LOG8470
Méthodes formelles en fiabilité et sécurité

I. Vérification des propriétés des systèmes de transitions

John Mullins

Dép. de génie informatique et de génie logiciel
École Polytechnique de Montréal

John.Mullins@polymtl.ca

2018 - 2019

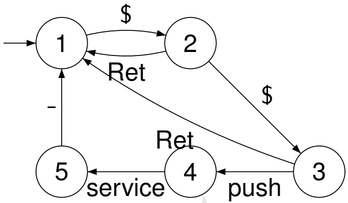
POLYTECHNIQUE MONTRÉAL

Plan de la partie I

Plan de la partie I

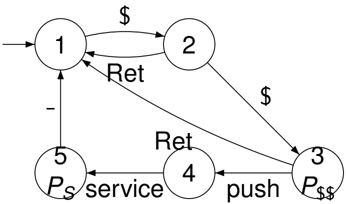
Model-checking : un exemple introductif (1)

Une machine à café



Model-checking : un exemple introductif (2)

La machine à café avec les propriétés atomiques



- $P_{\$\$}$: 2 pièces sont dans la machine (vrai dans l'état 3 seulement)
- P_S : La machine sert un café (vrai dans l'état 5 seulement)
- Chaque fois qu'on met 2 pièces, alors, fatalement , un café est servi

Système de transition

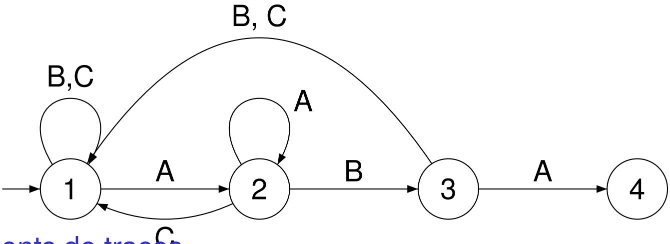
Un *système de transitions* est un triplet $\mathcal{A} = (S, S_0, T, \Sigma, \rho)$ où

- S est un ensemble d'états
- S_0 est un ensemble d'états initiaux
- Σ est un ensemble d'actions
- $T \subseteq S \times \Sigma \times S$ est l'ensemble des transitions
- Σ : un alphabet d'actions
- $\rho : S \rightarrow 2^{Prop}$ est une fonc. d'interprétation des propriétés atomiques $Prop = \{P_1, P_2, \ldots P_n\}$

Notation :

- Transition : $s_1 \xrightarrow{a} s_2$
- Fragment d'exécution : $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \cdots$
- Trace : $\rho(s_1)\rho(s_2)\rho(s_3)\ldots$

Un digicode



Fragments de traces

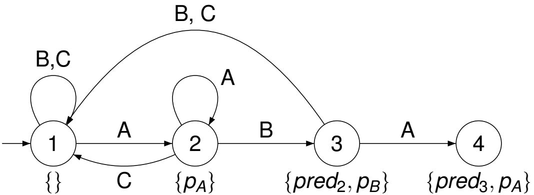
1

11, 12

111, 112, 121, 122, 123

1111, 1112, 11121, 1122, 1123, 1211, 1212, 1221, 1222, 1223, . . .

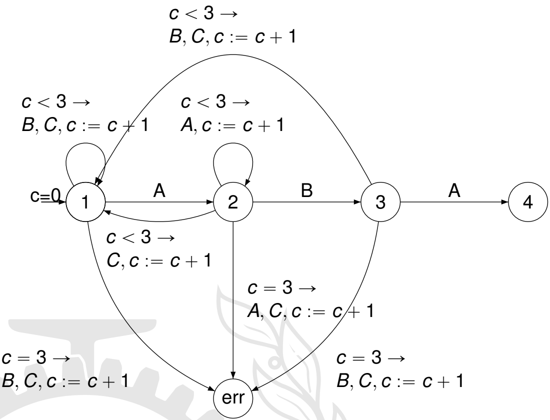
Digicode avec les propriétés atomiques



- P_A : un A vient tout juste d'être tapé
- P_B : un B vient tout juste d'être tapé
- P_C : un C vient tout juste d'être tapé
- $pred_2$: l'état précédent de toute exécution est 2
- $pred_3$: l'état précédent de toute exécution est 3
- Si la porte s'ouvre alors ABA était la dernière séquence tapée
- Toute séquence terminant par ABA ouvre la porte

Digicode avec transitions gardées

- Variable c
- Initialement : c=0
- Transitions gardées
- Actions $c := c + 1$ incrémentent c



Systèmes de transitions étendu

Un *systèmes de transitions étendu* sur un ensemble de variables typées Var est un tuplet

$$\mathcal{AE} = (S, S_0, \mathcal{T}, \Sigma, \mathit{Effet}, g_0)$$

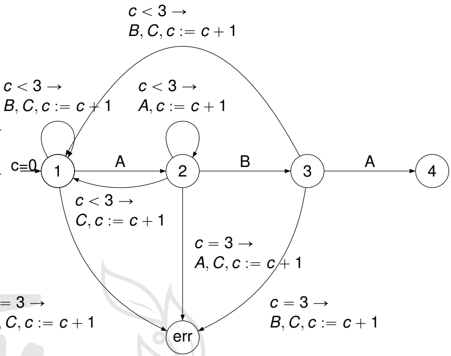
- S est un ensemble fini d'états de contrôle appelées locations ;
- $S_0 \in Q$ est l'ensemble des locations initiales ;
- Σ est un ensemble d'actions ;
- $\mathit{Effet} : \Sigma \times \mathit{Eval}(Var) \rightarrow \mathit{Eval}(Var)$ est la fonction d'effets où

  $$\mathit{Eval}(Var) = \bigtimes_{x \in Var} \mathit{Dom}(x)$$

  et $\mathit{Dom}(x)$, le domaine des valeurs de x ;
- $\mathcal{T} \subseteq S \times \mathit{Cond}(Var) \times \Sigma \times S$ est un ensemble fini de transitions où $\mathit{Cond}(Var)$ est l'ensemble des conditions booléennes sur Var ;
- $g_0 \in \mathit{Cond}(Var)$ est la condition initiale.

Digicode avec transitions gardées

$$
\begin{aligned}
Var &= \{c\} \\
Dom(c) &= \{0,1,2,3,4\} \\
Eval(Var) &= \{0,1,2,3,4\} \\
Effet(c := c+1, \sigma) &= \sigma + 1 \\
g_0 &\equiv c = 0
\end{aligned}
$$



Digicode avec compteur d'erreurs

- Configuration :

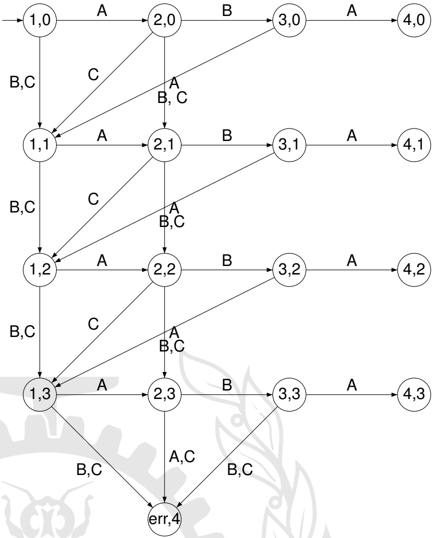
$$(\text{location}, \text{Eval. de } c)$$

- Effet de $c := c + 1$ sur la valeur de c

$$(s, n) \xrightarrow{a} (s', n+1)$$

si

1. g est valide en n
2. $s \xrightarrow{g \rightarrow a,\, c := c+1} s'$



Dépliage d'un système de transitions étendu

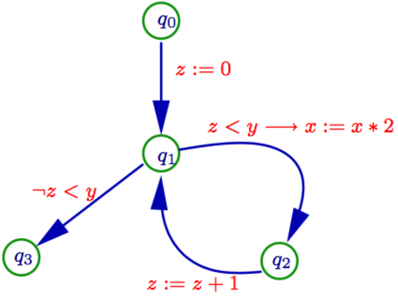
Le *dépliage* d'un systèmes de transitions étendu sur un ensemble de variables typées *Var*

$$\mathcal{AE} = (S, S_0, \mathcal{T}, \Sigma, \mathit{Effet}, g_0)$$

est le système de transitions tel que

- États : $S \times \mathit{Eval}(\mathit{Var})$;
- États initiaux : $S_0 \times \{\sigma \in \mathit{Eval}(\mathit{Var}) : g_0$ est valide dans $\sigma\}$;
- $(s, \sigma) \xrightarrow{a} (s', \sigma')$ est une transition si :
  - $s \overset{g \to}{\underset{}{\xrightarrow{a,\, c\, :=\, c+1}}} s'$
  - g est valide dans σ (noté $\sigma \models g$)
  - $\sigma' = \mathit{Effet}(c, \sigma)$

