

Systèmes de transition

Philippe Quéinnec, Xavier Thirioux, Aurélie Hurault

ENSEEIH
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 transition :
état = valeurs des variables + *flot de contrôle explicite*

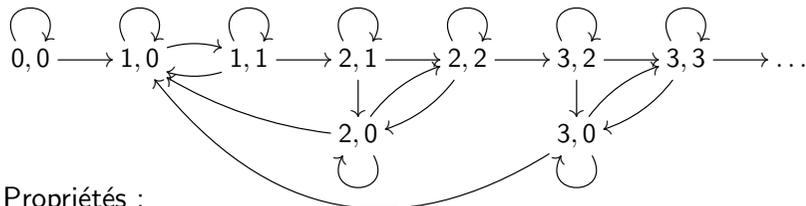
Exemple



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) \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 transition
- 5 Outils : vérificateur automatique, assistant de preuve

Auteur principal : **Leslie Lamport**



Plan du cours

- 1 (C) Le cadre formel : systèmes de transition
- 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) Concurrency : allocation de ressources
- 12 (C) Vérification par preuve et vérification de modèles
- 13 (TP) Étude d'un protocole de communication en mémoire partagée 

Ressources

- moodle : 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)

Introduction

Première partie

Systèmes de transition

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 transition peut être construit :

- *avant* l'écriture du programme, pour explorer la faisabilité de l'algorithme.
Le programme final est un raffinement en utilisant le système de transition comme guide.
- *après* l'écriture du programme, par abstraction, en ne conservant que les aspects significatifs du programme concret pour obtenir le principe de l'algorithme.

Plutôt que prouver des programmes concrets, on prouve des algorithmes.



Système de transition



Système de transition (ST)

Un système de transition 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' .



Plan

- 1 Définitions
 - Système de transition
 - Traces, exécutions
 - États, graphe
 - Système de transition étiqueté
- 2 Représentations
 - Explicite
 - Implicite
- 3 Propriétés générales
 - Blocage
 - Réinitialisable
 - Bégaiement
- 4 Composition de systèmes de transition

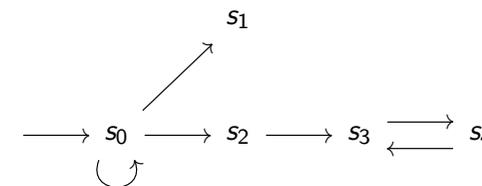


Exemple - système de transition

$$S = \{s_0, s_1, s_2, s_3, s_4\}$$

$$I = \{s_0\}$$

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



Séquences



Séquence

Soit S un ensemble.

S^* \triangleq l'ensemble des séquences finies sur S .

S^ω \triangleq l'ensemble des séquences infinies sur S .

σ_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 : $\langle s_1 \rightarrow s_2 \rightarrow \dots \rangle$.
- $\langle \rangle$: la séquence vide.

Pour une séquence finie σ :

- σ^* \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 infinies et traces issues d'un état



Traces infinies

Soit $\langle S, I, R \rangle$ un système de transition, et $s_0 \in S$.

On appelle **trace infinie** à partir de s_0 un élément $tr \in S^\omega$ tel que :

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

Traces issues d'un état

Soit $\langle S, I, R \rangle$ un système de transition, et $s \in S$.

$Traces(s)$ \triangleq l'ensemble des traces infinies ou finies maximales commençant à l'état s .



Traces finies



Traces finies

Soit $\langle S, I, R \rangle$ un système de transition.

On appelle **trace finie** une séquence finie $\sigma \in S^*$ telle que :

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

Traces finies maximales

Soit $\langle S, I, R \rangle$ un système de transition.

Une trace finie $\langle s_0 \rightarrow s_1 \rightarrow \dots \rightarrow s_{n-1} \rightarrow s_n \rangle \in S^*$ 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.



Exécutions



Exécutions

Soit $\mathcal{S} = \langle S, I, R \rangle$ un système de transitions.

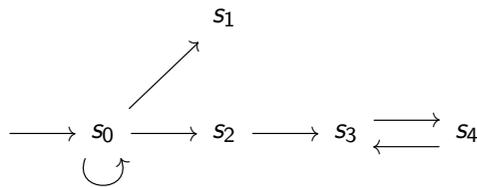
Une **exécution** $\sigma = \langle s_0 \rightarrow \dots \rangle$ est une trace infinie ou finie maximale telle que $s_0 \in I$.

$Exec(\mathcal{S}) \triangleq$ l'ensemble des exécutions de $\mathcal{S} = \bigcup_{s_0 \in I} Traces(s_0)$.

Une exécution vide $\langle \rangle$ existe si et seulement si $I = \emptyset$ (et c'est la seule exécution du ST).



Exemple - traces, exécutions



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

$$\text{Traces}(s_1) = \langle s_1 \rangle$$

$$\text{Traces}(s_3) = \langle (s_3 \rightarrow s_4)^\omega \rangle$$

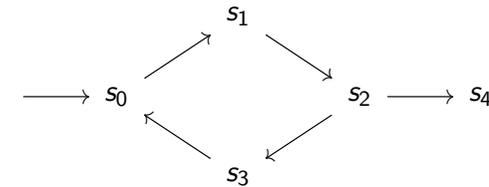
$$\text{Traces}(s_2) = \langle s_2 \rightarrow (s_3 \rightarrow s_4)^\omega \rangle$$

$$\text{Traces}(s_0) = \langle s_0^\omega \rangle, \langle s_0^+ \rightarrow s_1 \rangle, \langle s_0^+ \rightarrow s_2 \rightarrow (s_3 \rightarrow s_4)^\omega \rangle$$

$$\text{Exec}(S) = \text{Traces}(s_0)$$



Exemple 2 - traces, exécutions



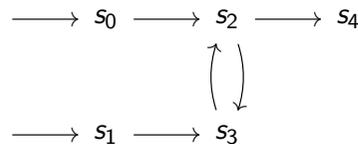
$$\text{Traces}(s_2) =$$

$$\text{Traces}(s_0) =$$

$$\text{Exec}(S) =$$



Exemple 3 - traces, exécutions



$$\text{Traces}(s_2) =$$

$$\text{Traces}(s_0) =$$

$$\text{Traces}(s_1) =$$

$$\text{Exec}(S) =$$



États accessibles

État accessible

Soit $S = \langle S, I, R \rangle$ un système de transition.

$s \in S$ 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).

$\text{Acc}(S) \triangleq$ l'ensemble des états accessibles de S .



Graphe des exécutions

Graphe des exécutions

Soit $\mathcal{S} = \langle S, I, R \rangle$ un système de transition.

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 $\langle S \cap Acc(\mathcal{S}), R \cap (Acc(\mathcal{S}) \times Acc(\mathcal{S})) \rangle$.



Équivalence aux ST sans étiquette



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

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

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.



Système de transition étiqueté



ST étiqueté

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

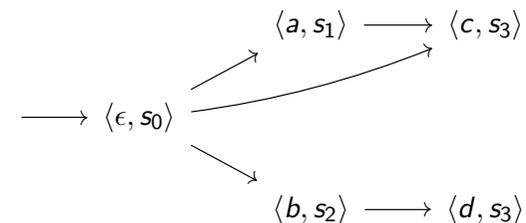
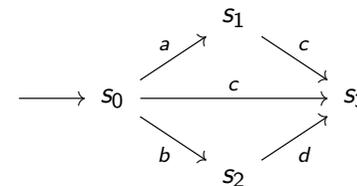
- S : ensemble d'états.
- $I \subseteq S$: ensemble des états initiaux.
- $R \subseteq S \times S$: relation de transitions entre paires d'états.
- L : ensemble d'étiquettes.
- $Etiq$: fonction qui associe une étiquette à chaque transition : $Etiq \in R \rightarrow L$.

Un ST étiqueté semble se rapprocher des automates.

Mais : exécutions infinies, pas d'état terminal, pas de contrôle externe des transitions.



Exemple - équivalence avec/sans étiquette



Différences entre système de transition et automate

Système de transition \neq automate à états finis

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

Contrairement aux automates à états finis, les systèmes de transition sont Turing-complets, i.e. permettent de décrire n'importe quel calcul.



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 transition où le nombre d'états et de transitions est fini.



Plan

- 1 Définitions
 - Système de transition
 - Traces, exécutions
 - États, graphe
 - Système de transition étiqueté
- 2 Représentations
 - Explicite
 - Implicite
- 3 Propriétés générales
 - Blocage
 - Réinitialisable
 - Bégaiement
- 4 Composition de systèmes de transition



Représentation en intention



Représentation symbolique à l'aide de variables.

