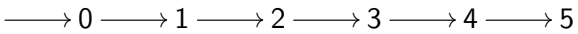
Triplet $\langle V, Init, Trans \rangle$ où

- $V = \{v_i\}_{i \in I}$: ensemble fini de variables.
- $Init(\{v_i\}_{i \in I})$: prédicat définissant les états initiaux et portant sur les variables v_i.
- $Trans(\{v_i, v_i'\}_{i \in I})$: 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;
}
```

Graphe d'exécution pour $N = 5$.

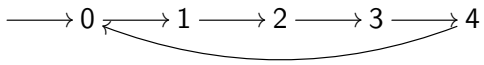


Symboliquement :
$V \triangleq i \in \mathbb{N}$
$I \triangleq i = 0$
$R \triangleq i < N \wedge i' = i + 1$

Exemple : un compteur cyclique

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

Graphe d'exécution pour $N = 5$.

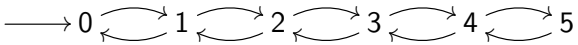


Symboliquement :
$V \triangleq i \in \mathbb{N}$
$I \triangleq i = 0$
$R \triangleq i' = (i + 1) \bmod N$

Exemple : un entier oscillant

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

Graphe d'exécution pour $N = 5$.



Symboliquement :

$V \triangleq i \in \mathbb{N}$

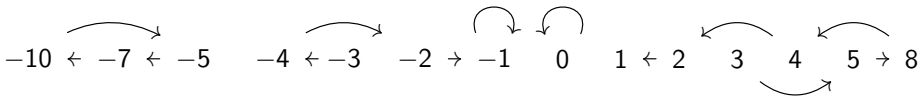
$I \triangleq i = 0$

$R \triangleq i > 0 \wedge i' = i - 1 \quad$ ou $R \triangleq |i' - i| = 1 \wedge 0 \leq i' \leq N$
$\vee\ i < N \wedge i' = i + 1$

Exemple : le problème de Collatz

```
void Collatz(int n) {
  while (n != 1)
    if (n mod 2 = 0) n = n / 2; else n = (3*n + 1) / 2;
}
```

Graphe d'exécution pour $-5 \leq n \leq 5$ initialement.



Symboliquement :

$V \triangleq n \in \mathbb{N}$

$I \triangleq -5 \leq n \leq 5$

$R \triangleq n \neq 1 \ \wedge \begin{pmatrix} (n' = n/2 \wedge n \equiv 0[2]) \\ \vee (n' = (3n+1)/2 \wedge n \equiv 1[2]) \end{pmatrix}$

ST correspondant

Le système de transitions correspondant est $\langle S, I, R \rangle$ où :

- $S = \Pi_{i \in I} D_i$, où $\{D_i\}_{i \in I}$ sont les domaines (types) de $\{v_i\}_{i \in I}$
- $I = \mathit{Init}(\{v_i\}_{i \in I})$
- $R = \mathit{Trans}(\{v_i, v_i'\}_{i \in I})$

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

$$\begin{array}{lcl} V & \triangleq & n \in \mathbb{N} \\ I & \triangleq & -5 \leq n \leq 5 \\ R & \triangleq & n \neq 1 \ \wedge \ \left(\begin{array}{l} (n' = n/2 \wedge n \equiv 0[2]) \\ \vee (n' = (3n+1)/2 \wedge n \equiv 1[2]) \end{array} \right) \end{array}$$

Prédicats d'état : I, $n < 20$
Prédicats de transition : R, $n' - n > 3$

Plan

Déterminisme

Déterminisme fort

Un système est déterministe fort $\triangleq$ la relation de transition est fonctionnelle, i.e. $\forall s \in S : |\{s' \in S : (s, s') \in R\}| \leq 1$.

Déterminisme opérationnel

Un système est déterministe au sens opérationnel $\triangleq$ la relation de transition *restreinte aux états accessibles* est fonctionnelle.

Note : ST étiqueté déterministe $\neq$ automate déterministe
(les étiquettes ne comptent pas)

Déterminisme – exemple

$$\begin{array}{lcl} V & \triangleq & n \in \mathbb{N} \\ I & \triangleq & n = 0 \\ R & \triangleq & \begin{array}{l} n \leq 4 \ \wedge \ n' = n + 1 \\ \vee \ n\%3 = 0 \ \wedge \ n' - n = 1 \\ \vee \ n \geq 6 \ \wedge \ n' = n + 2 \end{array} \end{array}$$

Graphe d'exécution : $\longrightarrow 0 \rightarrow 1 \longrightarrow 2 \longrightarrow 3 \rightarrow 4 \quad 5 \quad 6 \longrightarrow 7 \ldots$
$\searrow 8 \ldots$

Déterministe opérationnellement, mais pas fortement déterministe.

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.