Exemple à plusieurs variables



Un fragment d'exécution du système de transition (infini) associé :

$$
\begin{array}{llll}
(q_0, [3/x, 2/y, 10/z]) & \xrightarrow{z:=0} & (q_1, [3/x, 2/y, 0/z]) & \xrightarrow{x:=2x} \\
(q_2, [6/x, 2/y, 0/z]) & \xrightarrow{z:=z+1} & (q_1, [6/x, 2/y, 1/z]) & \xrightarrow{x:=2x} \\
(q_2, [12/x, 2/y, 1/z]) & \xrightarrow{z:=z+1} & (q_1, [12/x, 2/y, 2/z]) & \xrightarrow{-} \\
(q_3, [12/x, 2/y, 2/z]) & & &
\end{array}
$$

Produit libre de systèmes de transitions

Le *produit libre* $\mathcal{A}_1 \times \cdots \mathcal{A}_n$ de n systèmes de transitions $\mathcal{A}_i = (S_i, S_{0,i}, T, \Sigma_i, \rho_i)$ pour tout $1 \leqslant i \leqslant n$*et* est le système de transition $\mathcal{A} = (S, S_0, T, \Sigma, \rho)$ défini par

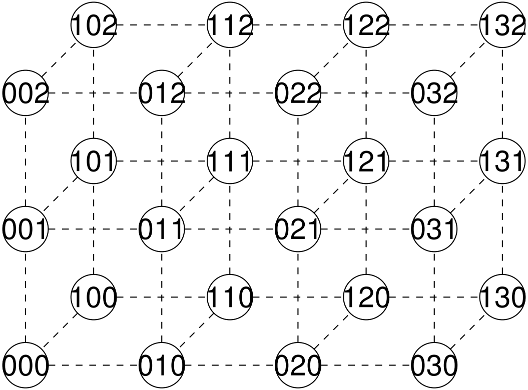
- $S = S_1 \times \cdots \times S_n$
- $S_0 = S_{0,1} \times \cdots \times S_{0,n}$
- $T = \left\{ \begin{array}{r} (s_1, \ldots, s_n) \stackrel{(a_1,\ldots,a_n)}{\longrightarrow} (s'_1, \ldots, s'_n) : \text{ pour tout i,} \\ s_i \stackrel{a_i}{\longrightarrow} s'_i \in T_i \text{ et } a_i \neq' \_' \text{ ou} \\ a_i =' \_' \text{ et } s_i = s'_i \end{array} \right\}$
- $\Sigma = (\Sigma_1 \cup \{\_\}) \times \cdots (\Sigma_n \cup \{\_\})$
- $\rho = \bigcup_{1 \leqslant i \leqslant n} \rho_i$

Exemple : $0 \xrightarrow{a} 1 \times 2 \xrightarrow{b} 3$

$$
\begin{tikzcd}
 & 0 \arrow[r, "a"] & 1 \\
2 \arrow[d, "b"'] & (0,2) \arrow[r, "{a,\_}"] \arrow[d, "{\_,b}"'] \arrow[dr, "{a,b}"] & (1,2) \arrow[d, "{\_,b}"] \\
3 & (0,3) \arrow[r, "{a,\_}"'] & (1,3)
\end{tikzcd}
$$

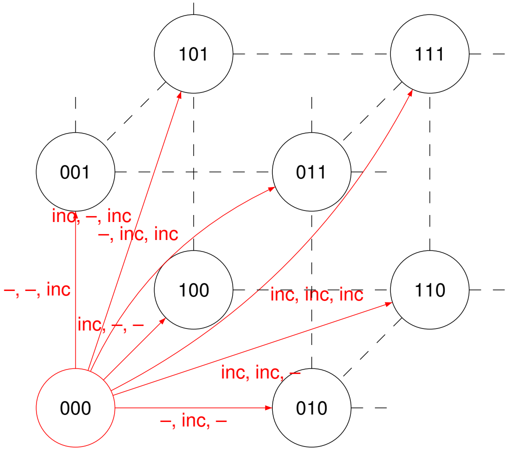
3 compteurs modulo : $\mathcal{A}_{\text{mod } 2} \times \mathcal{A}_{\text{mod } 4} \times \mathcal{A}_{\text{mod } 3}$

Espace des états



3 compteurs modulo : $\mathcal{A}_{\text{mod } 2} \times \mathcal{A}_{\text{mod } 4} \times \mathcal{A}_{\text{mod } 3}$

Transitions issues de $(0, 0, 0)$



Produit synchronisé des systèmes de transitions

Definition

Une *contrainte de synchronisation Sync* $\subseteq (\Sigma_1 \cup \{\_\}) \times \cdots \times (\Sigma_n \cup \{\_\})$ établit l'ensemble de toutes les actions globales permises, toutes les autres étant interdites.

Definition

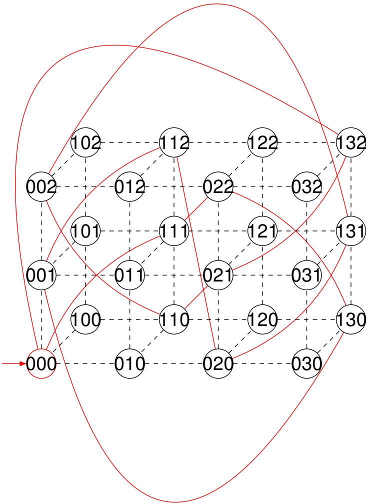
Le *produit synchronisé* $\mathcal{A}_1 \| \cdots \| \mathcal{A}_n$ de $\mathcal{A}_1, \ldots \mathcal{A}_n$ par rapport à *Sync* est le système de transition $\mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ restreint aux seules transitions étiquetées par une action de *Sync*

Example

$0 \xrightarrow{a} 1 \parallel 2 \xrightarrow{b} 3 = (0,2) \xrightarrow{a,b} (1,3)$ sur *Sync* $= \{(a,b)\}$

Exemple : $\mathcal{A}_{\text{mod } 2} \times \mathcal{A}_{\text{mod } 4} \times \mathcal{A}_{\text{mod } 3}$

$Sync = \{(\mathtt{inc}, \mathtt{inc}, \mathtt{inc}), (\mathtt{dec}, \mathtt{dec}, \mathtt{dec})\}$



Mécanismes de communication

Entrelacement des exécutions

Le modèle de calcul de la concurrence doit être :

- indépendant de la vitesse relative des processeurs
- Modèle indépendant de leurs distances relatives

Definition

Le *produit d'entrelacement* $\mathcal{A}_1 \|_I \mathcal{A}_2$ de de deux systèmes de transitions $\mathcal{A}_1, \mathcal{A}_2$ est le système de transition $\mathcal{A}_1 \| \mathcal{A}_2$ avec $Sync = \Sigma_1 \times \{\_\} \cup \{\_\} \times \Sigma_2$.

Mécanismes

- Communication par variables partagées
- Communication par messages synchrone (*handshaking*)
- Communication par canal (par messages asynchrone)

Communication par variables partagées

Le problème

- Le résultat de l'entrelacement d'affectations à une variables partagée dépend de l'ordre d'exécution :

Exemple

L'exécution de

$$\underbrace{x := 2x}_{\text{action } \alpha} \parallel \underbrace{x := x + 1}_{\text{action } \beta}$$

de l'état initial $x = 3$ termine dans l'état

- $x = 7$ si α est exécutée avant β
- $x = 8$ si β est exécutée avant α

Communication par variables partagées

La solution

- Entrelacement des actions
- Résolution de conflit entre les variables critiques par un arbitre

Remarque

- L'exécution d'affectations à des variables indépendantes est indépendant de l'ordre d'entrelacement

Exemple

L'exécution de

$$\underbrace{x := x + 1}_{\text{action } \alpha} \parallel \underbrace{y := y - 2}_{\text{action } \beta}$$

de l'état initial $x = 0, y = 7$

Exemple : Un gestionnaire d'imprimante

- Var. partagée : `turn` (initialisée à A)

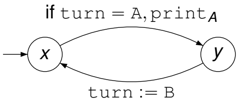


FIGURE: L'utilisateur A.

