Université de Sherbrooke
Faculté des sciences
Département d'informatique
professeur Marc Frappier

Brève introduction à CSP

1 Rappel des définitions

1. Σ dénote les actions (évènements) observables du système.
2. Σ^* dénote toutes les traces (*ie*, séquences) générées à partir de Σ (*ie*, la fermeture de Kleene).
3. ϵ dénote une trace vide, aussi appelée $[\,]$ (séquence vide) dans le langage B, et *mot vide* dans la théorie des langages.
4. τ dénote une action interne, c'est-à-dire un évènement non observable par l'environnement; notons que $\tau \neq \epsilon$, car τ peut être un élément d'une trace, alors que ϵ est une trace vide, donc elle ne contient aucun élément.
5. *STOP* dénote un processus qui ne peut rien faire.
6. *SKIP* dénote un processus qui termine normalement, et qui permet de déclencher le processus suivant dans une composition séquentielle.
7. Ω dénote le processus résultant de l'exécution de *SKIP*.
8. ✓ dénote la terminaison avec succès d'un processus. Produit par l'exécution du processus *SKIP* et de $\Omega \,[\![\, a \,]\!]\, \Omega$.
9. $A^{\checkmark} = A \cup \{\checkmark\}$.
10. $A^{\tau} = A \cup \{\tau\}$.
11. $\mathcal{BE}$ dénote l'ensemble des expressions de processus que l'on peut définir.
12. $B \overset{\mu}{\longmapsto} B'$ dénote que l'expression de processus B peut exécuter μ et se transformer en B'.
13. $B \overset{\mu}{\not\longmapsto}$ dénote que B ne peut exécuter μ.
14. $CHAOS(A) = (\sqcap\, x : A \;@\; x \rightarrow CHAOS(A)) \sqcap STOP$
15. $[a, b]$ dénote une séqence formée des éléments a et b
16. $s \frown t$ dénote la concaténation des séquences s et t
17. $\mu \rightarrow s = [\mu] \frown s$
18. $s \leftarrow \mu = s \frown [\mu]$

2 Syntaxe de CSP

2.1 Opérateurs de base

Expression de processus	Description	ASCII CSPM
$STOP$	Ne peut rien faire	`STOP`
$SKIP$	Termine normalement	`SKIP`
$CHAOS(a)$	Processus le plus nondéterministe sur les actions de l'ensemble a	`CHAOS(a)`
$c \rightarrow p$	Préfixe : exécute action c et se transforme ensuite en p	`c -> p`
$c?x : a \rightarrow p$	Préfixe, mais avec action c avec paramètre d'entrée x	`c?x:a -> p`
$c!v \rightarrow p$	Préfixe, mais avec action c avec paramètre de sortie x	`c!v -> p`
$p \,⨟\, q$	Composition séquentielle : exécute p et ensuite q si p termine normalement (*ie*, avec SKIP)	`p ; q`
$p \setminus a$	Exécute p en masquant les actions de l'ensemble a, qui sont observées comme des τ	`p \ a`
$p \,\Box\, q$	Choix externe	`p [] q`
$p \sqcap q$	Choix interne	`p |~| q`
$b \,\&\, q$	Garde : si la condition b est satisfaite, exécute q, et la garde b disparait ensuite	`b & q`
if b then p else q	Abréviation pour $(b \,\&\, p) \,\Box\, (\neg b \,\&\, q)$	`if b then p else q`
$p \,[\![\, a \,]\!]\, q$	Exécution en parallèle de p et q avec synchronisation sur les actions de a (et entrelacement sur les autres actions)	`p [| a |] q`
$p \,\|\!\|\!\|\, q$	Exécution en entrelacement de p et q Abréviation pour $[\![\{\}]\!]$	`p ||| q`

Note: $P \underset{X}{\|} Q$ du livre de Roscoe est noté ici $P \,[\![\, X \,]\!]\, Q$ comme dans la version ASCII de CSP.

2.2 Opérateurs quantifiés

Soit $n_s = card(s)$ et $n_a = card(a)$.