Boucle

Un système possède une boucle (livelock) $\triangleq$ il existe un état accessible dont le seul successeur par la relation R est lui-même.

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 d'actions pour revenir à l'état initial du système et ainsi redémarrer.

Bégaiement

Bégaiement

Un système de transitions bégaie ssi tout état possède une boucle vers lui-même : $R \supseteq Id$.

Utilité :

- Modéliser l'avancement arbitraire :  *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) $\ldots \to s_i$ est alors $\ldots \to {s_i}^{\omega}$.
- Composer plusieurs systèmes de transitions.

Composition de systèmes de transitions

Composition

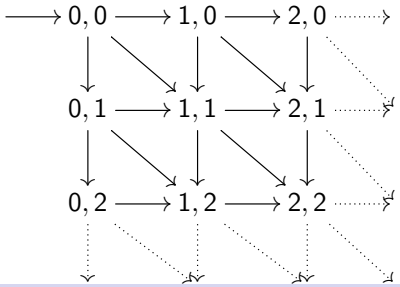
La composition des ST avec bégaiement $\langle V_1, I_1, R_1 \rangle$ et $\langle V_2, I_2, R_2 \rangle$ est $\langle V, I, R \rangle$ où :

- $V \triangleq V_1 \cup V_2$ (union des variables)
- $I \triangleq I_1 \wedge I_2$ (les deux sous-systèmes démarrent dans un état initial respectif)
- $R \triangleq R_1 \wedge R_2$ (les deux sous-systèmes évoluent chacun selon ses transitions)

Comme R_1 et R_2 peuvent bégayer, $R_1 \wedge R_2$ signifie donc qu'on peut exécuter une transition de R_1 seule et R_2 bégayant, ou bien réciproquement, ou bien encore exécuter R_1 en même temps que R_2.

Exemple - composition

$$\left(\begin{array}{l} V_1 \triangleq i \in \mathbb{N} \\ I_1 \triangleq i = 0 \\ R_1 \triangleq i' = i + 1 \\ \qquad \vee\ i' = i \end{array}\right) \otimes \left(\begin{array}{l} V_2 \triangleq j \in \mathbb{N} \\ I_2 \triangleq j = 0 \\ R_2 \triangleq j' = j + 1 \\ \qquad \vee\ j' = j \end{array}\right) \rightarrow \left(\begin{array}{l} V \triangleq i, j \in \mathbb{N} \\ I \triangleq i = 0 \wedge j = 0 \\ R \triangleq \quad i' = i + 1 \wedge j' = j \\ \qquad \vee i' = i \wedge j' = j + 1 \\ \qquad \vee i' = i + 1 \wedge j' = j + 1 \\ \qquad \vee i' = i \wedge j' = j \end{array}\right)$$



Deuxième partie

TLA+ – les actions

Plan

Structure d'un programme

Un « programme » = une spécification de ST =

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

Exemple

MODULE *exemple*1

EXTENDS *Naturals*

VARIABLE x

États initiaux

$Init \triangleq 0 \leq x \wedge x < 3$

Actions

$Plus \triangleq x' = x + 1$

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

$Next \triangleq Plus \vee Moins$

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

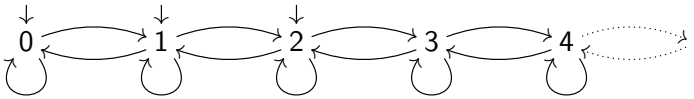
Exemple

Correspond au système de transitions :

$V \triangleq x \in \mathbb{N}$

$I \triangleq 0 \leq x < 3$

$R \triangleq \quad x' = x + 1$
$\quad \vee x > 0 \wedge x' = x - 1$
$\quad \vee x' = x$



Constantes

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

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 : $+$, $-$, $> \ldots$
- 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.

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

Action non déterministe : $x' > x$ ou $x' \in \{x+1, x+2, x+3\}$

Action non évaluable : $x' \in \{y : \exists z : z * y = x \wedge z\%2 = 0\}$

Action multiple : $x' = y \wedge y' = x$

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.