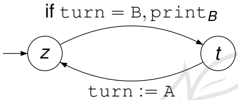


FIGURE: L'utilisateur B.

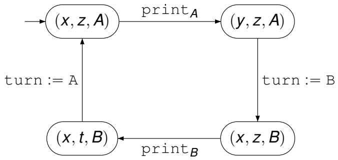


FIGURE: Système de transition associé à $\mathcal{AE}_A \|_I \mathcal{AE}_B$.

Exclusion mutuelle à l'imprimante

Aucun état accessible de la forme $(y, t, \_)$

L'algorithme d'exclusion mutuelle de Peterson

- Var. partagées : `d0`, `d1` (init. à `0`)
  Le processus P_0 exécute :

```
loop forever
     begin
     n0 : {section non critique}
     d0:=1;
     tour:=1;
     w0 : wait until (d1=0 or tour=0)
     c0 : {section critique}
     d0:=0
     end
```

P_1 exécute le code symétrique

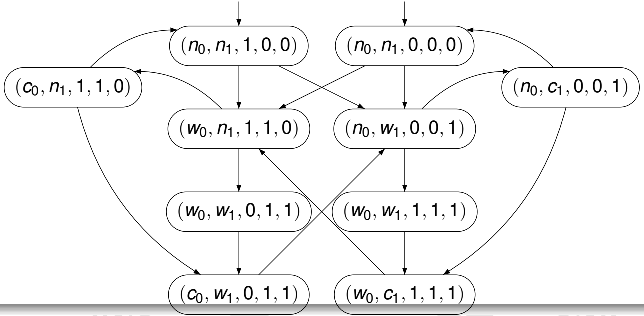
```
loop forever
     begin
     n1 : {section non critique}
     d1:=1;
     tour:=0;
     w1 : wait until (d0=0 or tour=1);
     c1 : {section critique}
     d1:=0
     end
```

L'algorithme d'exclusion mutuelle de Peterson

États de $\mathcal{P}_{Pet}$

$S_0 \times S_1 \times dom(tour) \times dom(d_0) \times dom(d_1)$ (72 états)

États accessibles



Exclusion mutuelle

Aucun état accessible de la forme $(c_0, c_1, \_, \_, \_)$

Communication par message (*Handshaking*)

Definition

Le *produit de* handshake $\mathcal{A}_1 \|_H \mathcal{A}_2$ de de deux systèmes de transitions $\mathcal{A}_1, \mathcal{A}_2$ où $H \subseteq \Sigma_1 \cap \Sigma_2$ sont les actions de *handshake*, est le système de transition $\mathcal{A}_1 \| \mathcal{A}_2$ avec

$$Sync = (\Sigma_1 \backslash H) \times \{\_\} \cup \{\_\} \times (\Sigma_2 \backslash H) \cup \{(a, a) : a \in H\}.$$

$\|_H$:

Cas particulier de $\|$ où la synchronisation est simultanée pour les actions de H et entrelacée pour les autres.

Exemple : Exclusion mutuelle avec un contrôleur

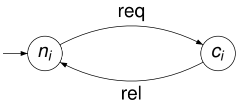


FIGURE: Processus P_i.

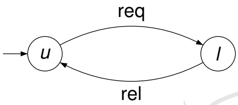


FIGURE: Le contrôleur.

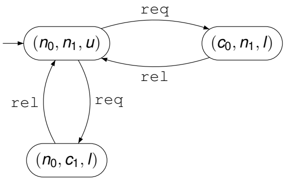


FIGURE: $(\mathcal{AE}_0 \|_I \mathcal{AE}_1) \|_H \mathcal{AE}_C$.

Exclusion mutuelle

Aucun état accessible de la forme $(c_0, c_1, \_)$

Communication par canal

Variable c de canal

- domaine : $dom(c)$ Ex. : Mots $\leqslant 200$ d'un alphabet $\mathcal{M}$
- capacité : $cap(c)$ Ex. : 200
- opérations : $\mathtt{c}!v$ (enfiler v si non plein), $\mathtt{c}?x$ (defiler si non vide)

- *Chan* : Un ensemble de variables de canal (FIFO)
- Actions supplémentaires :

$$\begin{aligned} Comm \quad = \quad & \{\mathtt{c}!v : v \in dom(c) \text{ et } c \in Chan\} \cup \\ & \{\mathtt{c}?x : dom(c) \subseteq dom(x), x \in Var \text{ et } c \in Chan\} \end{aligned}$$

Communication par canal : sémantique

$$\mathcal{AE} = (S, S_0, \mathcal{T}, \Sigma, Effet, g_0)$$

sur $Var \cup Chan$ avec

$$\mathcal{T} \subseteq S \times Cond(Var) \times (\Sigma \cup Comm) \times S$$

comme avant avec les effets supplémentaires :

- $s \xrightarrow{g;c!v} s'$: enfile v sur c si non plein
- $s \xrightarrow{g;c?x s'}$: défile et affecte à x si non vide)

Modèle de système concurrent général

$$\mathcal{AE}_i = (S_i, S_0^i, \mathcal{T}_i, \Sigma_i, Effet_i, g_0^i)$$

sur $(Var_i \cup Chan)$ et $Var = Var_0 \cup Var_1$ alors le système de transition associé à

$$\mathcal{AE}_0 | \mathcal{AE}_1$$

- $S = S_0 \times S_1 \times Eval(Var) \times Eval(Chan)$ où

$$\begin{aligned} Eval(Var) &= \bigtimes_{x \in Var} Dom(x) \\ Eval(Chan) &= \bigtimes_{c \in Chan} Dom(c) \end{aligned}$$

- $$\begin{aligned} S_0 &= \{(s_o, s_1, \sigma, \xi) : s_0 \in S_0, s_1 \in S_1, \sigma \models g_0^0 \wedge g_0^1, \\ &\quad \xi : \text{ tous les canaux vides}\} \end{aligned}$$

Modèle de système concurrent général (suite)

- $\mathcal{T}$ restreint aux actions de
  - Σ_i et aux affectations se comporte selon $\|_I$;
  - la forme $c!v$ et $c?x$ où $cap(c) > 0$ se comporte selon les règles d' échanges de messages asynchrones ;
  - la forme $c!v$ et $c?x$ où $cap(c) = 0$ se comporte selon $\|_H$
-

$$\begin{aligned}\Sigma \quad = \quad & \Sigma_0 \cup \Sigma_1 \cup \\ & \{x := e : x \in Var, e, \text{ une expresion}\} \cup \\ & \{c!v, c?x : c \in Chan, v \in dom(c), x \in Var, dom(c) \subseteq dom(x)\}\end{aligned}$$

Le protocole de Dolev-Klawe-Rodeh

- On suppose $N \geqslant 2$ processus dans une topologie en anneau connectés par des canaux non-bornés
- Émettent ds le sens des aiguilles d'une montre
- Chacun a un *identificateur* unique (ici un entier) représentant sa capacité d'exercer une certaine tâche (gardé dans d)
- But : Assurer que l'élection élira un et un seul processus (le plus apte) i.e. avec le plus grand *identificateur*

Le protocole de Dolev-Klawe-Rodeh (suite)

- Les participants sont ou actif ou relai
- Initialement : tous actif
- À chaque ronde tous les processus actifs reçoivent les identificateurs des deux voisins les plus près (dans la direction entrante) (e et f).
- Un processus demeure actif seulement si la valeur du plus près est la plus grande des 3 (entre d, e et f). Le processus adopte alors ce nombre comme le sien.
- À la sortie : 1 et 1 seul actif, le plus grand