Système de transition à base de variables

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

- $V = \{v_1, \dots, v_n\}$: ensemble fini de variables.
- $Init(v_1, \dots, v_n)$: prédicat définissant les états initiaux et portant sur les variables v_i .
- $Trans(v_1, \dots, v_n, v'_1, \dots, v'_n)$: 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

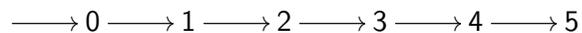


```
i = 0;
while (i < N) {
  i = i+1;
}
```

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 $N = 5$:



Symboliquement (en intention) :

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

$$I \triangleq i = 0$$

$$T \triangleq i < N \wedge i' = i + 1 \quad \text{ou} \quad T \triangleq i' \leq N \wedge i' - i = 1$$



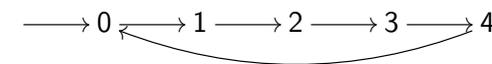
Exemple : un compteur cyclique

```
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 $N = 5$.



Symboliquement :

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

$$I \triangleq i = 0$$

$$T \triangleq i' = (i + 1) \bmod N$$



Exemple : un entier oscillant

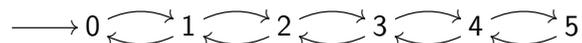


```
i = 0;
while (true) {
  i > 0 -> i = i - 1;
  or i < N -> i = i + 1;
}
```

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 $N = 5$.



Symboliquement :

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

$$I \triangleq i = 0$$

$$T \triangleq i > 0 \wedge i' = i - 1 \quad \text{ou} \quad T \triangleq |i' - i| = 1 \wedge 0 \leq i' \leq N$$

$$\vee i < N \wedge i' = i + 1$$



Système de transition correspondant



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

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



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

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

Exemple - prédicats

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

Prédicats d'état : $I, n < 20$

Prédicats de transition : $T, n' - n > 3$

Plan

- 1 Définitions
 - Système de transition
 - Traces, exécutions
 - États, graphe
 - Système de transition étiqueté
- 2 Représentations
 - Explicite
 - Implicite
- 3 Propriétés générales
 - Blocage
 - Réinitialisable
 - Bégaïement
- 4 Composition de systèmes de transition

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.



Plan

- 1 Définitions
 - Système de transition
 - Traces, exécutions
 - États, graphe
 - Système de transition étiqueté
- 2 Représentations
 - Explicite
 - Implicite
- 3 Propriétés générales
 - Blocage
 - Réinitialisable
 - Bégaïement
- 4 Composition de systèmes de transition



Bégaïement



Bégaïement

Un état s bégaie \triangleq l'état possède une boucle : $(s, s) \in R$.

Un système de transition 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 bégaïement) a un unique successeur avec bégaïement : lui-même. La terminaison (l'interblocage) $\dots \rightarrow s_i$ est alors $\dots \rightarrow s_i^\omega$.
- Composer plusieurs systèmes de transition.



Composition : produit libre



Produit libre

La composition des ST avec bégaïement $\langle V_1, I_1, T_1 \rangle$ 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)

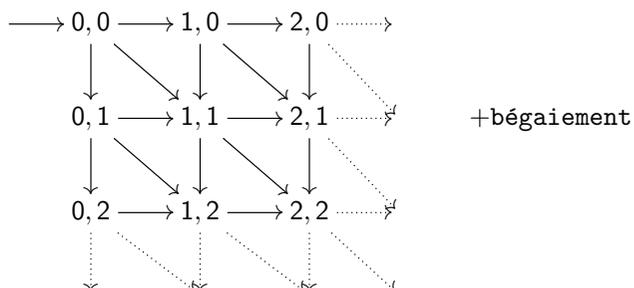
Comme T_1 et T_2 peuvent bégayer, $T_1 \wedge T_2$ signifie qu'on peut exécuter une transition de T_1 seule et T_2 bégayant, ou bien l'inverse, ou bien exécuter T_1 en même temps que T_2 .

Variables communes : si $V_1 \cap V_2 \neq \emptyset$, les transitions dans T doivent respecter les contraintes des deux ST (conjonction).



Exemple - produit libre

$$\left(\begin{array}{l} V_1 \triangleq i \in \mathbb{N} \\ I_1 \triangleq i = 0 \\ T_1 \triangleq i' = i + 1 \\ \vee i' = i \end{array} \right) \otimes \left(\begin{array}{l} V_2 \triangleq j \in \mathbb{N} \\ I_2 \triangleq j = 0 \\ T_2 \triangleq j' = j + 1 \\ \vee j' = j \end{array} \right) \rightarrow \left(\begin{array}{l} V \triangleq i, j \in \mathbb{N} \\ I \triangleq i = 0 \wedge j = 0 \\ T \triangleq i' = i + 1 \wedge j' = j \\ \vee i' = i \wedge j' = j + 1 \\ \vee i' = i + 1 \wedge j' = j + 1 \\ \vee i' = i \wedge j' = j \end{array} \right)$$



Composition : produit synchronisé strict (ou fermé)



Produit synchronisé strict

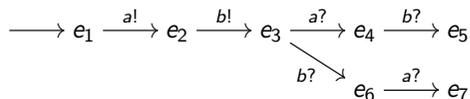
Le produit synchrone des ST étiquetés $\langle S_1, I_1, R_1, L_1 \rangle$ et $\langle S_2, I_2, R_2, L_2 \rangle$ est $\langle S, I, R, L \rangle$ où :

- $S \triangleq S_1 \times S_2$ (couple d'états)
(chaque sous-système démarre dans un de ses états initiaux)
- $I \triangleq I_1 \times I_2$
(les deux sous-systèmes évoluent selon des transitions portant les mêmes étiquettes)
- $R \triangleq \{((s_1, s_2), (s'_1, s'_2)) \mid (s_1, s'_1) \in R_1 \wedge (s_2, s'_2) \in R_2 \wedge \text{Etiqu}((s_1, s'_1)) = \text{Etiqu}((s_2, s'_2))\}$
- $L = L_1 \cap L_2$ (étiquettes communes seulement)

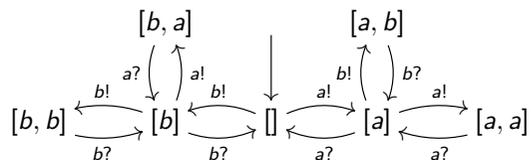
Exemple – produit synchronisé strict



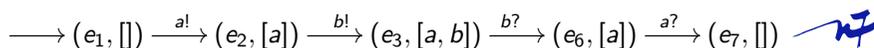
Interprétation : a et b sont des messages, $a!$ / $a?$ sont l'envoi / la réception d'un message. Le premier ST est une application quelconque et le deuxième décrit les propriétés de la communication.



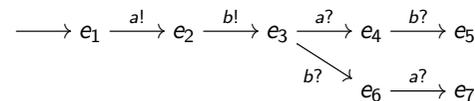
Synchronisé strict avec LIFO à 2 éléments (pile)



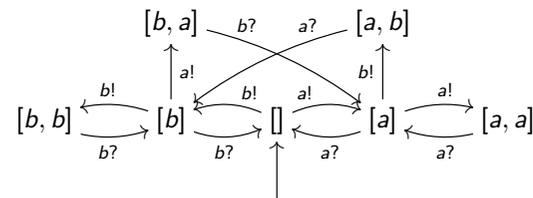
Donne



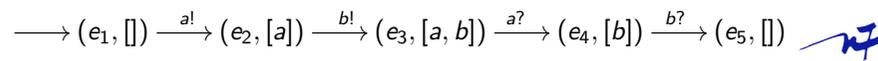
Exemple – produit synchronisé strict



Synchronisé strict avec FIFO à 2 éléments (file)



Donne



Produit synchronisé ouvert

Le produit synchronisé 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)
- $I \triangleq I_1 \times I_2$
- $R \triangleq \left\{ \begin{array}{l} ((s_1, s_2), (s'_1, s'_2)) \mid (s_1, s'_1) \in R_1 \wedge (s_2, s'_2) \in R_2 \\ \quad \wedge \text{Etiqu}((s_1, s'_1)) = \text{Etiqu}((s_2, s'_2)) \\ ((s_1, s_2), (s'_1, s_2)) \mid (s_1, s'_1) \in R_1 \wedge \text{Etiqu}((s_1, s'_1)) \notin L_2 \\ ((s_1, s_2), (s_1, s'_2)) \mid (s_2, s'_2) \in R_2 \wedge \text{Etiqu}((s_2, s'_2)) \notin L_1 \end{array} \right\}$
- $L = L_1 \cup L_2$

Synchronisation sur étiquette commune, bégaiement sur étiquette absente.



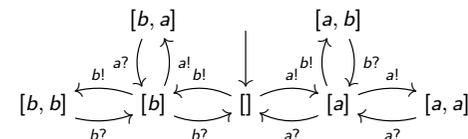
Cette séance a présenté :

- la définition de système de transition (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



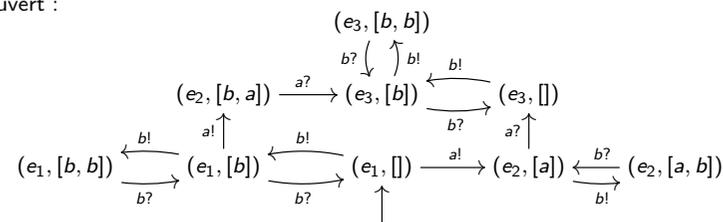
$$\longrightarrow e_1 \xrightarrow{a!} e_2 \xrightarrow{a?} e_3$$

Synchronisé avec LIFO à 2 éléments (pile)



Donne

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



Deuxième partie

TLA⁺ – les actions

NF

TLA⁺ : Temporal Logic of ActionsTLA⁺ : Temporal Logic of Actions

- Un langage outillé pour modéliser les programmes et systèmes
- Particulièrement adapté aux programmes et systèmes distribués / concurrents
- Basé sur les systèmes de transition
- Une toolbox embarquant un éditeur de texte, un outil de vérification de modèles (TLC) et pour visualiser les exécutions, un outil pour écrire et vérifier des preuves (TLAPS)
- <http://lamport.azurewebsites.net/tla/tla.html>

NF

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

NF

Plan

- 1 Spécification
 - Structure
 - Constantes
 - Expressions
- 2 Actions
- 3 Fonctions
 - Fonctions de X dans Y
 - Les enregistrements (records)
 - Tuples & séquences
 - Définition récursive
- 4 Divers

NF

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)

nf

MODULE *exemple1*

EXTENDS *Naturals*

VARIABLE *x*

États initiaux

Init $\triangleq x \in 0..2$ équivalent à $x \in \text{Nat} \wedge 0 \leq x \wedge x < 3$

Actions

Plus $\triangleq x' = x + 1$

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

Next $\triangleq \text{Plus} \vee \text{Moins}$

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

nf

Exemple

Correspond au système de transition :

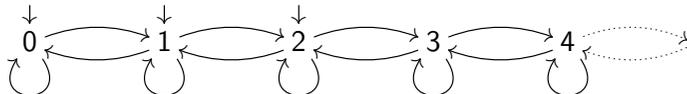
$V \triangleq x \in \mathbb{N}$

$I \triangleq 0 \leq x \leq 2$

$R \triangleq x' = x + 1$

$\vee x > 0 \wedge x' = x - 1$

$\vee x' = x$



nf

Constantes

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

nf

Expressions autorisées

Tout ce qui est axiomatisable :

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

Plan

- 1 Spécification
 - Structure
 - Constantes
 - Expressions
- 2 Actions
- 3 Fonctions
 - Fonctions de X dans Y
 - Les enregistrements (records)
 - Tuples & séquences
 - Définition récursive
- 4 Divers

Opérateurs ensemblistes

$\{e_1, \dots, e_n\}$	ensemble en extension
$n..m$	$\{i \in \text{Nat} : n \leq i \leq m\}$
$\{x \in S : p(x)\}$	l'ensemble des éléments de S vérifiant la propriété p $\{n \in 1..10 : n\%2 = 0\} = \{2, 4, 6, 8, 10\}$ $\{n \in \text{Nat} : n\%2 = 1\} =$ les entiers impairs
$\{f(x) : x \in S\}$	l'ensemble des valeurs de l'opérateur f en S $\{2 * n : n \in 1..5\} = \{2, 4, 6, 8, 10\}$ $\{2 * n + 1 : n \in \text{Nat}\} =$ les entiers 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 $\{1, 2\} = \{\{\}, \{1\}, \{2\}, \{1, 2\}\}$

Actions

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

Une action n'est pas une affectation.

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

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 $var' = \dots$
ou un prédicat de transition non déterministe $var' \in \dots$

Se rapproche d'une instruction exécutable.

$x < 10 \wedge x' = x + 1$
plutôt que $x' = x + 1 \wedge x' < 11$
ou $x' - x = 1 \wedge x' < 11$

nf

Bégaiement

Bégaiement

$[A]_f \triangleq \mathcal{A} \vee f' = f$, 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

$(x' = x + 1) = (x' = x + 1 \wedge y' = \text{n'importe quoi})$
 $\neq (x' = x + 1 \wedge y' = y)$

UNCHANGED

UNCHANGED $e \triangleq e' = e$

nf

MODULE *AlternatingBit*

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

Init $\triangleq \wedge val \in Data$
 $\wedge ready \in \{0, 1\}$
 $\wedge ack = ready$

Send $\triangleq \wedge ready = ack$
 $\wedge val' \in Data$
 $\wedge ready' = 1 - ready$
 $\wedge \text{UNCHANGED } ack$

Receive $\triangleq \wedge ready \neq ack$
 $\wedge ack' = 1 - ack$
 $\wedge \text{UNCHANGED } \langle val, ready \rangle$

Next $\triangleq Send \vee Receive$
Spec $\triangleq Init \wedge \square [Next]_{\langle val, ready, ack \rangle}$

Mise en pratique : factorielle

Écrire la spécification d'un programme qui définit la factorielle d'un entier 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 réaliser plusieurs multiplications en une transition

nf

Plan

- 1 Spécification
 - Structure
 - Constantes
 - Expressions
- 2 Actions
- 3 Fonctions
 - Fonctions de X dans Y
 - Les enregistrements (records)
 - Tuples & séquences
 - Définition récursive
- 4 Divers

Fonctions

Fonction au sens mathématique : *mapping*, correspondance.

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

Une fonction est une **valeur**.
Une variable contenant une fonction peut changer de valeur
 \Rightarrow "la fonction change" par abus de langage.

Définition

Définition d'un symbole constant

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

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

Définition d'une valeur

$[x \in S \mapsto expr]$

Exemples : $[x \in 1..4 \mapsto 2 * x]$, $[x \in \{1, 2, 3, 5, 7, 11\} \mapsto 2 * x + 1]$

Tableaux

Tableau : fonction $t \in [X \rightarrow Y]$ 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  $\triangleq \wedge a[1] = 2$ 
       $\wedge a' = [i \in 1..6 \mapsto i * 2]$ 
Act2  $\triangleq \wedge a[2] = 4$ 
       $\wedge a' = [i \in 1..6 \mapsto \text{IF } i = 2 \text{ THEN } 8 \text{ ELSE } a[i]]$ 