$$
\begin{aligned}
& x < 10 \wedge x' = x + 1 \\
\text{plutôt que}\quad & x' = x + 1 \wedge x' < 11 \\
\text{ou}\quad & x' - x = 1 \wedge x' < 11
\end{aligned}
$$

Bégaiement

Bégaiement

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

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

Non bégaiement

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

Variables non contraintes

$$
\begin{aligned}
(x' = x + 1) \quad &= \quad (x' = x + 1 \wedge y' = \text{n'importe quoi}) \\
&\neq \quad (x' = x + 1 \wedge y' = y)
\end{aligned}
$$

UNCHANGED

UNCHANGED $e \triangleq e' = e$

MODULE *AlternatingBit*

EXTENDS *Naturals*
CONSTANT *Data*
VARIABLES *val*, *ready*, *ack*

$$
\begin{aligned}
\mathit{Init} &\triangleq \land\ \mathit{val} \in \mathit{Data} \\
&\quad\ \land\ \mathit{ready} \in \{0, 1\} \\
&\quad\ \land\ \mathit{ack} = \mathit{ready} \\
\mathit{Send} &\triangleq \land\ \mathit{ready} = \mathit{ack} \\
&\quad\ \land\ \mathit{val}' \in \mathit{Data} \\
&\quad\ \land\ \mathit{ready}' = 1 - \mathit{ready} \\
&\quad\ \land\ \text{UNCHANGED}\ \mathit{ack} \\
\mathit{Reveive} &\triangleq \land\ \mathit{ready} \neq \mathit{ack} \\
&\quad\ \land\ \mathit{ack}' = 1 - \mathit{ack} \\
&\quad\ \land\ \text{UNCHANGED}\ \langle \mathit{val}, \mathit{ready} \rangle \\
\mathit{Next} &\triangleq \mathit{Send} \lor \mathit{Reveive} \\
\mathit{Spec} &\triangleq \mathit{Init} \land \Box[\mathit{Next}]_{\langle \mathit{val}, \mathit{ready}, \mathit{ack} \rangle}
\end{aligned}
$$

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 ⇒ la "fonction change".

Définition

Définition d'un symbole

$f[x \in \mathit{Nat}] \triangleq$ expression utilisant x

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

Définition d'une valeur

$[x \in S \mapsto \mathit{expr}]$

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

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

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

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

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

EXCEPT

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

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

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, il est possible de donner une définition récursive :

$$fact[n \in Nat] \triangleq \text{IF } n = 0 \text{ THEN } 1 \text{ ELSE } n * fact[n - 1]$$

En théorie, il faudrait démontrer la validité de cette définition (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

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

Plan

Définition de symbole local

LET

Expression : LET $v \triangleq e$ IN f

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

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

Choix déterministe

Opérateur de choix

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

CHOOSE $x : p =$ CHOOSE $x : p$ (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

- Le programme

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

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

- Le programme

$$(x \in Nat) \wedge \Box[x' \in Nat]_x$$

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

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.

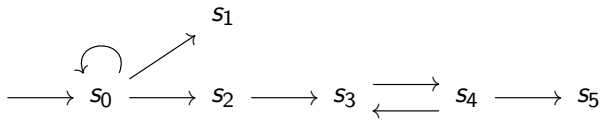
Plan

Équité simple sur les états

On se donne $F \subseteq S$ un ensemble d'états équitables.
Alors toute exécution σ doit être telle que $Inf_S(\sigma) \cap F \neq \emptyset$.

σ contient au moins un état récurrent appartenant à F.

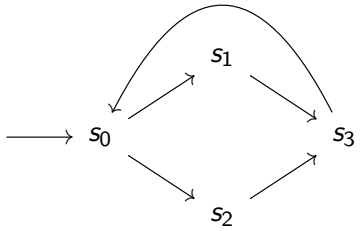
Exemple - équité simple



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

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

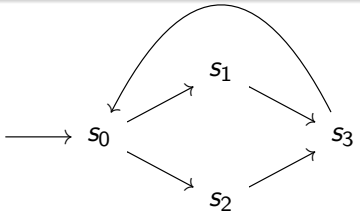
Exemple - équité simple



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

légale	$\langle (s_0 \rightarrow s_1 \rightarrow s_3)^{\omega} \rangle$
légale	$\langle (s_0 \rightarrow s_1 \rightarrow s_3 \rightarrow s_0 \rightarrow s_2 \rightarrow s_3)^{\omega} \rangle$
illégale	$\langle (s_0 \rightarrow s_1 \rightarrow s_3)^{\star} \rightarrow (s_0 \rightarrow s_2 \rightarrow s_3)^{\omega} \rangle$
légale	$\langle (s_0 \rightarrow s_1 \rightarrow s_3 \rightarrow (s_0 \rightarrow s_2 \rightarrow s_3)^{\star})^{\omega} \rangle$

Exemple - équité simple



$Exec(S) \quad =$

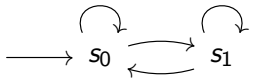
Équité simple	
$\{s_2\}$	
$\{s_1, s_2\}$	

Équité multiple sur les états

On se donne un ensemble dénombrable, donc indexable par un ensemble de nombres entiers J, d'ensembles équitables $\{F_i\}_{i \in J}$. Toute exécution σ doit être telle que $\forall i \in J : Inf_S(\sigma) \cap F_i \neq \emptyset$.

Exécutions vérifiant l'équité multiple = intersection des exécutions vérifiant l'équité simple sur chacun des F_i.

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

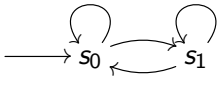
Cas simple : J est fini.