Exemple

En classe

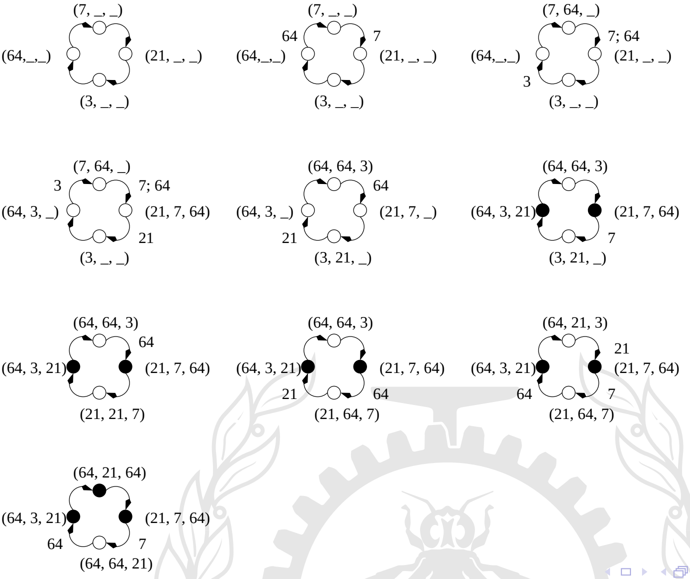
Modèle du protocole de Dolev-Klawe-Rodeh

```
eligible :
d :=ident ;
do forever
begin
        send(d) ;
        receive(e) ;
        if e=d then goto stop (* le processus d est le leader*) else skip
        send(e) ;
        receive(f) ;
        if e ⩾ max (d, f) then d := e else goto relai ;
end

relai :
do forever
begin
        receive(d) ;
        send(d) ;
end

stop : skip
```

Une exécution synthétique de Dolev-Klawe-Rodeh



Le langage PROMELA

- SPIN : Outil de simulation et de vérification d'une spécification
- PROMELA : Langage d'entrée de SPIN
- Syntaxe des commandes :

$$
\begin{array}{rcl}
S & ::= & \textbf{skip} \mid x := e \mid c?x \mid c!e \mid \\
 & & S_1 ; S_2 \mid \texttt{atomic}\{S\} \mid \\
 & & \textbf{if} \ :: g_1 \Rightarrow S_1 \ldots :: g_n \Rightarrow S_n \ \textbf{fi} \mid \\
 & & \textbf{do} \ :: g_1 \Rightarrow S_1 \ldots :: g_n \Rightarrow S_n \ \textbf{od}
\end{array}
$$

- Programme PROMELA : une suite de processus $P_1, \ldots P_n$ modélisant un réseau

$$\mathcal{PE}_1 | \ldots | \mathcal{PE}_n$$

Le langage PROMELA

- La déclaration

```
proctype P (parametres formels) {declarations locales;
 enonces}
```

- L'instanciation

```
init{ run P(parametres reels); run Q(parametres reels); ...}
```

- Basé sur les *commandes gardées* de Dijkstra
  - Toute commande est soit activée soit bloquée :
    `(a == b)` est équivalente à `while (a != b) do skip`
  - La commande vide : `skip`
  - L'affectation : `x = 7`
  - La conditionnelle non-déterministe `if` :

```
if :: guarde1 -> commande1
   ::  ...........
   :: guarden -> commanden
fi
```

Le langage PROMELA

- Basé sur les *commandes gardées* de Dijkstra (suite)
  - La boucle non-déterministe do :

```
do :: guarde1 -> commande1
   ::  ............
   :: guarden -> commanden
od
```

  - Les commandes send et receive
    - Un canal fini : `chan c = [5] of byte`
    - L'émission ! :
      c!2 est activé seulement si c n'est pas plein
    - réception ? :
      c?x est activée seulement si c n'est pas vide
    - Un canal de capacité nulle (*rendez-vous*) :
      c! (resp. c?) est activé si et seulement si c? (resp. c!) l'est

Le processus P_0 de l'exclusion mutuelle

```
do :: true -> skip;
              atomic{d0 = 1; t = 1};
              if :: (d1 == 0) || t == 0) ->
                    d0 = 0;
                fi;
od;
```

Le protocole de Dolev-Klawe-Rodeh

```
proctype processus (chan in, out; byte ident)
{       byte d, e, f;
        printf("MSC: %d\n", ident);

eligible:
        d = ident;
        do :: true -> out!d;
                      in?e;
                      if :: (e == d) ->
                             printf("MSC: %d is LEADER\n", d);
                             nbre_leaders = nbre_leaders + 1;
                             goto stop /* d est le leader */
                         :: else -> skip
                        fi;
                        out!e;
                        in?f;
                        if :: (e >= d) && (e >= f) -> d = e
                           :: else -> goto relai
                        fi
        od;

relai:
end:
        do :: in?d -> out!d
        od;

stop:
        skip
}
```

Le protocole de Dolev-Klawe-Rodeh

- Définition des variables globales :

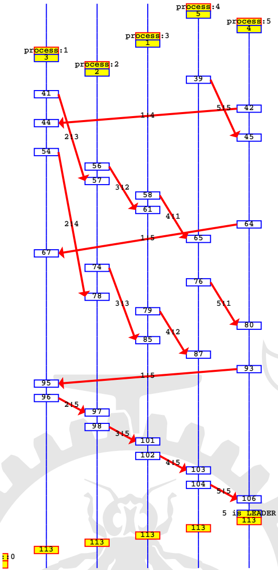
```
#define N        5   /* nombre de processus */
#define I        3   /* processus avec le plus faible ident */
#define L        10  /* dimension du buffer (>= 2*N) */#

chan q[N] = [L] of {byte}; /* N canaux de capacité L chacun */
```

- Création de l'anneau et instanciation des processus

```
init {
    byte i;
    atomic {
      i = 1;
      do :: i <= N -> run processus(q[i-1],q[i%N],(N+I-i)%N+1);
                                       i = i+1
           :: i > N   -> break
      od
            }
      }
```

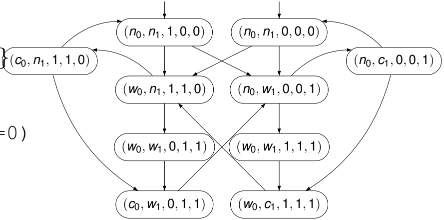
Le protocole de Dolev-Klawe-Rodeh



Plan de la partie I

Définition

loop forever
begin
n_0 : {section non critique}
d0:=1;
tour:=1;
w_0 : **wait until** (d1=0 or tour=0)
c_0 : {section critique}
d0:=0
end



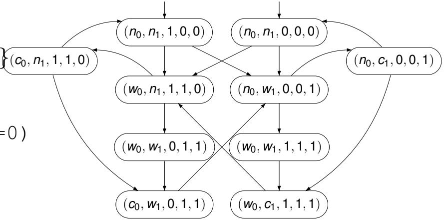
- Exécution $\pi = ((n_0, n_1, 0, 0) \cdot (w_0, n_1, 1, 1, 0) \cdot (c_0, n_1, 1, 1, 0))^\omega$
- exclusion mutuelle : Au plus un seul processus doit être dans sa section critique à un instant donné.
- $Prop = \{P_{sc0}, P_{sc1}\}$
- Trace de $\pi = (\emptyset \cdot \emptyset \cdot \{P_{sc0}\})^\omega = ((0,0) \cdot (0,0) \cdot (1,0))^\omega$
- Une propriété temporelle est un ω-langage $P \subseteq (\mathbb{B}^n)^\omega$

Exemple

$$P_{mutex} = \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right)^\omega$$

Un autre exemple

loop forever
begin
n_0 : {section non critique}
d0:=1;
tour:=1;
w_0 : **wait until** (d1=0 or tour=0)
c_0 : {section critique}
d0:=0
end



- Absence de blocage : Si les 2 processus essaient d'entrer en section critique alors un des 2 finira par y parvenir.
- Absence de famine : Si un processus essaie d'entrer dans sa section critique alors il y parviendra.
- Attente bornée : Si un processus attend pour accéder à sa SC, alors il y arrivera et on peut borner le nombre de fois où d'autres processus pourront accéder à leur SC avant lui.
- $Prop = \{P_{w0}, P_{sc0}, P_{sc1}, P_{w1}\}$
- Trace de π :

$$(\emptyset \cdot \{P_{w0}\} \cdot \{P_{sc0}\})^\omega = ((0,0,0,0) \cdot (1,0,0,0) \cdot (0,1,0,0))^\omega$$

Propriétés de sûreté

Definition

$P \subseteq (\mathbb{B}^n)^\omega$ est une propriété de sûreté ssi toute trace σ qui viole P contient un “mauvais” préfixe $\hat{\sigma}$

P_{mutex}

Le langage (régulier) des mauvais préfixes est

$$\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix}\right)^* \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Vivacité

Definition

$P \subseteq (\mathbb{B}^n)^\omega$ est une propriété de vivacité ssi n'importe quel mot de $(\mathbb{B}^n)^*$ peut être étendu en un mot (infini) de P.

- "Quelque chose de bien va arriver"
- Tout préfixe peut être étendu à un mot de P.