```

EXCEPT

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

$(a' = [a \text{ EXCEPT } ![2] = 8]) \neq (a[2]' = 8)$

Enregistrements

Enregistrement

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

Écriture simplifiée :

$["toto" \mapsto 1, "titi" \mapsto 2] = [toto \mapsto 1, titi \mapsto 2]$
 $rec["toto"] = rec.toto$

Tuple

n-tuple

Notation : $\langle a, b, c \rangle$.

Un n-tuple est une fonction de domaine $= \{1, \dots, n\}$:

$\langle a, b, c \rangle[3] = c$

Pratique pour représenter des relations :

$\{\langle x, y \rangle \in X \times Y : R(x, y)\}$.

Exemple : $\{\langle a, b \rangle \in \text{Nat} \times \text{Nat} : a = 2 * b\}$.

Séquences

Séquences

$Seq(T) \triangleq \text{UNION } \{[1..n \rightarrow T] : n \in \text{Nat}\}$

\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 :

$s \circ t \triangleq [i \in 1..(Len(s) + Len(t))$

$\mapsto \text{IF } i \leq Len(s) \text{ THEN } s[i] \text{ ELSE } t[i - Len(s)]]$

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

Fonction \neq opérateur

$$\text{SuccF}[x \in \text{Nat}] \triangleq x + 1$$

$$\text{SuccO}(x) \triangleq x + 1$$

- *SuccF* est une définition de fonction au sens mathématique
 - Équivalent à $\text{SuccF} \triangleq [x \in \text{Nat} \mapsto x + 1]$
 - Son domaine est un ensemble : $\text{DOMAIN SuccF} = \mathbb{N}$
 - Son co-domaine est un ensemble : $\{\text{SuccF}[x] : x \in \text{DOMAIN SuccF}\} = \mathbb{N}^*$
 - $\text{SuccF} \in [X \rightarrow Y]$ a du sens
- *SuccO* 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
 - $\text{SuccO} \in [X \rightarrow Y]$ n'a pas de sens

Plan

- 1 Spécification
 - Structure
 - Constantes
 - Expressions
- 2 Actions
- 3 Fonctions
 - Fonctions de X dans Y
 - Les enregistrements (records)
 - Tuples & séquences
 - Définition récursive
- 4 Divers

Définition récursive

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

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

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

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 : $\text{LET } i \triangleq g(x) \text{ IN } f(i) \equiv f(g(x))$

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

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 .

Maximum d'un ensemble

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

Somme des éléments d'un ensemble

$\text{Somme}[S \in \text{SUBSET } \text{Nat}] \triangleq$
 IF $S = \emptyset$ THEN 0
 LET $e \triangleq \text{CHOOSE } x \in S : \text{TRUE}$
 IN $e + \text{Somme}[S \setminus \{e\}]$

NF

Conclusion

Les actions TLA⁺ sont un noyau minimal permettant d'exprimer toute spécification de système de transition (= tout programme) à partir de :

- la logique du premier ordre
- la théorie des ensembles
- les fonctions (*mapping*)
- les valeurs avant/après des variables

NF

Choix déterministe - 2

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.

- La spécification

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

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

- La spécification

$(x \in \text{Nat}) \wedge \square [x' \in \text{Nat}]_{(x)}$

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

NF

Troisième partie

L'équité dans les systèmes de transition

Plan

- 1 Contraintes d'équité
- 2 Équité sur les états
 - Équité simple
 - Équité multiple
 - Équité conditionnelle
- 3 Équité sur les transitions
 - Équité faible
 - Équité forte
 - Équité sur les étiquettes



Pourquoi de l'équité ?

MODULE *oscillant*

CONSTANT N

VARIABLE i

$Init \triangleq i = 0$

$Next \triangleq \bigvee i > 0 \wedge i' = i - 1$
 $\bigvee i < N \wedge i' = i + 1$

$Spec \triangleq Init \wedge \square[Next]_i$



On pourrait vouloir **éliminer** les exécutions :

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



Contraintes d'équité / *fairness*



Les contraintes d'équité spécifient que certains états (resp. certaines transitions) doivent être visités (resp. exécutés) **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.





Ensemble récurrent d'états

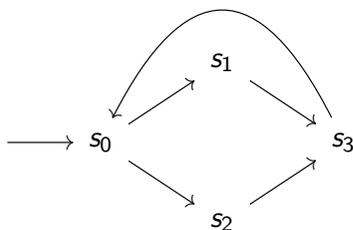
Soit $\mathcal{S} = \langle S, I, R \rangle$ un système de transition et $\sigma = \langle s_0 \rightarrow \dots \rangle$ une exécution.

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 .

$Inf_{\mathcal{S}}(P, \sigma) \triangleq P$ est un ensemble récurrent d'états dans σ .

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



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

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



Ensemble récurrent de transitions

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

Un ensemble de transitions Q est **récurrent** dans σ si :

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

$Inf_{\mathcal{T}}(Q, \sigma) \triangleq Q$ est un ensemble récurrent de transitions dans σ .



- 1 Contraintes d'équité
- 2 Équité sur les états
 - Équité simple
 - Équité multiple
 - Équité conditionnelle
- 3 Équité sur les transitions
 - Équité faible
 - Équité forte
 - Équité sur les étiquettes



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.

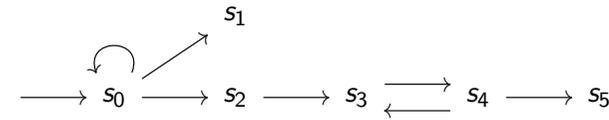
Alors toute exécution σ doit être telle que $\text{Inf}_S(F, \sigma)$.

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 $S \triangleq i = 0 \wedge \square((i' = i + 1) \vee (i' = i))$, si on se donne $F \triangleq \{i \% 2 = 0\}$, les exécutions $0 \rightarrow 1 \rightarrow 2^\omega$ et $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \dots$ sont valides.