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

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

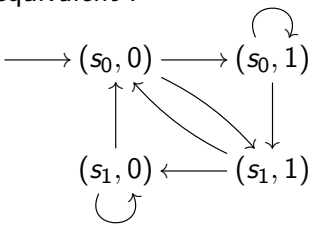
Le premier ensemble de R' est pour le cas où on visite un état de F_j et on cherche donc à visiter l'ensemble suivant ; le deuxième ensemble est pour le cas où on n'est pas en train de visiter un état de F_j, que l'on continue à attendre.

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 ↔ simple

Cas général (J potentiellement infini).

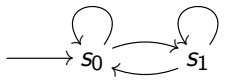
Le système $\langle S', I', R' \rangle$ à équité simple suivant est équivalent :

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

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

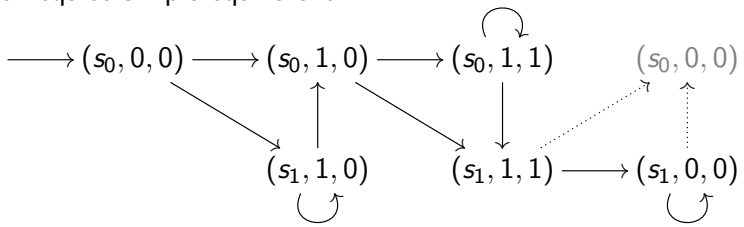
Une exécution typique des compteurs i, j forme un triangle :
$\langle (0,0) \to (1,0) \to (1,1) \to (2,0) \to (2,1) \to (2,2) \to (3,0) \to \ldots \rangle$

Exemple équité multiple



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

ST en équité simple équivalent :

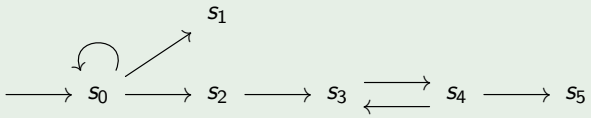


avec équité simple sur $\{(s_0,0,0), (s_0,1,0)\}$

Équité conditionnelle sur les états

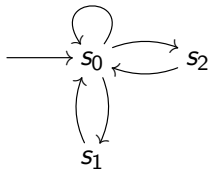
On se donne deux ensembles F et G. Toute exécution σ doit être telle que $Inf_S(\sigma) \cap F \neq \emptyset \;\Rightarrow\; Inf_S(\sigma) \cap G \neq \emptyset$.

S'il existe un état récurrent de σ qui est dans F, alors il doit exister un état récurrent qui est dans G.



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

Exemple - équité conditionnelle



$Exec(S) \;=$

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

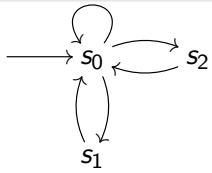
Équivalence équité conditionnelle ↔ simple

On étudie le système $\langle S', I', R' \rangle$ tel que :

- $S' = (S \times \{0\}) \cup ((S \setminus F) \times \{1\})$
- $I' = I \times \{0\}$
- $R' = \{(\langle s, i \rangle, \langle s', j \rangle) \mid (s, s') \in R \wedge 1 \geq j \geq i \geq 0\} \cap (S' \times S')$
- Équité simple $F' = (G \times \{0\}) \cup ((S \setminus F) \times \{1\})$

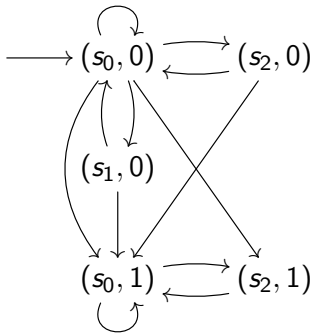
Les états $\langle s, 0 \rangle$ correspondent aux exécutions où G doit être infiniment souvent visité, alors que les états $\langle s, 1 \rangle$ correspondent aux exécutions où F ne doit plus être visité du tout.

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 faible sur les transitions

Soit $F \subseteq R$ un sous-ensemble des transitions. F est dit faiblement équitable ssi dans toute exécution :

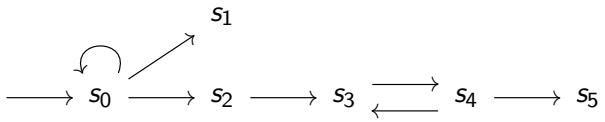
- l'ensemble d'états $S \setminus dom(F)$ est récurrent,
- ou l'ensemble de transitions F est récurrent.

Ou, de manière équivalente,

- 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.