Example

Absence de famine

En général

Soit P, une propriété alors

$$\exists_{P_{safe}, P_{live}} P = P_{safe} \cap P_{live}$$

où P_{safe} est une propriété de sûreté et P_{live}, une propriété de vivacité.

Example

Le système dont les traces sont spécifiées par

$$(e^* ae^* b)^* e^\omega \cup (e^* ae^* b)^\omega$$

satisfait la propriété :

1. toute occurrence de a doit être suivie de b (vivacité) et
2. toute occurrence de b doit être précédée de a (sûreté).

Formules temporelles linéaires

- Un ensemble de propriétés atomiques :
  $Prop = \{P_0, P_1, P_2, \ldots, P_n\}$
- Pour $P \in Prop$,

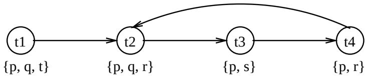
$$\phi, \psi ::= P \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \phi \leftrightarrow \psi \mid \mathbf{N}\phi \mid \phi \mathbf{U} \psi$$

- $\sigma \in (\mathbb{B}^n)^\omega$, une chaîne infinie d'interprétations de *Prop*
  - $\sigma \models P$ ssi $\sigma(0) \models P$
  - $\sigma \models \phi \wedge \psi$ ssi $\sigma \models \phi$ et $\sigma \models \psi$
  - $\sigma \models \neg\phi$ ssi $\sigma \not\models \phi$
  - $\sigma \models \mathbf{N}\phi$ ssi $c = \sigma(0) \cdot \sigma'$ et $\sigma' \models \phi$
  - $\sigma \models \phi \mathbf{U} \psi$ ssi
    - $\sigma \models \psi$ ou
    - Il existe un entier n tel que $\sigma = \sigma(0) \ldots \sigma(n)\sigma'$ avec $\mathcal{A}, \sigma' \models \psi$ et pour tout $i \in \{0, \ldots, n\}$ on a, $\sigma(i) \ldots \sigma(n)\sigma' \models \phi$

Quelques notations

- $\mathcal{A} = (S, S_0, T)$: Un système de transitions (on oublie les étiquettes des transitions)
- Un ensemble de propriétés atomiques : $Prop = \{P_0, P_1, P_2, \ldots, P_n\}$
- Fonction d'interprétation : $\rho : S \rightarrow 2^{Prop}$
- Soit $c = t_1, t_2 \ldots$, une exécution de $\mathcal{A}$
  - $c \models \phi$ ssi $\rho(t_1)\rho(t_2)\ldots \models \phi$
- $\mathcal{A} \models \phi$ si toute trace de $\mathcal{A}$ à partir de S_0 est un modèle de ϕ

Exemples

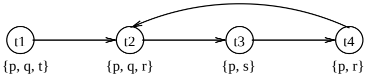
1. 

    - $t_1(t_2t_3t_4)^\omega \models r \rightarrow s$
    - $t_1(t_2t_3t_4)^\omega \not\models \mathbf{N}(p \leftrightarrow s)$
    - $t_1(t_2t_3t_4)^\omega \models \mathbf{NN}\,s$
    - $t_1(t_2t_3t_4)^\omega \models q\,\mathbf{U}\,s$
    - $t_1(t_2t_3t_4)^\omega \models s\,\mathbf{U}\,q$
    - $(t_2t_3t_4)^\omega \models \neg(p\,\mathbf{U}\,t)$

2. L'algorithme de Peterson satisfait la propriété d'exclusion mutuelle ssi

$$\mathcal{P}_{Pet} \models \neg(1\,\mathbf{U}(P_{sc0} \wedge P_{sc1})).$$

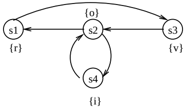
Opérateurs temporaux auxilliaires



- $\Diamond\phi \stackrel{\text{Def}}{=} \mathbf{1U}\phi$ (Inéxorablement ϕ) Ex : $t_1(t_2t_3t_4)^\omega \models \Diamond p$
- $\Box\phi \stackrel{\text{Def}}{=} \neg\Diamond\neg\phi$ (Toujours ϕ) Ex : $t_1(t_2t_3t_4)^\omega \models \Box p$
- Quelques schémas de formules utiles
  1. $c \models \Box\Diamond\phi$ ssi il existe une infinité de suffixes c_2 de c tel que $c_2 \models \phi$.
     Ex : $t_1(t_2t_3t_4)^\omega \models \Box\Diamond(q\,\mathbf{U}s)$
  2. $c \models \Diamond\Box\neg\phi$ ssi il existe un nombre fini de suffixes c' de c tels que $c' \models \phi$ Ex : $t_1(t_2t_3t_4)^\omega \models \Diamond\Box\neg t$

Un feu de circulation intermittent

$\mathcal{A} = (S, T)$: modèle de feu de circulation avec feu orange intermittent :



Propositions atomiques : r (rouge), o (orange), v (vert), i (intermittent).
On a alors les propriétés suivantes :

- $(\mathcal{A}, s_1) \models \mathbf{NN}\, o$
- $(s_1 s_3 s_2)^*(s_4 s_2)^\omega \subseteq \{\sigma \in \mathbb{B}^4 : \sigma \models \neg i \,\mathbf{U}\, i\}$
- $(\mathcal{A}, s_1) \not\models \neg i \,\mathbf{U}\, i$
- $(\mathcal{A}, s_4) \models \neg i \,\mathbf{U}\, i$
- $(\mathcal{A}, s_4) \models i \,\mathbf{U}\, \neg i$
- $(\mathcal{A}, S) \models \Diamond\, o$
- $(s_1 s_3 s_2)^\omega \models \Box \neg i$
- $(\mathcal{A}, S) \models \Box\Diamond\, o$
- $(s_1 s_3 s_2)^*(s_4 s_2)^\omega \subseteq \{\sigma \in \mathbb{B}^4 : \sigma \models \Diamond\Box \neg r\}$

Model-checking, réalisabilité et validité

1. ϕ *réalisable* s'il existe $\sigma \in (\mathbb{B}^n)^\omega$ tel que $\sigma \models \phi$.
   - $\Box(p \Rightarrow \mathbf{N}q)$ est réalisable
2. ϕ est *valide* si pour tout $\sigma \in (\mathbb{B}^n)^\omega$, σ est un modèle de ϕ
   - $(p \wedge \Box(p \Rightarrow \mathbf{N}p)) \Rightarrow \Box p$ est valide
3. Le problème du *model-checking* :
   - $\mathcal{A}$: un système de transitions
   - Montrer que

$$\mathcal{A} \models \phi$$

Problème du modèle checking

$Tr(\mathcal{A})$: Ensemble des traces de $\mathcal{A}$
$[\![\phi]\!]$: Ensemble des modèles de la propriété ϕ

$$\mathcal{A} \models \phi \text{ ssi } Tr(\mathcal{A}) \subseteq [\![\phi]\!]$$

Un canal de communication

Le canal

- Un émetteur S et d'un récepteur R.
- Munis d'un tampon de capacité infinie parfait $S.out$ et $R.in$ resp.
- $S.out$ et $R.in$ sont connectés par un canal unidirectionnel parfait.
- Un message m envoyé par S est d'abord inséré dans $S.out$.
- À la réception par R, m est éliminé de $R.in$.
- On suppose que :
  - les messages sont atomiques et uniquement identifiés
  - l'ensemble M des messages est fini.

Un canal de communication

$$Prop = \{m \in S.out : m \in M\} \cup \{m \in R.in : m \in M\}$$

On formalise les requis suivants en logique temporelle linéaire :

La spécification en LTL

- “Un message ne peut être dans les deux tampons en même temps”.
- “Le canal ne perd pas de messages”.

Un canal de communication

$$Prop = \{m \in S.out : m \in M\} \cup \{m \in R.in : m \in M\}$$

On formalise les requis suivants en logique temporelle linéaire :

La spécification en LTL

- “Un message ne peut être dans les deux tampons en même temps”.

$$\Box\neg(m \in S.out \wedge m \in R.in)$$

- “Le canal ne perd pas de messages”.

Un canal de communication

$$Prop = \{m \in S.out : m \in M\} \cup \{m \in R.in : m \in M\}$$

On formalise les requis suivants en logique temporelle linéaire :

La spécification en LTL

- "Un message ne peut être dans les deux tampons en même temps".

$$\square\neg(m \in S.out \wedge m \in R.in)$$

- "Le canal ne perd pas de messages".

$$\square[m \in S.out \Rightarrow \Diamond(m \in R.in)]$$

Si on assume l'unicité des messages.