Exemple - équité simple

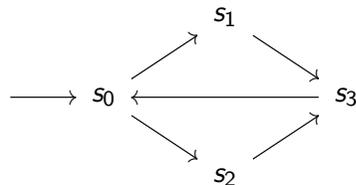


$$\text{Exec}(S) = \langle s_0^\omega \rangle, \langle s_0^+ \rightarrow s_2 \rightarrow (s_3 \rightarrow s_4)^\omega \rangle, \langle s_0^+ \rightarrow s_1 \rangle, \langle s_0^+ \rightarrow s_2 \rightarrow (s_3 \rightarrow s_4)^+ \rightarrow s_5 \rangle$$

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



Exemple - équité simple

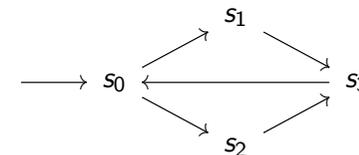


On fixe : équité simple sur $\{s_1\}$.

- légale $\langle (s_0 \rightarrow s_1 \rightarrow s_3)^\omega \rangle$
- légale $\langle (s_0 \rightarrow s_1 \rightarrow s_3 \rightarrow s_0 \rightarrow s_2 \rightarrow s_3)^\omega \rangle$
- illégal $\langle (s_0 \rightarrow s_1 \rightarrow s_3)^* \rightarrow (s_0 \rightarrow s_2 \rightarrow s_3)^\omega \rangle$
- légale $\langle (s_0 \rightarrow s_1 \rightarrow s_3 \rightarrow (s_0 \rightarrow s_2 \rightarrow s_3)^*)^\omega \rangle$



Exemple - équité simple



$$\text{Exec}(S) =$$

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 $J = \{0, 1, 2, \dots\}$, d'ensembles équitables $\{F_i\}_{i \in J}$.

Toute exécution σ doit être telle que $\forall i \in J : \text{Inf}_S(F_i, \sigma)$.

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

⇒ l'équité simple est un cas particulier de l'équité multiple.

Exemple - équité multiple



$\text{Exec}(S) =$

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

Équivalence équité multiple finie \leftrightarrow simple



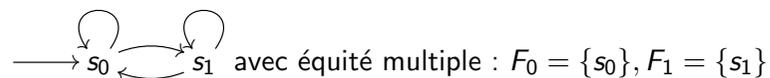
Cas simple : J est fini, de cardinalité $|J|$.

Le système $\langle S, I, R \rangle$ avec équité multiple $\{F_i\}_{i \in J}$ est équivalent à $\langle S', I', R' \rangle$ à équité simple F' (mêmes exécutions projetées sur S) :

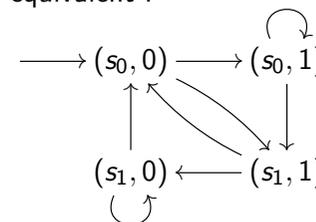
- $S' = S \times J$
- $I' = I \times \{0\}$
- $R' = \{(\langle s, j \rangle, \langle s', j + 1 \text{ mod } |J| \rangle) \mid (s, s') \in R \wedge s \in F_j\} \cup \{(\langle s, j \rangle, \langle s', j \rangle) \mid (s, s') \in R \wedge s \notin F_j\}$
- Équité simple $F' = F_0 \times \{0\}$

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



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 F' est équivalent :

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

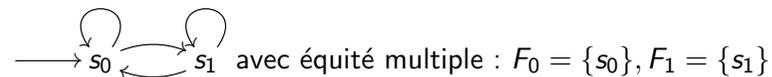
avec : $i \oplus 1 \triangleq \begin{cases} i + 1 & \text{si } J \text{ est infini} \\ i + 1 \bmod |J| & \text{sinon} \end{cases}$

Dans une exécution équitale, les compteurs i, j forment un triangle :

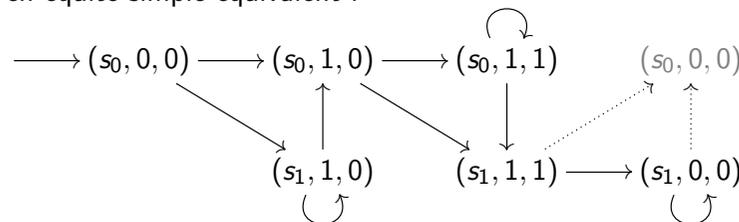
$\langle (0, 0) \rightarrow (1, 0) \rightarrow (1, 1) \rightarrow (2, 0) \rightarrow (2, 1) \rightarrow (2, 2) \rightarrow (3, 0) \rightarrow \dots \rangle$



Exemple équité multiple



ST en équité simple équivalent :



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



Équité conditionnelle sur les états



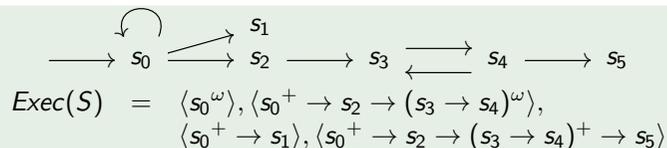
Équité conditionnelle

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 $Inf_S(F, \sigma) \Rightarrow Inf_S(G, \sigma)$.

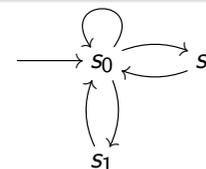
Si F est récurrent dans σ , alors G doit être récurrent dans σ .



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



Exemple - équité 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 F' est équivalent :

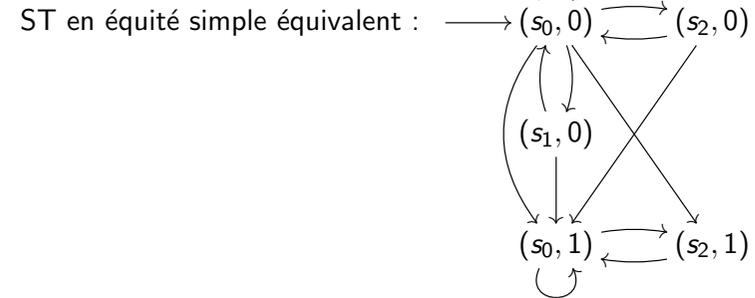
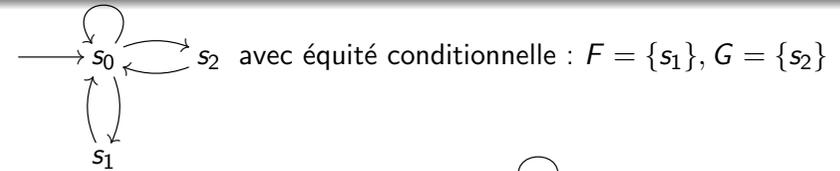
- $S' = (S \times \{0\}) \cup ((S \setminus F) \times \{1\})$
- $I' = I \times \{0\}$
- $R' = \{(\langle s, 0 \rangle, \langle s', 0 \rangle) \mid (s, s') \in R\} \cup \{(\langle s, 0 \rangle, \langle s', 1 \rangle) \mid (s, s') \in R \wedge s' \in (S \setminus F)\} \cup \{(\langle s, 1 \rangle, \langle s', 1 \rangle) \mid (s, s') \in R \wedge s, s' \in (S \setminus F)\}$
- 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 simple sur $\{(s_2, 0), (s_0, 1), (s_2, 1)\}$



Plan

- 1 Contraintes d'équité
- 2 Équité sur les états
 - Équité simple
 - Équité multiple
 - Équité conditionnelle
- 3 Équité sur les transitions
 - Équité faible
 - Équité forte
 - Équité sur les étiquettes



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 :

$$\text{Inf}_S(S \setminus \text{dom}(F), \sigma) \vee \text{Inf}_T(F, \sigma)$$

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

Ou, de manière équivalente :

$$\neg \text{Inf}_S(S \setminus \text{dom}(F), \sigma) \Rightarrow \text{Inf}_T(F, \sigma)$$

(si l'ensemble d'états $S \setminus \text{dom}(F)$ 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

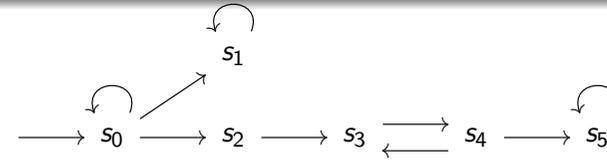


$$\text{Exec}(S) = \langle (s_0^+ \rightarrow s_1^+)^{\omega}, \langle (s_0^+ \rightarrow s_1^+)^* \rightarrow s_0^{\omega}, \langle (s_0^+ \rightarrow s_1^+)^* \rightarrow s_0^+ \rightarrow s_1^{\omega} \rangle \rangle$$

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



Exemple - équité faible

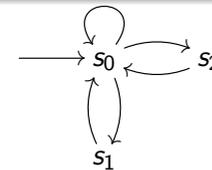


$$\text{Exec}(S) = \langle s_0^{\omega}, \langle s_0^+ \rightarrow s_2 \rightarrow (s_3 \rightarrow s_4)^{\omega}, \langle s_0^+ \rightarrow s_1^{\omega}, \langle s_0^+ \rightarrow s_2 \rightarrow (s_3 \rightarrow s_4)^+ \rightarrow s_5^{\omega} \rangle \rangle$$

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



Exemple - équité faible



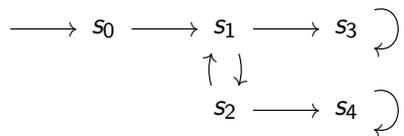
On note $T \triangleq s_0^* \rightarrow (s_0 \rightarrow s_1)^* \rightarrow (s_0 \rightarrow s_2)^* \setminus \langle \rangle$.
(le $\setminus \langle \rangle$ garantit que T ne contient pas la séquence vide)

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

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



Exemple - équité faible

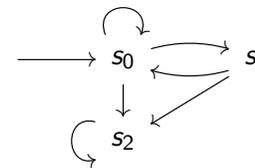


Exec(S) =

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

nf

Exemple - équité faible



Exec(S) =

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

nf

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 F' 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 .

nf

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 :