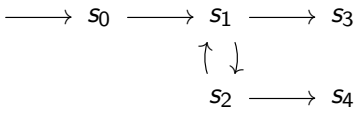
Exemple - équité faible



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

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

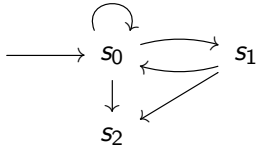
Exemple - équité faible



$$Exec(S) =$$

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

Exemple - équité faible



$$Exec(S) =$$

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

Équivalence équité faible $\leftrightarrow$ équité simple sur les états

On étudie le système $\langle S', I', R' \rangle$ tel que :

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

Soit $F \subseteq R$ un sous-ensemble des transitions. F est dit fortement équitable ssi dans toute exécution,

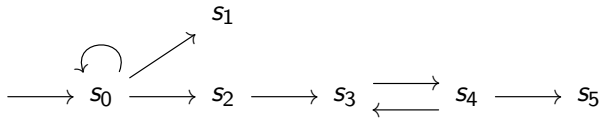
- l'ensemble d'états $dom(F)$ n'est pas récurrent,
- ou l'ensemble de transitions F est récurrent.

Ou, de manière équivalente,

- 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 état où une transition de r est exécutable, alors une transition (s, s') de r finit par être exécutée.

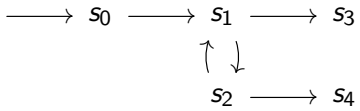
Exemple - équité forte



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

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

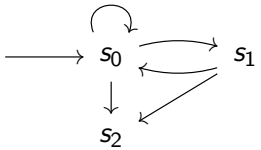
Exemple - équité forte



$Exec(S) =$

Équité forte	
$\{(s_2, s_4)\}$	

Exemple - équité forte



$Exec(S) =$

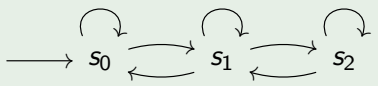
Équité forte	
$\{(s_0, s_1)\}$ $\{(s_1, s_2)\}$ $\{(s_0, s_1), (s_1, s_2)\}$	

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 définitif)
Équité faible sur $\{(s_1, s_2)\}$ (idem)
Équité faible sur $\{(s_2, s_1)\}$ (idem)
Équité forte sur $\{(s_1, s_2)\}$ (ici, équivalent à équité simple sur $\{s_2\}$)

Équivalence équité forte $\leftrightarrow$ équité simple sur les états

On étudie le système $\langle S', I', R' \rangle$ tel que :

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

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 $Etiq^{-1}(F)$.

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.

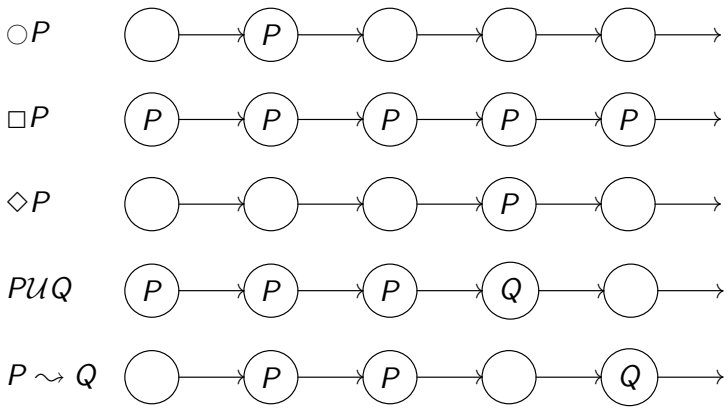
Note : plutôt que d'*état*, on parle souvent d'*instant*.

Syntaxe de la LTL

formule	nom	interprétation
$\bigcirc P$	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 finit par être vrai (dans le futur)
$P \mathcal{U} Q$	until	Q finit par être vrai, et en attendant P reste vrai
$P \rightsquigarrow Q$	leadsto	quand P est vrai, alors Q l'est plus tard
$\neg P$		
$P \vee Q$		
$P \wedge Q$		
s		l'état courant de l'exécution est s

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 $P \mathcal{U} Q$:

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

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
  $P \mathcal{R} Q \triangleq \neg(\neg P \mathcal{U} \neg Q)$
  Q est toujours vrai, sauf à partir du moment où P est vrai.

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

Guère d'utilité en pratique...

Sémantique (système)

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

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 :

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

Sémantique (opérateurs logiques)

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

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

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

Sémantique (opérateurs temporels)

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

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

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

Sémantique (opérateurs temporels dérivés)

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

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

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

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 $F_{\mathcal{S}pec}$:

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

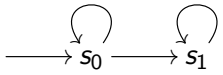
- revient à déterminer la validité de :

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

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).