Un canal de communication

La spécification en LTL

- “Le canal préserve à la sortie, l’ordre d’entrée des messages”
- “Le canal ne génère pas spontanément de messages”.

Un canal de communication

La spécification en LTL

- "Le canal préserve à la sortie, l'ordre d'entrée des messages"

$$\Box[m \in S.out \wedge \neg m' \in S.out \wedge \Diamond(m' \in S.out)$$
$$\Rightarrow \Diamond(m \in R.in \wedge \neg m' \in R.in \wedge \Diamond(m' \in R.in))]$$

- "Le canal ne génère pas spontanément de messages".

Un canal de communication

La spécification en LTL

- "Le canal préserve à la sortie, l'ordre d'entrée des messages"

$$\square[m \in S.out \wedge \neg m' \in S.out \wedge \Diamond(m' \in S.out)$$
$$\Rightarrow \Diamond(m \in R.in \wedge \neg m' \in R.in \wedge \Diamond(m' \in R.in))]$$

- "Le canal ne génère pas spontanément de messages".

$$\square[(\neg m \in R.in)\mathbf{U}(m \in S.out)]$$

Élection dynamique d'un leader

On fait les suppositions suivantes sur le comportement dynamique des processus :

- Chaque processus a un identificateur unique (un entier)
- ils sont initialement inactifs (ne participent donc pas à l'élection)
- ils peuvent s'activer à tout moment (et participer à l'élection)
- ils ne peuvent rester inactifs indéfiniment
- une fois activés ils ne peuvent se désactiver

Élection dynamique d'un leader

On pose :

$$Prop = \{leader_i : i \leqslant N\} \cup \{active_i : i \leqslant N\} \cup \{i < j : i, j \leqslant N\}$$

On suppose qu'un processus inactif n'est pas un leader. On formalise les requis suivants en logique temporelle linéaire :

Specification en LTL

"Il y a toujours un et un seul leader"

Élection dynamique d'un leader

On pose :

$$Prop = \{leader_i : i \leqslant N\} \cup \{active_i : i \leqslant N\} \cup \{i < j : i, j \leqslant N\}$$

On suppose qu'un processus inactif n'est pas un leader. On formalise les requis suivants en logique temporelle linéaire :

Specification en LTL

"Il y a toujours un et un seul leader"

$$\Box[\exists_i leader_i \wedge (\forall_{j \neq i} \neg leader_j)]$$

qui n'est pas vérifié puisqu'initialement tous les processus sont inactifs. De plus, la commutation d'un leader à un autre peut difficilement être atomique.

Élection dynamique d'un leader

$$Prop = \{leader_i : i \leqslant N\} \cup \{active_i : i \leqslant N\} \cup \{i < j : i, j \leqslant N\}$$

Specification en LTL

"Il y a toujours un et un seul leader"

Une autre tentative :

Élection dynamique d'un leader

$$Prop = \{leader_i : i \leqslant N\} \cup \{active_i : i \leqslant N\} \cup \{i < j : i, j \leqslant N\}$$

Specification en LTL

"Il y a toujours un et un seul leader"

Une autre tentative :

$$\Box\Diamond[\exists_i leader_i \wedge (\forall_{j \neq i} \neg leader_j)]$$

qui permet de n'avoir temporairement aucun leader.
Malheureusement elle permet également plus d'un leader à la fois.

Élection dynamique d'un leader

$$Prop = \{leader_i : i \leqslant N\} \cup \{active_i : i \leqslant N\} \cup \{i < j : i, j \leqslant N\}$$

Specification en LTL

On considère donc les 2 propriétés suivantes :

- "Il y a toujours au plus un leader".
- "Il y aura assez de leaders en temps voulu"
  (pour éviter un protocole d'élection qui n'élit personne)

Élection dynamique d'un leader

$$Prop = \{leader_i : i \leqslant N\} \cup \{active_i : i \leqslant N\} \cup \{i < j : i, j \leqslant N\}$$

Specification en LTL

On considère donc les 2 propriétés suivantes :

- "Il y a toujours au plus un leader".

$$\Box[\exists_i leader_i \rightarrow (\forall_{j \neq i} \neg leader_j)]$$

- "Il y aura assez de leaders en temps voulu"
(pour éviter un protocole d'élection qui n'élit personne)

Élection dynamique d'un leader

$$Prop = \{leader_i : i \leqslant N\} \cup \{active_i : i \leqslant N\} \cup \{i < j : i, j \leqslant N\}$$

Specification en LTL

On considère donc les 2 propriétés suivantes :

- "Il y a toujours au plus un leader".

$$\Box[\exists_i leader_i \rightarrow (\forall_{j \neq i} \neg leader_j)]$$

- "Il y aura assez de leaders en temps voulu" (pour éviter un protocole d'élection qui n'élit personne)

$$\Box\Diamond[\exists_i leader_i]$$

Élection dynamique d'un leader

$$Prop = \{leader_i : i \leqslant N\} \cup \{active_i : i \leqslant N\} \cup \{i < j : i, j \leqslant N\}$$

Specification en LTL

- "En présence d'un processus actif de rang supérieur le leader finira par se démettre"
- "Tout nouveau leader sera plus qualifié que le précédent"

Élection dynamique d'un leader

$$Prop = \{leader_i : i \leqslant N\} \cup \{active_i : i \leqslant N\} \cup \{i < j : i, j \leqslant N\}$$

Specification en LTL

- "En présence d'un processus actif de rang supérieur le leader finira par se démettre"

$$\square[\forall_{i,j}((leader_i \wedge (i < j) \wedge \neg leader_j \wedge active_j) \Rightarrow \lozenge \neg leader_i)]$$

- "Tout nouveau leader sera plus qualifié que le précédent"

Élection dynamique d'un leader

$$Prop = \{leader_i : i \leqslant N\} \cup \{active_i : i \leqslant N\} \cup \{i < j : i,j \leqslant N\}$$

Specification en LTL

- "En présence d'un processus actif de rang supérieur le leader finira par se démettre"

$$\Box[\forall_{i,j}((leader_i \wedge (i < j) \wedge \neg leader_j \wedge active_j) \Rightarrow \Diamond\neg leader_i)]$$

- "Tout nouveau leader sera plus qualifié que le précédent"

$$\Box[\forall_{i,j}(leader_i \wedge \neg \mathbf{N} leader_i \wedge \mathbf{N}\Diamond leader_j) \Rightarrow (i < j)]$$

Plan de la partie I

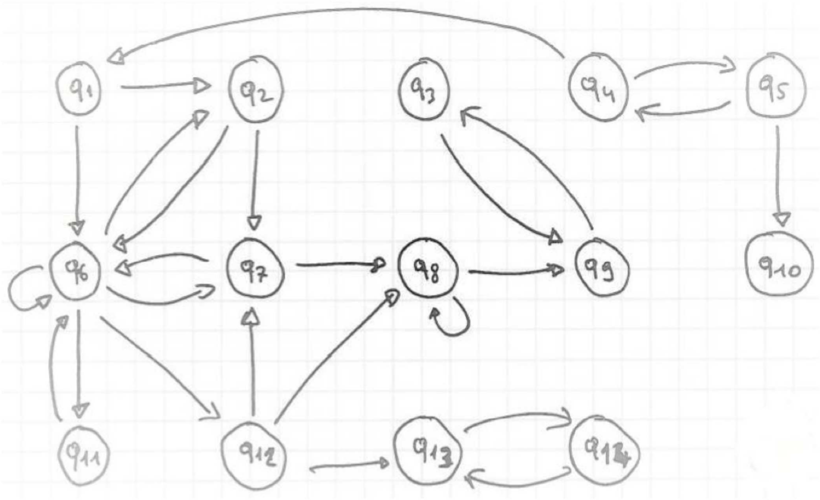
Composantes fortement connexes d'un graphe orienté