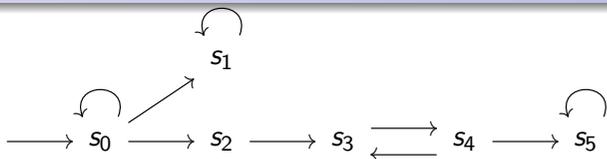
$$Inf_S(dom(F), \sigma) \Rightarrow Inf_T(F, \sigma)$$

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.

nf

Exemple - équité forte

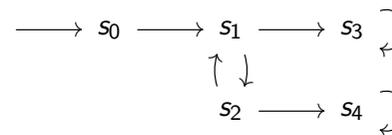


$$Exec(S) = \langle s_0^\omega \rangle, \langle s_0^+ \rightarrow s_2 \rightarrow (s_3 \rightarrow s_4)^\omega \rangle, \langle s_0^+ \rightarrow s_1^\omega \rangle, \langle s_0^+ \rightarrow s_2 \rightarrow (s_3 \rightarrow s_4)^+ \rightarrow s_5^\omega \rangle$$

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



Exemple - équité forte

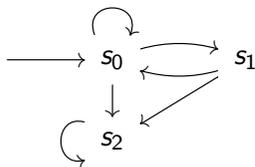


$$Exec(S) =$$

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 \rightarrow 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 $F' \Rightarrow G'$ 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 $F' = \text{dom}(F) \times \{0, 1\}$
 $G' = S \times \{1\}$

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 infini sur s_0)

Équité faible sur $\{(s_1, s_2)\}$ (idem pour s_1)

Équité faible sur $\{(s_2, s_1)\}$ (idem pour s_2)

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.

MF

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 faisable / infiniment souvent faisable, elle aura lieu : le système n'est pas injuste vis-à-vis de ces actions.

MF

Équité sur les étiquettes

Dans le cas d'un système de transition 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 $Etq^{-1}(F)$.

MF

Plan

Quatrième partie

LTL – logique temporelle linéaire

- 1 Logiques temporelles
- 2 Logique temporelle linéaire – LTL
 - Syntaxe
 - Sémantique
 - Réduction
- 3 Expressivité
 - Exemples
 - Propriétés classiques

Logiques temporelles



Plan

- 1 Logiques temporelles
- 2 Logique temporelle linéaire – LTL
 - Syntaxe
 - Sémantique
 - Réduction
- 3 Expressivité
 - Exemples
 - Propriétés classiques

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.

Linear Temporal Logic



Modèles

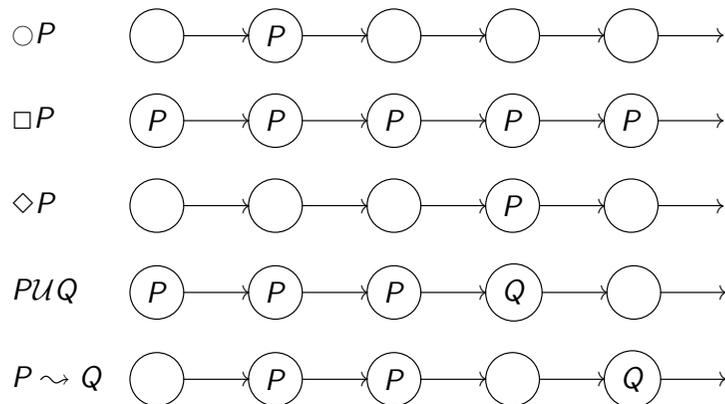
Une formule LTL se rapporte toujours à une **trace** donnée σ d'un système : les traces constituent les modèles de cette logique.

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 $\sigma \in S^* \cup S^\omega$, tel que pour tout s_i, s_{i+1} consécutifs, $(s_i, s_{i+1}) \in R$.



Intuition sémantique



Syntaxe de la LTL



formule	nom	interprétation
s		le premier état de la trace est s
$\neg P$		
$P \vee Q$		
$P \wedge Q$		
$\bigcirc P$	<i>next</i>	P est vrai à l'instant suivant
$\square P$	<i>always</i>	P est toujours vrai i.e. à tout instant à partir de l'instant courant
$\diamond P$	<i>eventually</i>	P sera vrai (dans le futur)
$P U Q$	<i>until</i>	Q sera vrai, et en attendant P reste vrai
$P \rightsquigarrow Q$	<i>leadsto</i>	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.



Opérateurs minimaux



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

- $\diamond P \triangleq \text{True } U P$
- $\square P \triangleq \neg \diamond \neg P$
- $P \rightsquigarrow Q \triangleq \square (P \Rightarrow \diamond Q)$



Syntaxe alternative



Syntaxe alternative

On trouve fréquemment une autre syntaxe :

- $\square \leftrightarrow G$ (*globally*)
- $\diamond \leftrightarrow F$ (*finally*)
- $\circ \leftrightarrow X$ (*next*)

Opérateurs complémentaires

- Opérateur *waiting-for* (ou *unless* ou *weak-until*)
 $PWQ \triangleq \square P \vee PUQ$
 Q finit peut-être par être vrai et en attendant P reste vrai
- Opérateur *release*
 $PRQ \triangleq QU(P \wedge Q)$
 Q reste vrai jusqu'à ce que P le devienne.



Opérateurs du passé

formule	nom	interprétation
$\ominus P$	<i>previously</i>	P est vrai dans l'instant précédent
$\boxminus P$	<i>has-always-been</i>	P a toujours été vrai jusqu'à l'instant courant
$\diamond P$	<i>once</i>	P a été vrai dans le passé
PSQ	<i>since</i>	Q a été vrai dans le passé et P est resté vrai depuis la dernière occurrence de Q
PBQ	<i>back-to</i>	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 $\langle s_i \rightarrow s_{i+1} \rightarrow \dots \rangle$ d'une trace
 $\sigma = \langle s_0 \rightarrow s_1 \rightarrow \dots \rangle$.

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

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

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

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



Sémantique (opérateurs temporels)



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

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

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



Sémantique (opérateurs temporels dérivés)



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

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

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



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_{Spec} :

$$\mathcal{M} \models F_{Spec}$$

- revient à déterminer la validité de :

$$F_{\mathcal{M}} \Rightarrow F_{Spec}$$

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).

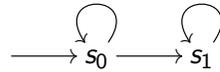


Plan

- 1 Logiques temporelles
- 2 Logique temporelle linéaire – LTL
 - Syntaxe
 - Sémantique
 - Réduction
- 3 Expressivité
 - Exemples
 - Propriétés classiques



Exemple 1



	pas d'équité	équité faible (s ₀ , s ₁)
$s_0 \wedge \circ s_0$		
$s_0 \wedge \circ (s_0 \vee s_1)$		
$\square (s_0 \Rightarrow \circ s_0)$		
$\square (s_0 \Rightarrow \circ (s_0 \vee s_1))$		
$\square (s_1 \Rightarrow \circ s_1)$		
$\diamond (s_0 \wedge \circ s_1)$		
$\square s_0$		
$\diamond \neg s_0$		
$\diamond \square s_1$		
$s_0 \mathcal{W} s_1$		
$s_0 \mathcal{U} s_1$		

nf

Exemple 2



	pas d'équité	faible (s ₁ , s ₂)	forte (s ₁ , s ₂)
$\square \diamond \neg s_1$			
$\square (s_1 \Rightarrow \diamond s_2)$			
$\diamond \square (s_1 \vee s_2)$			
$\square (s_1 \mathcal{U} s_2)$			
$\square (s_0 \Rightarrow s_0 \mathcal{U} s_1)$			
$\square (s_0 \mathcal{U} (s_1 \vee s_2))$			
$\square (s_1 \Rightarrow s_1 \mathcal{U} s_2)$			
$\diamond (s_1 \mathcal{U} s_2)$			
$\diamond (s_1 \mathcal{W} s_2)$			
$\square \diamond (s_1 \mathcal{U} (s_0 \vee s_2))$			

nf

Sûreté/vivacité – Safety/Liveness

On qualifie de

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

nf

Invariance, stabilité

Invariance

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

$$S \models \square P$$

où P est un prédicat d'état.

Stabilité

Spécifier la stabilité d'une situation si elle survient :

$$S \models \square (P \Rightarrow \square P)$$

où P est un prédicat d'état.

nf

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 :

$$S \not\models \Box \neg P$$

Attention à la négation : $\neg \Box P = \Diamond \neg P$ mais $S \not\models \Box P \not\equiv S \models \Diamond \neg P$



Négation



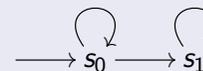
Négation : danger !

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

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

$S \not\models Q$ 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 $S \not\models Q \wedge S \not\models \neg Q$:



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



Combinaisons

Infiniment souvent – Réponse

Spécifier que P est infiniment souvent vrai dans toute exécution :

$$S \models \Box \Diamond P$$

Finalement toujours – Persistance

Spécifier que P finit par rester définitivement vrai :

$$S \models \Diamond \Box P$$

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 une requête donnée (P) :

$$S \models \Box (P \Rightarrow \Diamond Q)$$

Souvent nommé leads-to :

$$S \models P \rightsquigarrow Q$$

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 :

$$S \models \Box (P \Rightarrow PWQ)$$



Équité des transitions – *Fairness*

Rappel informel :

faible : continûment faisable \rightarrow infiniment souvent fait

forte : infiniment souvent faisable \rightarrow infiniment souvent fait

Équité faible des transitions

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

$$\mathcal{S} \models \diamond \square \text{dom}(r) \Rightarrow \square \diamond r$$

$$\mathcal{S} \models \square \diamond \neg \text{dom}(r) \vee \square \diamond r$$

Équité forte des transitions

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

$$\mathcal{S} \models \square \diamond \text{dom}(r) \Rightarrow \square \diamond r$$

$$\mathcal{S} \models \diamond \square \neg \text{dom}(r) \vee \square \diamond r$$

(une transition $s_1 \rightarrow s_2$ est équivalente à $s_1 \wedge \bigcirc s_2$, et un ensemble de transition $\{t_1, t_2, \dots\}$ est équivalent à $t_1 \vee t_2 \vee \dots$)



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 $\langle S, I, R \rangle$:

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

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 :

- \square = 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 »
- ...