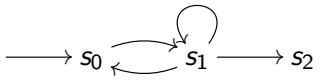
Plan

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)
$\Box\Diamond\neg s_1$			
$\Box(s_1 \Rightarrow \Diamond s_2)$			
$\Diamond\Box(s_1 \vee s_2)$			
$\Box(s_1 \mathcal{U} s_2)$			
$\Box(s_0 \Rightarrow s_0 \mathcal{U} s_1)$			
$\Box(s_0 \mathcal{U}(s_1 \vee s_2))$			
$\Box(s_1 \Rightarrow s_1 \mathcal{U} s_2)$			
$\Diamond(s_1 \mathcal{U} s_2)$			
$\Diamond(s_1 \mathcal{W} s_2)$			
$\Box\Diamond(s_1 \mathcal{U}(s_0 \vee s_2))$			

Invariance, stabilité

Invariance

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

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

Stabilité

Spécifier la stabilité d'une situation si elle survient :

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

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

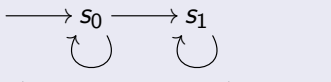
Négation

Négation : danger !

$$\sigma \models \neg P \equiv \sigma \not\models P$$

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

$\mathcal{S} \not\models Q$ signife 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 $\mathcal{S} \not\models Q \wedge \mathcal{S} \not\models \neg Q$:



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

Combinaisons

Infiniment souvent

Spécifier que P est infiniment souvent vrai dans toute exécution :

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

Finalement toujours

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 (par Q) à un requête donnée (par P) :

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

Souvent nommé leads-to :

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

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 – *Fairness*

Équité faible des transitions

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

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

Équité forte des transitions

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

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

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.
- Réinitialisabilité : quelque soit l'état, il est possible de revenir dans un des états initiaux.

Spécification d'un ST

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.

Sûreté/vivacité – *Safety/Liveness*

On qualifie de

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

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

Plan

La logique TLA+

Expressions logiques

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

Pas de $\mathcal{U}$ mais :

$$\Box(p \Rightarrow (p\mathcal{W}q)) = \Box(p \Rightarrow (p' \vee q))$$
$$\Box(p \Rightarrow (p\mathcal{U}q)) = \Box(p \Rightarrow (p' \vee q)) \wedge \Box(p \Rightarrow \Diamond q)$$

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

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

Forme d'un programme 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 programme

Raffinage simple

Un programme (concret) *Pc* raffine un programme (abstrait) *Pa* si $Pc \Rightarrow Pa$: tout ce que fait *Pc* est possible dans *Pa*.

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

Somme plus concrète

MODULE *somme2*

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

$TypeInvariant \triangleq res \in Nat \wedge acc \in Nat \wedge disp \in \mathrm{SUBSET}\ 1 \ldots N$

$Init \triangleq res = 0 \wedge acc = 0 \wedge disp = 1 \ldots N$
$Next \triangleq \vee \exists i \in disp : acc' = acc + i \wedge disp' = disp \setminus \{i\}$
$\qquad \wedge \mathrm{UNCHANGED}\ res$
$\qquad \vee disp = \{\} \wedge res' = acc \wedge \mathrm{UNCHANGED}\ \langle disp, acc \rangle$
$Spec \triangleq Init \wedge \Box[Next]_{\langle res, disp, acc \rangle} \wedge \mathrm{WF}_{\langle res, disp, acc \rangle}(Next)$

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

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

$Init \triangleq res = 0 \land acc = 0 \land i = N$