$G = (S, A)$, un graphe orienté

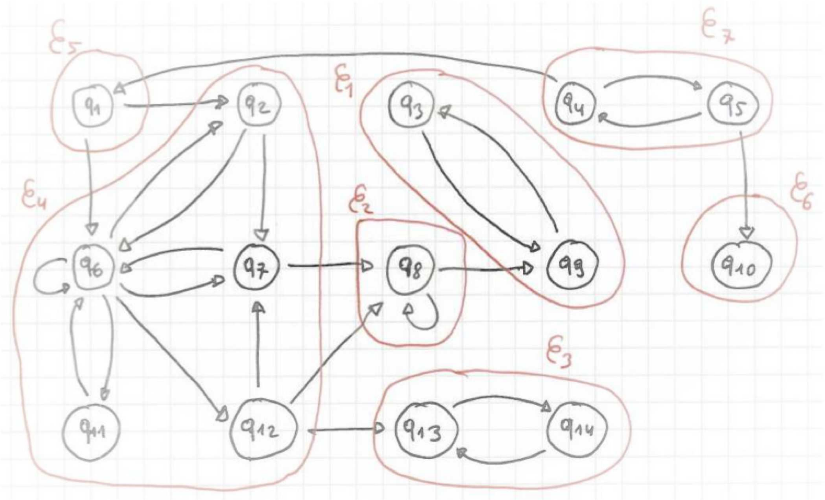
Définition

Une composante fortement connexe (CFC) $\mathcal{C}$ de G est un sous-ensemble maximal de sommets de G tel que si $u, v \in \mathcal{C}$ alors $u \rightarrow^* v$ et $v \rightarrow^* u$

Exemple : le graphe G



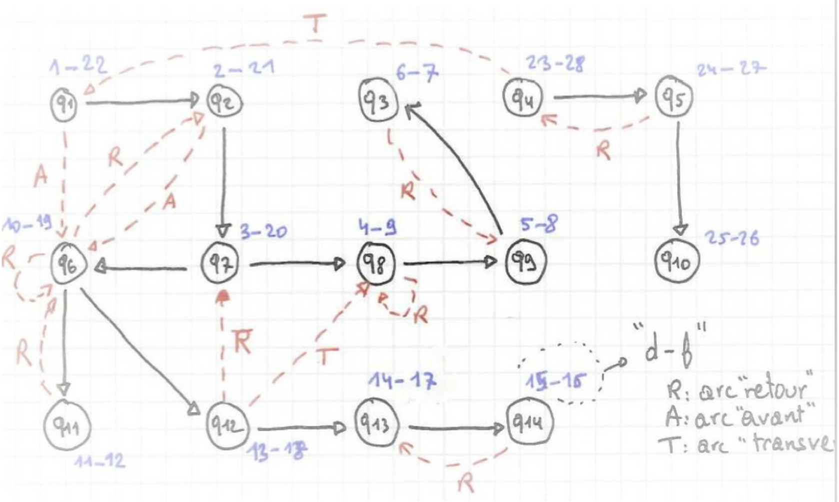
Exemple : les CFC de G



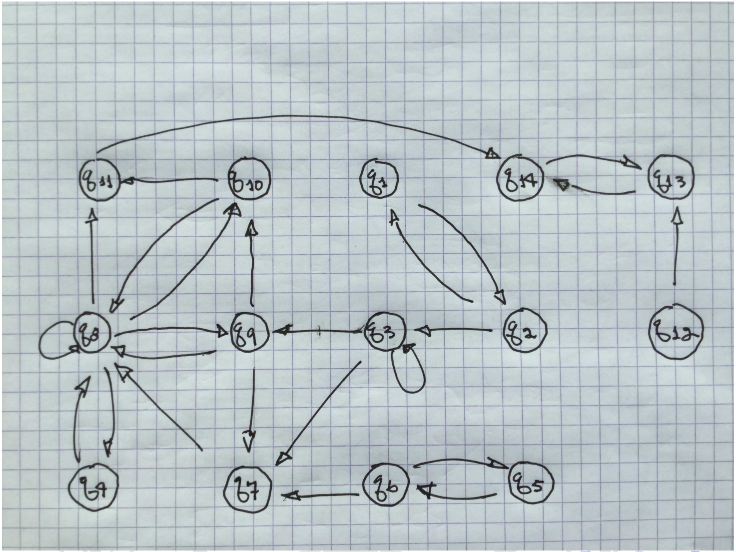
Calcul des CFC : Algorithme de Kosaraju (1978)

1. Faire un parcours en profondeur de G et numéroter les sommets dans l'ordre de complétion des appels récursifs
2. Construire un nouveau graphe G_r obtenu de G en inversant la direction des arcs de G
3. Fair un parcours en profondeur de G_r en appelant les somments dans l'ordre décroissant de la numérotation calculée à l'étape 1
4. Les CFC de G (et aussi de G_r) sont les arbres ainsi obtenus

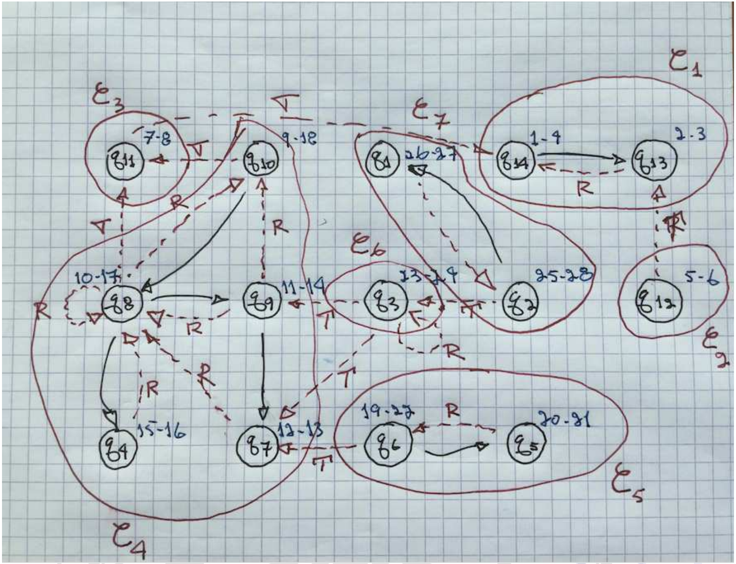
Exemple : 1. parcours en profondeur de G



Exemple : 2. Construction de G_r



Exemple : 3. parcours en profondeur de G_r

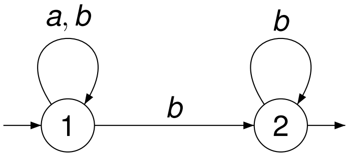


Automates sur des mot infinis

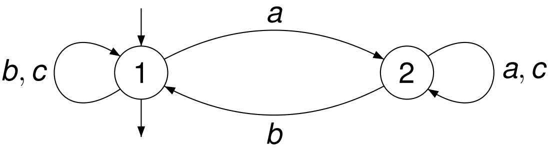
- Automate de Büchi $\mathcal{B} = (S, T, S_0, F, \Sigma)$ où
  - S est un ensemble d'états
  - $S_0 \subseteq S$ est l'ensemble des états initiaux
  - $T \subseteq S \times \Sigma \times S$ est l'ensemble des transitions
  - $F \subseteq S$, un ensemble d'états finaux
  - Σ est un alphabet
- Critère d'acceptation de Büchi : Un ω-mot sur Σ est reconnaissable par $\mathcal{B}$ si la chaîne des états visités par $\mathcal{B}$ en le lisant de gauche à droite depuis s_0, passe par F infiniment souvent
- $L(\mathcal{B})$ l'ensemble des ω-mots sur Σ reconnaissables par $\mathcal{B}$.

Automates de Büchi : Exemples

$L(\mathcal{B}) =$

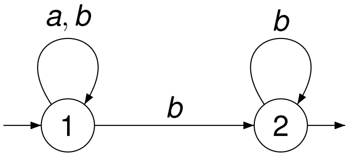


$L(\mathcal{B}) =$



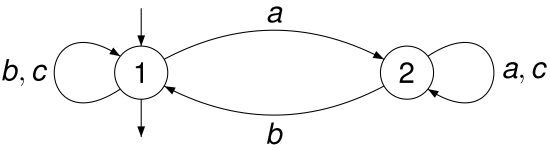
Automates de Büchi : Exemples

$L(\mathcal{B}) = (a + b)^* b^\omega$



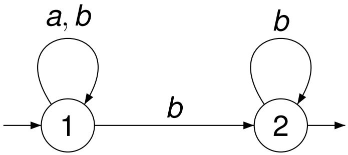
$\alpha \in L(\mathcal{B})$ ss'il n'y a qu'un nombre fini de a