Objectifs

Cinquième partie

TLA⁺ – la logique

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



Plan

- 1 LTL et TLA⁺
 - Logique TLA⁺
 - Raffinage
- 2 Preuve axiomatique
- 3 Vérification de modèles



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 :

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



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

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

Raffinage de spécification

Raffinage simple

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

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

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 $\mathcal{A}_1 \vee \mathcal{A}_2 \vee \mathcal{A}_3 \vee \dots$
- \mathcal{E} = conjonction de contraintes d'équité portant sur les actions : $WF_v(\mathcal{A}_1) \wedge SF_v(\mathcal{A}_3) \wedge \dots$

Raffinage – exemple

Somme abstraite

```

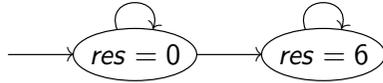
MODULE somme1
EXTENDS Naturals
CONSTANT N
VARIABLE res

TypeOK  $\triangleq$  res  $\in$  Nat

Init  $\triangleq$  res = 0
Next  $\triangleq$  res' = ((N + 1) * N)  $\div$  2
Spec  $\triangleq$  Init  $\wedge$   $\Box$ [Next]res  $\wedge$  WFres(Next)
    
```

Raffinage – exemple

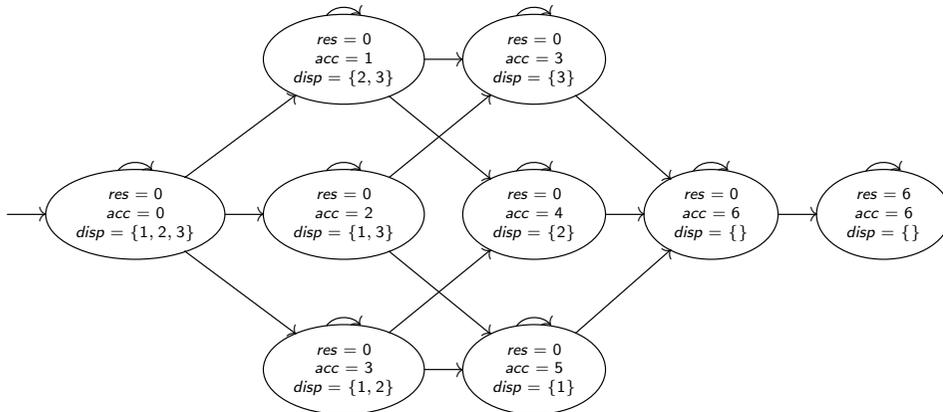
Grphe des exécutions pour N = 3



NT

Raffinage – exemple

Grphe des exécutions pour N = 3



Décomposition : introduction de transitions intermédiaires.

NT

Raffinage – exemple

Somme plus concrète

```

MODULE somme2
EXTENDS Naturals
CONSTANT N
VARIABLE res, acc, disp

TypeOK ≜ res ∈ Nat ∧ acc ∈ Nat ∧ disp ∈ SUBSET 1..N

Init ≜ res = 0 ∧ acc = 0 ∧ disp = 1..N
Next ≜ ∨ ∃ i ∈ disp : acc' = acc + i ∧ disp' = disp \ {i}
      ∧ UNCHANGED res
      ∧ ∨ disp = {} ∧ res' = acc ∧ UNCHANGED ⟨disp, acc⟩
Spec ≜ Init ∧ □[Next]⟨res, disp, acc⟩ ∧ WF⟨res, disp, acc⟩(Next)
    
```

NT

Raffinage – exemple

Somme2 raffine somme1

```

MODULE somme2_raffine_somme1
EXTENDS somme2
Orig ≜ INSTANCE somme1
Raffinement ≜ Orig!Spec
THEOREM Spec ⇒ Orig!Spec
    
```

Équivalent à $somme2!Spec \Rightarrow somme1!Spec$, où les variables homonymes (*res*) sont fusionnées.

NT

Raffinage – exemple

Somme concrète

MODULE *somme3*

EXTENDS *Naturals*

CONSTANT *N*

VARIABLE *res, acc, i*

$TypeOK \triangleq res \in Nat \wedge acc \in Nat \wedge i \in 1..N$

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

$Next \triangleq \vee i > 0 \wedge acc' = acc + i \wedge i' = i - 1 \wedge UNCHANGED\ res$
 $\vee i = 0 \wedge res' = acc \wedge UNCHANGED\ \langle i, acc \rangle$

$Spec \triangleq Init \wedge \square [Next]_{\langle res, i, acc \rangle} \wedge WF_{\langle res, i, acc \rangle}(Next)$

NT

Raffinage – exemple

Somme3 raffine somme2

MODULE *somme3_raffine_somme2*

EXTENDS *somme3*

$dispMapping \triangleq 1..i$

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

$Raffinement \triangleq Orig!Spec$

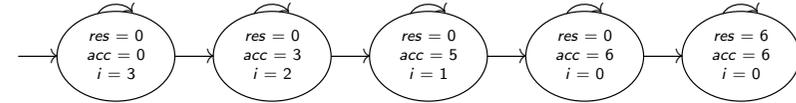
THEOREM $Spec \Rightarrow Orig!Spec$

Équivalent à $somme3!Spec \Rightarrow somme2!Spec$, où les variables homonymes (*res* et *acc*) sont fusionnées, et la variable $somme2!disp$ évolue comme $1..somme3!i$.

NT

Raffinage – exemple

Graphe des exécutions pour $N = 3$



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

NT

Plan

- 1 LTL et TLA⁺
 - Logique TLA⁺
 - Raffinage
- 2 Preuve axiomatique
- 3 Vérification de modèles

NT

Règles de preuve – simple temporal logic

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

$$\overline{\Box F \Rightarrow F} \text{STL2} \quad \overline{\Box \Box F = \Box F} \text{STL3}$$

$$\overline{\Box(F \wedge G) = (\Box F) \wedge (\Box G)} \text{STL5} \quad \overline{\Diamond \Box F \wedge \Diamond \Box G = \Diamond \Box(F \wedge G)} \text{STL6}$$

nf

Règles de preuve – TLA⁺ invariant

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

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

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

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.

nf

Règles de preuve – TLA⁺ vivacité avec équité faible

$$\frac{P \wedge [\mathcal{N}]_v \Rightarrow (P' \vee Q') \quad P \wedge \langle \mathcal{N} \wedge \mathcal{A} \rangle_v \Rightarrow Q' \quad P \Rightarrow \text{ENABLED } \langle \mathcal{A} \rangle_v}{\Box[\mathcal{N}]_v \wedge \text{WF}_v(\mathcal{A}) \Rightarrow (P \rightsquigarrow Q)} \text{WF1}$$

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.

nf

Règles de preuve – TLA⁺ vivacité avec équité forte

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

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)

nf

Règles de preuve dérivées

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

$\Box(P \Rightarrow \Box P)$ 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.

$$\frac{P \rightsquigarrow Q \wedge Q \rightsquigarrow R}{P \rightsquigarrow R} \text{TRANS}$$

Transitivité du \rightsquigarrow .

nf

Plan

- 1 LTL et TLA⁺
 - Logique TLA⁺
 - Raffinage
- 2 Preuve axiomatique
- 3 Vérification de modèles

nf

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 $\Diamond P$, $P \rightsquigarrow Q \dots$: 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.

nf

Complexité

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

nf

Vérificateur TLC

Le vérificateur de modèles TLC sait vérifier :

- les spécifications avec des actions gardées dont l'ordre des variables primées est directement évaluable
($x' = 3 \wedge y' = x' + 1$ est ok, $y' = x' + 1 \wedge x' = 3$ est KO) ;
- (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 ;
- $P \rightsquigarrow Q$ 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 exemple.



Conclusion

- Propriétés de sûreté et de vivacité : LTL (logique temporelle linéaire)
- Équité pour isoler les contraintes de progression
- Vérification mécanisée (par modèle ou par preuve axiomatique)



Plan

Sixième partie

CTL – logique temporelle arborescente

- 1 CTL
 - Syntaxe
 - Sémantique

- 2 Expressivité
 - Exemples
 - Propriétés classiques

Ensemble des exécutions vs arbre des exécutions

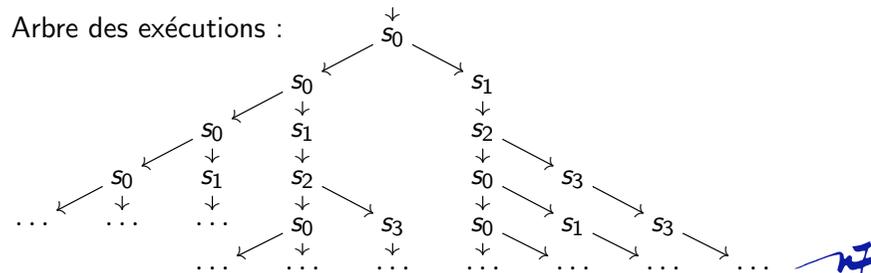


Soit le système de transition :

Ensemble des exécutions :

$$\langle (s_0^+ \rightarrow s_1 \rightarrow s_2)^* \rightarrow s_0^\omega \rangle, \langle (s_0^+ \rightarrow s_1 \rightarrow s_2)^\omega \rangle, \langle (s_0^+ \rightarrow s_1 \rightarrow s_2)^+ \rightarrow s_3^\omega \rangle$$

ou

$$\left\{ \begin{array}{l} s_0 \rightarrow s_0 \rightarrow \dots, s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_0 \rightarrow s_0 \rightarrow \dots, \\ s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_0 \rightarrow \dots, s_0 \rightarrow s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_0 \rightarrow \dots, \dots \\ s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_3 \rightarrow \dots, \dots \end{array} \right\}$$