$Next \triangleq \lor i > 0 \land acc' = acc + i \land i' = i - 1 \land \text{UNCHANGED } res$
$\lor i = 0 \land res' = acc \land \text{UNCHANGED } \langle i, acc \rangle$

$Spec \triangleq Init \land \Box[Next]_{\langle res, i, acc \rangle} \land \text{WF}_{\langle res, i, acc \rangle}(Next)$

Raffinage - exemple

Somme3 raffine somme2

MODULE *somme3_raffine_somme2*

EXTENDS *somme3*

$dispMapping \triangleq 1 \,..\, i$

$Orig \triangleq$ INSTANCE *somme2* WITH $disp \leftarrow dispMapping$

$Raffinement \triangleq Orig\,!\,Spec$

THEOREM $Spec \Rightarrow Orig\,!\,Spec$

Plan

Règles de preuve – simple temporal logic

$$\frac{F \text{ prouvable en logique propositionnelle}}{\Box F}\text{STL1} \qquad \frac{F \Rightarrow G}{\Box F \Rightarrow \Box G}\text{STL4}$$

$$\frac{}{\Box F \Rightarrow F}\text{STL2} \qquad \frac{}{\Box\Box F = \Box F}\text{STL3}$$

$$\frac{}{\Box(F \wedge G) = (\Box F) \wedge (\Box G)}\text{STL5} \qquad \frac{}{\Diamond\Box F \wedge \Diamond\Box G = \Diamond\Box(F \wedge G)}\text{STL6}$$

Règles de preuve – TLA+ invariant

$$\frac{P \wedge (v' = v) \Rightarrow P'}{\Box P = P \wedge \Box[P \Rightarrow P']_v}\text{TLA1} \qquad \frac{P \wedge [\mathcal{A}]_{v_1} \Rightarrow Q \wedge [\mathcal{B}]_{v_2}}{\Box P \wedge \Box[\mathcal{A}]_{v_1} \Rightarrow \Box Q \wedge \Box[\mathcal{B}]_{v_2}}\text{TLA2}$$

$$\frac{I \wedge [\mathcal{N}]_v \Rightarrow I'}{I \wedge \Box[\mathcal{N}]_v \Rightarrow \Box I}\text{INV1} \qquad \frac{}{\Box I \Rightarrow (\Box[\mathcal{N}]_v = \Box[\mathcal{N} \wedge I \wedge I']_v)}\text{INV2}$$

Règles de preuve – TLA+ vivacité

$$\frac{\begin{array}{c} P \wedge [\mathcal{N}]_v \Rightarrow (P' \vee Q') \\ P \wedge \langle \mathcal{N} \wedge \mathcal{A} \rangle_v \Rightarrow Q' \\ P \Rightarrow \text{ENABLED } \langle \mathcal{A} \rangle_v \end{array}}{\Box[\mathcal{N}]_v \wedge WF_v(\mathcal{A}) \Rightarrow (P \rightsquigarrow Q)}\text{WF1}$$

$$\frac{\begin{array}{c} P \wedge [\mathcal{N}]_v \Rightarrow (P' \vee Q') \\ P \wedge \langle \mathcal{N} \wedge \mathcal{A} \rangle_v \Rightarrow Q' \\ \Box P \wedge \Box[\mathcal{N}]_v \wedge \Box F \Rightarrow \Diamond \text{ENABLED } \langle \mathcal{A} \rangle_v \end{array}}{\Box[\mathcal{N}]_v \wedge SF_v(\mathcal{A}) \wedge \Box F \Rightarrow (P \rightsquigarrow Q)}\text{SF1}$$

Règles de preuve dérivées

$$\frac{\Box(P \Rightarrow \Box P) \wedge \Diamond P}{\Diamond\Box P}\text{LDSTBL} \qquad \frac{P \rightsquigarrow Q \wedge Q \rightsquigarrow R}{P \rightsquigarrow R}\text{TRANS}$$

$$\frac{\forall m \in W : (P(m) \rightsquigarrow Q)}{(\exists m \in W : P(m)) \rightsquigarrow Q}\text{INFDIJ}$$

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 $\Diamond P$, $P \rightsquigarrow Q \ldots$: 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 $O(|S| \times 2^{|F|})$.

Le principe est de parcourir l'espace d'états en marche avant, i.e. à partir des états initiaux du système, on visite les états successeurs par R. On arrête le parcours lorsque la formule LTL courante est atomique (i.e. de la forme s_i ou $\neg s_i$) en comparant simplement avec l'état courant, ou bien lorsqu'on retombe sur un état déjà visité.

Vérificateur TLC

Le vérificateur de modèles TLC sait vérifier :

- les programmes avec des actions gardées ;
- (efficacement) les invariants sans variables primées : $\Box P$ où P est prédicat d'état ;
- les formules de sûreté pure avec variables primées : $\Box P$ où P est prédicat de transition ;
- $P \rightsquigarrow Q$ 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 ex.

Sixième partie

CTL – logique temporelle arborescente

Plan

Computational Tree Logic

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

Opérateurs duaux construits à partir du quantificateur $\exists$.

formule	interprétation (pour s un état) pour au moins une trace partant de s
$\exists\Box P$	P est toujours vrai à chaque état
...	...