$L(\mathcal{B}) =$



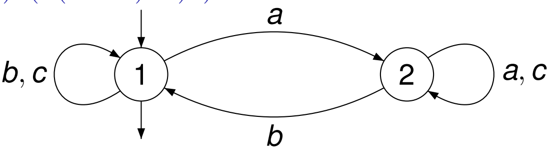
Automates de Büchi : Exemples

$L(\mathcal{B}) = (a+b)^* b^\omega$



$\alpha \in L(\mathcal{B})$ ss'il n'y a qu'un nombre fini de a

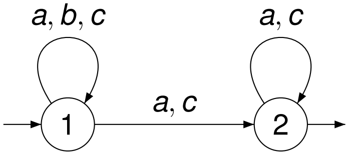
$L(\mathcal{B}) = ((b+c)^*(a(a+c)^*b)^*)^\omega$



$\alpha \in L(\mathcal{B})$ ssi en tout temps si a alors plus tard b

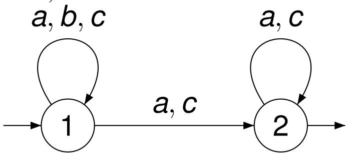
Automates de Büchi : Exemples

$L(\mathcal{B}) =$



Automates de Büchi : Exemples

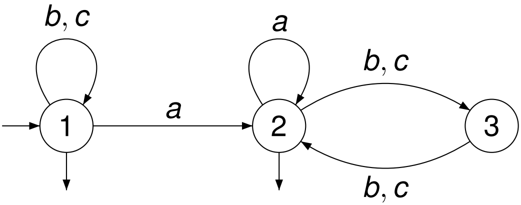
$L(\mathcal{B}) = (a + b + c)^*(a + c)^\omega$



$\alpha \in L(\mathcal{B})$ ss'il n'y a qu'un nombre fini de b

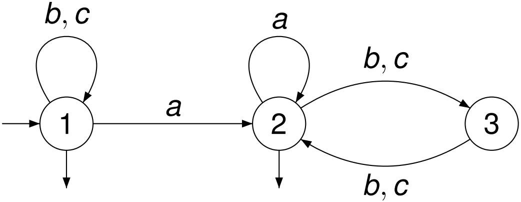
Automates de Büchi : Exemples

$L(\mathcal{B})$ =



Automates de Büchi : Exemples

$L(\mathcal{B}) = (b+c)^\omega +$
$(b+c)^* a(a^*((b+c)^2)^*)^\omega$



$\alpha \in L(\mathcal{B})$ ssi entre 2 suites de *a* il y a toujours un nombre pair de lettres

Automates sur des mot infinis

Théorème (Décidabilité)

Pour un automate de Büchi $\mathcal{B}$, on peut décider si $L(\mathcal{B})$ est vide.

- Preuve :

Propriétés de fermeture

Les langages ω-réguliers sont fermés pour l'intersection, l'union et le complément :

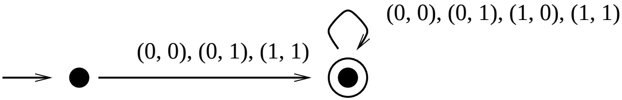
- Si U et V sont ω-réguliers alors $U \cap V$, $U \cup V$, $\Sigma^\omega \backslash U$ sont ω-réguliers

Remarque

Les automates de Büchi déterministes sont strictement moins expressifs que les automates de Büchi non-déterministe (et donc ne reconnaissent pas tous les langages ω-réguliers).

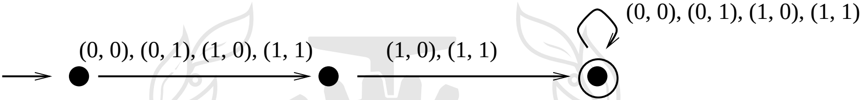
Générateur des modèles d'une formule de LTL (1)

- Exemple 1 : L'automate de Büchi



reconnait précisément les modèles de la formule $p_1 \rightarrow p_2$

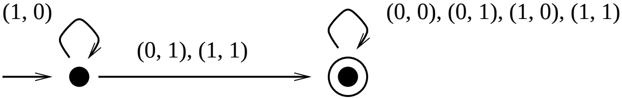
- Exemple 2 : L'automate de Büchi



reconnait précisément les modèles de la formule $\mathbf{N}\, p_1$

Générateur des modèles d'une formule de LTL (2)

- Exemple 3 : L'automate de Büchi

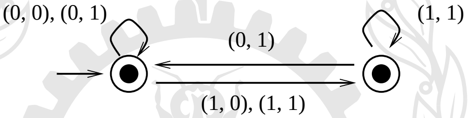


reconnait précisément les modèles de la formule $p_1 \mathbf{U} p_2$

- Exemple 4 : les modèles de la formule

$$\Box(p_1 \rightarrow \mathbf{N} p_2)$$

sont générés par l'automate de Büchi



Le problème du model-checking (rappel)

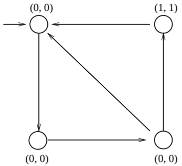
Problème du modèle checking

- $\mathcal{S}$: Système de transitions (avec n propriétés atomiques)
- $\mathit{Traces}(\mathcal{S})$: Ensemble des traces de $\mathcal{S}$
- ϕ : Propriété temporelle linéaire
- $[\![\phi]\!](\subseteq (\mathbb{B}^n)^\omega)$: Ensemble des modèles ϕ

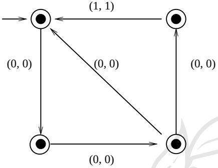
$$
\begin{aligned}
\mathcal{S} \text{ satisfait } \phi(\mathcal{S} \models \phi) \quad & \leftrightarrow \quad \mathit{Traces}(\mathcal{S}) \subseteq [\![\phi]\!] \\
& \leftrightarrow \quad \mathit{Traces}(\mathcal{S}) \cap \overline{[\![\phi]\!]} = \emptyset \\
& \leftrightarrow \quad \mathit{Traces}(\mathcal{S}) \cap [\![\neg\phi]\!] = \emptyset
\end{aligned}
$$

Algorithme de model-checking de la LTL (1)

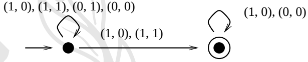
Vérifier : $\phi \equiv \Box(p_1 \rightarrow \mathbf{N}\Diamond p_2)$ sur



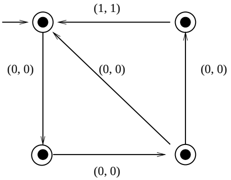
1. Transformer $\mathcal{S}$ en automates de Büchi qui accepte $Traces(\mathcal{S})$:

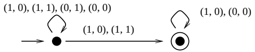


2. Calculer $\mathcal{B}_{\neg\phi}$. $(\neg\phi \equiv \Diamond(p_1 \wedge \mathbf{N}\Box(\neg p_2)))$

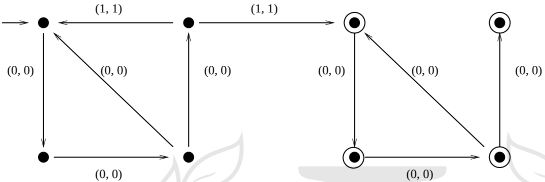


Algorithme de model-checking de la LTL (2)





3. Calculer le produit synchronisé (qui reconnait l'intersection)



4. Déterminer si l'intersection est vide (Calcul des composantes fortement connexes)
5. OUI : ok ; NON : Retourner $x \cdot y^\omega$ qui viole ϕ. Ici :

$$(0,0)(0,0)(0,0)(1,1)((0,0)(0,0)(0,0))^\omega$$

La spécification de l'élection d'un leader dans Spin

- `[] p`
- **Déclaration** : `#define p      (nbre_leaders <= 1)`
- `nbre_leaders` est une variable auxilliaire globale

```
byte nbre_leaders = 0
proctype process (chan in, out; byte ident)
       byte d, e, f;
        printf("MSC: %d\n", ident);
activ:
        d = ident;
        do :: true -> out!d;
                      in?e;
                      if :: (e == d) ->
                             printf("MSC: %d is LEADER\n", d);
                             nbre_leaders = nbre_leaders + 1;
                             goto stop /* d est le leader */
                         :: else -> skip

......... comme avant .....................
```

Calcul du générateur des modèles de $\neg(\Box p)$ par SPIN

```
        /*
         * Formula As Typed: []  p
         * The Never Claim Below Corresponds
         * To The Negated Formula !([]  p)
         * (formalizing violations of the original)
         */

never {     /* !([]  p) */
T0_init:
        if
        :: (! ((p))) -> goto accept_all
        :: (1) -> goto T0_init
        fi;
accept_all:
        skip
}
```

Architecture du model-checker de SPIN