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

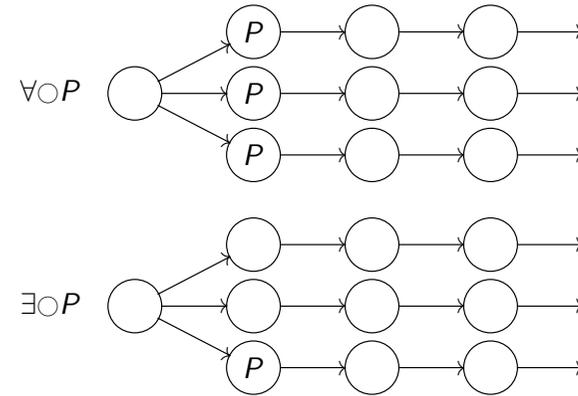
formule	interprétation (pour s un état) pour toute trace partant de s
$\forall \bigcirc P$	P est vrai à l'instant suivant
$\forall \square P$	P est toujours vrai à chaque état
$\forall \diamond P$	P finit par être vrai (dans le futur)
$P \forall U Q$	Q finit par être vrai, et en attendant P reste vrai

Quantification existentielle

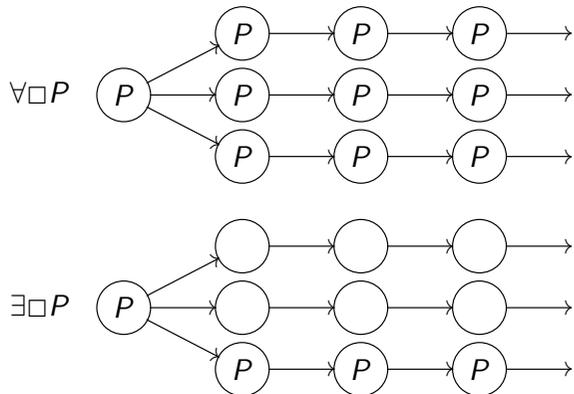
formule	interprétation (pour s un état) pour au moins une trace partant de s
$\exists \bigcirc P$	P est vrai à l'instant suivant
$\exists \square P$	P est toujours vrai à chaque état
$\exists \diamond P$	P finit par être vrai (dans le futur)
$P \exists U Q$	Q finit par être vrai, et en attendant P reste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



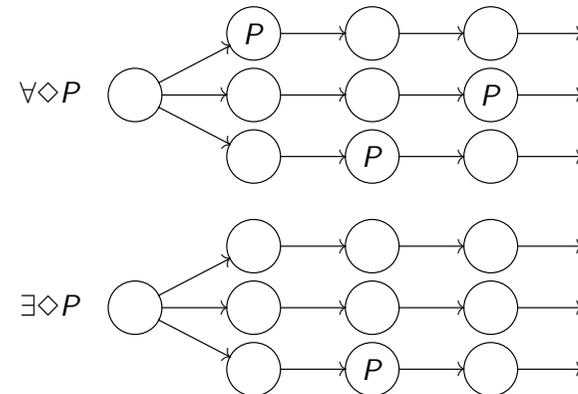
Intuition sémantique $\forall \bigcirc, \exists \bigcirc$



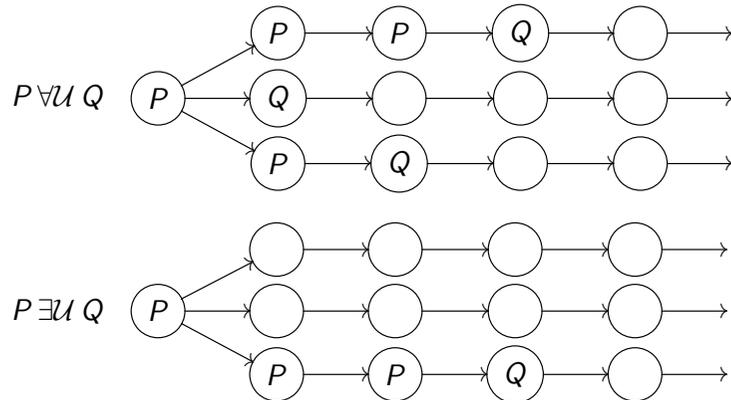
Intuition sémantique $\forall \square, \exists \square$



Intuition sémantique $\forall \diamond, \exists \diamond$



Intuition sémantique $\forall U, \exists U$



Opérateurs minimaux



Un ensemble d'opérateurs minimaux est $\forall \square, \forall U$ et $\exists U$:

- $\exists \square P \triangleq \neg \forall \square \neg P$
- $\forall \diamond P \triangleq True \forall U P$
- $\exists \diamond P \triangleq True \exists U P$
- $\forall \square P \triangleq \neg \exists \diamond \neg P$
- $\exists \square P \triangleq \neg \forall \diamond \neg P$

(autres ensembles minimaux : $\{\exists \square, \exists \square, \exists U\}$ ou $\{\forall \diamond, \exists U, \exists \square\}$)

Syntaxe alternative

Syntaxe alternative

On trouve très fréquemment une autre syntaxe :

- $\forall \leftrightarrow A$ (all)
- $\exists \leftrightarrow E$ (exists)
- $\square \leftrightarrow G$ (globally)
- $\diamond \leftrightarrow F$ (finally)
- $\circ \leftrightarrow X$ (next)

Par exemple :

- $\forall \square \exists \diamond P \leftrightarrow AG EF P$
- $f \forall U g \leftrightarrow A(f U g)$

Opérateur complémentaire waiting-for

- $P \exists W Q \triangleq \exists \square P \vee P \exists U Q$
- $P \forall W Q \triangleq \forall \square P \vee P \forall U Q$ – trop fort
- $\triangleq \neg(\neg Q \exists U(\neg P \wedge \neg Q))$

Sémantique (système)



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

Vérification par un système

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

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

(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)

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

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

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

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

nf

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)

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

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

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

nf

Sémantique (opérateurs temporels dérivés)

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

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

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

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

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

nf

Négation



Négation

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

Négation des formules $\forall, \exists, \square, \diamond$

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

exemples :

$$\neg(\forall \diamond \exists \square p) = \exists \square \forall \diamond \neg p$$

$$(\forall \diamond \neg s_0 \Rightarrow \forall \diamond s_3) = (\exists \square s_0 \vee \forall \diamond s_3) \text{ car } (p \Rightarrow q) = (\neg p \vee q)$$

nf

Définition par point-fixe

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

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

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

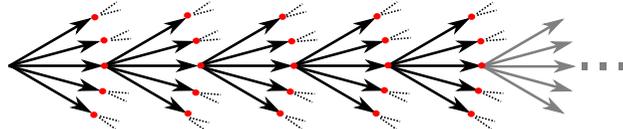
Plan

- 1 CTL
 - Syntaxe
 - Sémantique
- 2 Expressivité
 - Exemples
 - Propriétés classiques

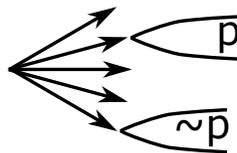
Exemples amusants



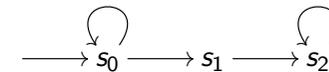
- $\exists\Box\forall\Box p$: une exécution avec une "enveloppe" qui vérifie p



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



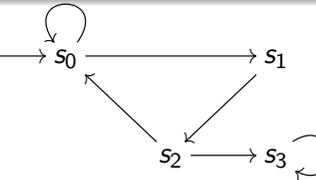
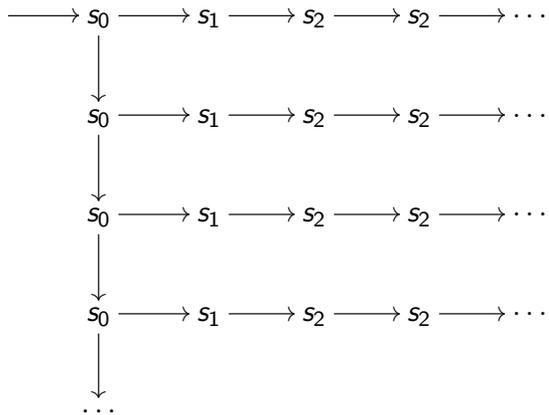
Exemple



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

Exemple - Arbre des exécutions

Arbre des exécutions du système de transition précédent



	pas d'équité	faible (s0, s1)	forte (s2, s3)	forte (s2, s3) faible (s0, s1)
$\exists \Box s_0$				
$\forall \Box \exists \Diamond s_3$				
$\forall \Box \forall \Diamond s_3$				
$\forall \Diamond \forall \Box s_3$				
$\exists \Box s_0 \vee \forall \Diamond s_3$				
$\forall \Diamond \neg s_0 \Rightarrow \forall \Diamond s_3$				

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 $\langle s_1 \rightarrow s_2 \rightarrow \dots \rightarrow s_n \rangle$ est possible.

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

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) :

$$S \models \forall \square (P \Rightarrow \forall \diamond Q)$$

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 :

$$S \models \forall \square (P \Rightarrow P \vee W Q)$$

Spécification d'un ST

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

$$S \models I \wedge \forall \square R$$

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

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

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

qui nécessite une quantification sur une variable.

Combinaisons



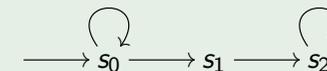
Infiniment souvent

Spécifier que P est infiniment souvent vrai dans toute exécution :
 $S \models \forall \square \forall \diamond P$

Finalement toujours

Spécifier que P finit par rester définitivement vrai :
impossible! $S \models \forall \diamond \forall \square P$ ne convient pas (trop fort)

Soit $S =$



en LTL : $S \models \diamond \square (s_0 \vee s_2)$

mais CTL : $S \not\models \forall \diamond \forall \square (s_0 \vee s_2)$

(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 $\mathcal{X} \in \{\forall \square, \exists \square, \forall \diamond, \exists \diamond\}$

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 $\neg P$	$S \models P \vee S \models \neg P$	$S \models P \vee S \models \neg P$
négation	$S \models \neg P \equiv S \not\models P$	$S \models \neg P \equiv S \not\models P$
l'un de P ou Q inévitable	$S \models \forall \diamond P \vee \forall \diamond Q$ $S \models \forall \diamond (P \vee Q)$	$S \models \diamond P \vee \diamond Q$
l'un de P ou Q continu	$S \models \forall \square (P \vee Q)$ $S \models \forall \square P \vee \forall \square Q$	$S \models \square P \vee \square Q$
$\neg P$ transitoire	$S \models \forall \diamond \forall \square P$	$S \models \diamond \square P$
répétition	$S \models \forall \diamond (P \wedge \forall \square P)$	$S \models \diamond (P \wedge \square P)$
possibilité	$S \models \exists \diamond P$	$S \models \diamond P$

Conséquence : l'équité n'est pas exprimable en CTL. Mais on peut vérifier des propriétés CTL sur un ST avec 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 $\square, \diamond, \circ, \mathcal{U}$.