Opérateurs minimaux

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

- $\exists\bigcirc P \triangleq \neg\forall\bigcirc\neg P$
- $\forall\Diamond P \triangleq \mathit{True}\,\forall\mathcal{U}\,P$
- $\exists\Diamond P \triangleq \mathit{True}\,\exists\mathcal{U}\,P$
- $\forall\Box P \triangleq \neg\exists\Diamond\neg P$
- $\exists\Box P \triangleq \neg\forall\Diamond\neg P$
- Opérateur waiting-for
  $P\,\exists\mathcal{W}\,Q \triangleq \exists\Box P \vee P\,\exists\mathcal{U}\,Q$
  $P\,\forall\mathcal{W}\,Q \triangleq \forall\Box P \vee P\,\forall\mathcal{U}\,Q$
  $\triangleq \neg(\neg Q\,\exists\mathcal{U}(\neg P \wedge \neg Q))$

Syntaxe alternative

On trouve fréquemment une autre syntaxe :

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

$\forall\Box\exists\Diamond P \leftrightarrow$ AG EF P
$f\,\forall\mathcal{U}\,g \leftrightarrow$ A(f U g)

Sémantique (système)

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

Vérification par un système

Un système $\mathcal{S}$ vérifie (valide) la formule F ssi tous les états initiaux de $\mathcal{S}$ la valident :

$$\frac{\forall s \in I : s \models F}{\mathcal{S} \models F}$$

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

Sémantique (opérateurs logiques)

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

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

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

Sémantique (opérateurs temporels)

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

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

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

Sémantique (opérateurs temporels dérivés)

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

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

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

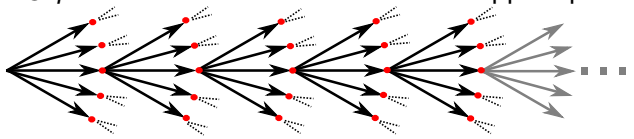
$$\frac{\forall \sigma \in \mathit{Traces}(s) : \exists i \geq 0 : \sigma_i \models P}{s \models \forall \Diamond P}$$

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

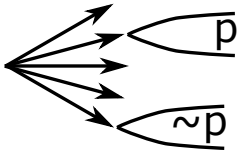
Plan

Exemples amusants

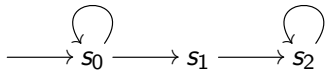
- $\exists \Box\ \forall \bigcirc\ p$: une exécution avec une “enveloppe” qui vérifie p



- $\exists \bigcirc\ \forall \Box p\ \wedge\ \exists \bigcirc\ \forall \Box \neg p$
  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

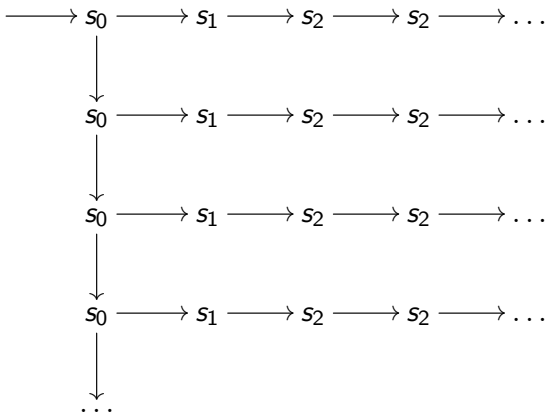


Exemple

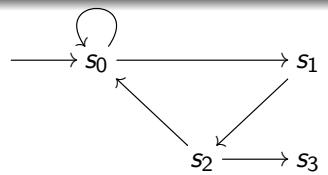


	pas d'équité	équité faible (s_0, s_1)
$s_0 \wedge \forall\bigcirc s_0$		
$s_0 \wedge \exists\bigcirc s_0$		
$\forall\Box(s_0 \Rightarrow \exists\bigcirc s_0)$		
$\forall\Box(s_0 \Rightarrow \exists\Diamond s_2)$		
$\forall\Box(s_0 \Rightarrow \forall\Diamond s_2)$		
$\exists\Diamond\neg s_0$		
$\forall\Diamond\neg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

Exemple - l'arbre d'exécutions



Exemple 2



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

Invariance, Possibilité

Invariance

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

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

Stabilité

Spécifier la stabilité d'une situation si elle survient :

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

Possibilité

Spécifier qu'il est possible d'atteindre un certain état vérifiant P dans une certaine exécution :

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

Possibilité complexe

Séquence

Spécifier qu'un scénario d'exécution $\langle s_1 \to s_2 \to \ldots \to s_n \rangle$ 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 (par Q) à un requête donnée (par 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

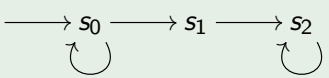
Spécifier que P est infiniment souvent vrai dans toute exécution :

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

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 $S =$ 

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
si $\|I\| = 1$	$\mathcal{S} \models \neg P \vee P$	~~$\mathcal{S} \models \neg P \vee P$~~
l'un de P ou Q inévitable	~~$\mathcal{S} \models \forall\Diamond P \vee \forall\Diamond Q$~~ $\mathcal{S} \models \forall\Diamond(P \vee Q)$	$\mathcal{S} \models \Diamond P \vee \Diamond Q$
l'un de P ou Q continu	~~$\mathcal{S} \models \forall\Box(P \vee Q)$~~ ~~$\mathcal{S} \models \forall\Box P \vee \forall\Box Q$~~	$\mathcal{S} \models \Box P \vee \Box Q$
$\neg P$ transitoire	~~$\mathcal{S} \models \forall\Diamond\forall\Box P$~~	$\mathcal{S} \models \Diamond\Box P$
possibilité	$S \models \exists\Diamond P \wedge \exists\Diamond\neg P$	~~$S \models \Diamond P \wedge \Diamond\neg P$~~
négation	$S \models \neg P \equiv S \not\models P$	$S \models \neg P \not\equiv S \not\models P$

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 CTL vs. LTL

Linear Time Logic

- \+ Intuitive
- − . . . sauf la négation
- \+ Suffisante pour décrire un système de transition
- \+ y compris l'équité exprimable
- − 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

CTL*

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

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

CTL* est strictement plus expressif que CTL et LTL.