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).



Objectif

Septième partie

Vérification de propriétés

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

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



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.

→ **système de transition 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 (*Anciens*);
- mémoriser les nouveaux états à explorer (*Nouveaux*) : 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 = $Init \wedge \square[Next]$

```

procédure 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

```

procédure 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, polyèdres...
- mixte : explicite + fingerprint (TLA⁺)

Ordre global sur les états nouveaux

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



Syntaxe alternative (rappel)

Syntaxe alternative

A (all)	$\leftrightarrow \forall$
E (exists)	$\leftrightarrow \exists$
G (globally)	$\leftrightarrow \square$
F (finally)	$\leftrightarrow \diamond$
X (next)	$\leftrightarrow \circ$
$A(f \text{ U } g)$ ou $f \text{ AU } g$	$\leftrightarrow f \forall U g$
$E(f \text{ U } g)$ ou $f \text{ EU } g$	$\leftrightarrow f \exists U g$

Par exemple : $AG \ EF \ f \leftrightarrow \forall \square \exists \diamond f$



Plan

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



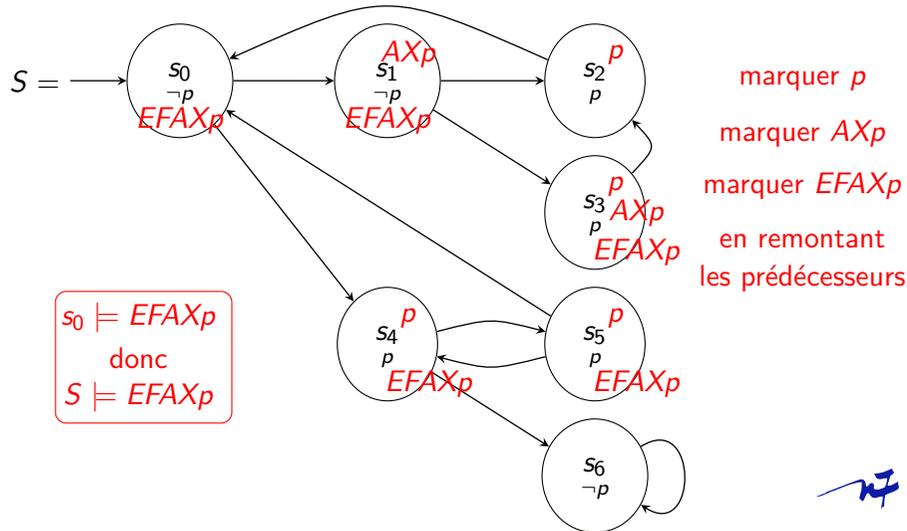
Principe

- 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 .



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 $F = p$ 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 $F = F_1 \wedge F_2$, 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 $F = H EU K$, idée : $H EU K \equiv K \vee (H \wedge EX(H EU K))$

```
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 $F = H \text{ AU } K$, idée : $H \text{ AU } K \equiv K \vee (H \wedge AX(H \text{ AU } K))$
 $mark(H)$; $mark(K)$;
 $L := \emptyset$;
 pourtout $s \in S$:
 $s.F := s.K$; $s.nb := card(succ(s))$;
 si $s.F$ alors $L := L \cup \{s\}$;
 tantque $L \neq \emptyset$ faire:
 $s := \text{choose any in } L$; // ordre sans importance
 $L := L \setminus \{s\}$;
 pourtout $t \in pred(s)$:
 $t.nb := t.nb - 1$;
 si $t.nb = 0$ et $t.H$ et $\neg t.F$ alors
 $t.F := true$;
 $L := L \cup \{t\}$;
 finsi
 finpour
 fintq

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

CTL équitable

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

- Contraintes d'équité : un ensemble d'ensemble d'états F_1, \dots, 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 :

$$M \models_f \phi \triangleq M \models fair \Rightarrow \phi \text{ avec } fair = \bigwedge_{1 \leq i \leq n} \square \diamond F_i$$

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

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 $s \models EX(\phi \wedge fair)$
- $s \models_f AX\phi$ ssi $s \models AX(fair \Rightarrow \phi)$
- $s \models_f EG\phi$ est compliqué
- $s \models_f AG\phi$ ssi $s \models AG(fair \Rightarrow \phi)$
- $s \models_f \phi EU\psi$ ssi $s \models \phi EU(fair \wedge \psi)$
- $s \models_f \phi AU\psi$ 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 : $S'' = \{u \in S : u \rightarrow^* v \text{ où } v \in S'\}$
 (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*.

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)

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)

Plan

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

Réduction du problème

Dans un système ayant un nombre fini d'états, la seule solution pour obtenir une exécution infinie est de finir par boucler sur un cycle.

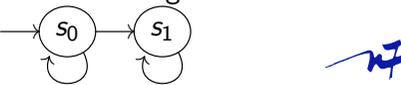
→ On se ramène à un problème de traces ω -régulières : un préfixe puis un cycle (un lasso).

Traces ω -régulières

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

$$\begin{aligned} \exists \sigma \in S^\omega : \sigma \in Exec(S) \wedge \sigma \models L \\ \Leftrightarrow \exists \sigma_p, \sigma_c \in S^* : \langle \sigma_p \rightarrow \sigma_c^\omega \rangle \in Exec(S) \wedge \langle \sigma_p \rightarrow \sigma_c^\omega \rangle \models L \end{aligned}$$

Même ainsi, il peut exister une infinité de traces ω -régulières : une infinité de préfixes distincts. Considérer



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.)

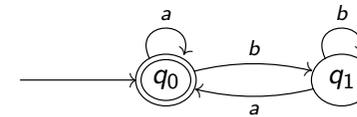
Automate de Büchi

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.

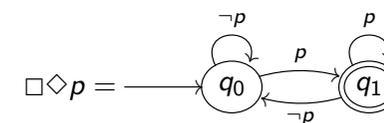
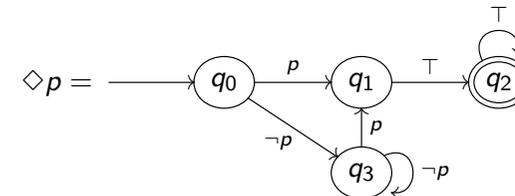
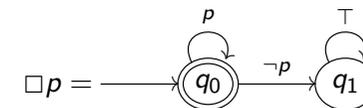
Un mot infini est accepté si sa reconnaissance visite infiniment F .



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

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 $\mathcal{M} \models \varphi$; 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 $\Diamond\Box(\text{ENABLED } A) \Rightarrow \Box\Diamond A$) → directement pris en compte

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

Plan

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

Exemple : xyplus1 (spécification)

```

MODULE xyplus1
EXTENDS Naturals, FiniteSetTheorems, TLAPS
VARIABLES x, y

TypeInv ≜ (x ∈ Nat ∧ y ∈ Nat)
Ecart ≜ (0 ≤ x − y ∧ x − y ≤ 1)

Init ≜ x = 0 ∧ y = 0
Act1 ≜ x' = y + 1 ∧ UNCHANGED ⟨y⟩
Act2 ≜ y' = x ∧ UNCHANGED ⟨x⟩
Spec ≜ Init ∧ □[Act1 ∨ Act2]⟨x, y⟩
```

Exemple : xyplus1 (preuves)

```

MODULE xyplus1
THEOREM TypelnvOK  $\triangleq$  Spec  $\Rightarrow$   $\square$  Typelnv
  (1)1 Init  $\Rightarrow$  Typelnv BY DEF Init, Typelnv
  (1)2 (Act1  $\vee$  Act2)  $\wedge$  Typelnv  $\Rightarrow$  Typelnv' BY DEF Act1, Act2, Typelnv
  (1)3 UNCHANGED  $\langle x, y \rangle \wedge$  Typelnv  $\Rightarrow$  Typelnv' BY DEF Typelnv
  (1)4 QED BY PTL, (1)1, (1)2, (1)3 DEF Typelnv, Spec
THEOREM EcartOK  $\triangleq$  Spec  $\Rightarrow$   $\square$  Ecart
  (1)1 Init  $\Rightarrow$  Ecart BY DEF Init, Ecart
  (1)2 (Act1  $\vee$  Act2)  $\wedge$  Typelnv  $\wedge$  Ecart  $\Rightarrow$  Ecart'
    (2)1 Act1  $\wedge$  Typelnv  $\wedge$  Ecart  $\Rightarrow$  Ecart' BY DEF Act1, Typelnv, Ecart
    (2)2 Act2  $\wedge$  Typelnv  $\wedge$  Ecart  $\Rightarrow$  Ecart' BY DEF Act2, Typelnv, Ecart
    (2)3 QED BY (2)2, (2)1
  (1)3 UNCHANGED  $\langle x, y \rangle \wedge$  Ecart  $\Rightarrow$  Ecart' BY DEF Ecart
  (1)4 QED BY PTL, (1)1, (1)2, (1)3, TypelnvOK DEF Ecart, Spec, Typelnv
THEOREM Spec  $\Rightarrow$   $\square$ (Typelnv  $\wedge$  Ecart) BY TypelnvOK, EcartOK

```

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

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

Huitième partie

Conclusion

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 transition 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