Expression de processus	Description	ASCII CSPM
$⨟\, x : s \bullet p$	Exécution de n_s instances de $p[x := v]$, dans l'ordre donné par la séquence s, *ie*, une instance pour chaque valeur $s(i)$, avec $i \in 1..n_s$, soit $p[x := s(1) ⨟ \ldots ⨟ p[x := s(n_s)]$.	`; x : a @ p`
$\Box\, x : a \bullet p$	Choix externe entre n_a instances de $p[x := v]$, *ie*, une instance pour chaque valeur $v \in a$, soit $p[x := v_1) \Box \ldots \Box p[x := v_{n_a}]$.	`[] x : a @ p`
$\sqcap\, x : a \bullet p$	Choix interne entre n_a instances de $p[x := v]$, *ie*, une instance pour chaque valeur $v \in a$, soit $p[x := v_1] \sqcap \ldots \sqcap p[x := v_{n_a}]$.	`|~| x : a @ p`
$\vert\vert\vert\, x : a \bullet p$	Entrelacement de n_a instances de $p[x := v]$, *ie*, une instance pour chaque valeur $v \in a$, soit $p[x := v_1] \vert\vert\vert \ldots \vert\vert\vert p[x := v_{n_a}]$.	`||| x:a @ p`
$[\![a']\!]\, x : a \bullet p$	Synchronisation sur a' de n_a instances de $p[x := v]$, *ie*, une instance pour chaque valeur $v \in a$, soit $p[x := v_1] [\![a']\!] \ldots [\![a']\!] p[x := v_{n_a}]$.	`||| x:a @ p`

3 Sémantique opérationnelle de CSP

3.1 Règles d'inférence

$$\frac{}{a \rightarrow P \xmapsto{a} P}\ \rightarrow \qquad \frac{}{SKIP \xmapsto{\checkmark} \Omega}\ SKIP$$

$$\frac{}{P \sqcap Q \xmapsto{\tau} P}\ \sqcap_1 \qquad \frac{}{P \sqcap Q \xmapsto{\tau} Q}\ \sqcap_2$$

$$\frac{P \xmapsto{\tau} P'}{P \,\Box\, Q \xmapsto{\tau} P' \,\Box\, Q}\ \Box_1 \qquad \frac{Q \xmapsto{\tau} Q'}{P \,\Box\, Q \xmapsto{\tau} P \,\Box\, Q'}\ \Box_2$$

$$\frac{P \xmapsto{\sigma} P' \quad \sigma \neq \tau}{P \,\Box\, Q \xmapsto{\sigma} P'}\ \Box_3 \qquad \frac{Q \xmapsto{\sigma} Q' \quad \sigma \neq \tau}{P \,\Box\, Q \xmapsto{\sigma} Q'}\ \Box_4$$

$$\frac{P \xmapsto{\sigma} P' \quad \sigma \neq \checkmark}{P \fatsemi Q \xmapsto{\sigma} P' \fatsemi Q}\ \fatsemi_1 \qquad \frac{P \xmapsto{\checkmark} P'}{P \fatsemi Q \xmapsto{\tau} Q}\ \fatsemi_2$$

$$\frac{P \xmapsto{\sigma} P' \quad \sigma \notin B \cup \{\checkmark\}}{P \setminus B \xmapsto{\sigma} P' \setminus B}\ \setminus_1 \qquad \frac{P \xmapsto{\checkmark} P'}{P \setminus B \xmapsto{\checkmark} \Omega}\ \setminus_2 \qquad \frac{P \xmapsto{\sigma} P' \quad \sigma \in B}{P \setminus B \xmapsto{\tau} P' \setminus B}\ \setminus_3$$

$$\frac{P \xmapsto{\sigma} P' \quad \sigma \notin X \cup \{\checkmark\}}{P \llbracket X \rrbracket Q \xmapsto{\sigma} P' \llbracket X \rrbracket Q}\ \llbracket X \rrbracket_1 \qquad \frac{Q \xmapsto{\sigma} Q' \quad \sigma \notin X \cup \{\checkmark\}}{P \llbracket X \rrbracket Q \xmapsto{\sigma} P \llbracket X \rrbracket Q'}\ \llbracket X \rrbracket_2$$

$$\frac{P \xmapsto{\sigma} P' \quad Q \xmapsto{\sigma} Q' \quad \sigma \in X}{P \llbracket X \rrbracket Q \xmapsto{\sigma} P' \llbracket X \rrbracket Q'}\ \llbracket X \rrbracket_3$$

$$\frac{P \xmapsto{\checkmark} P'}{P \llbracket X \rrbracket Q \xmapsto{\tau} \Omega \llbracket X \rrbracket Q}\ \llbracket X \rrbracket_4 \qquad \frac{Q \xmapsto{\checkmark} Q'}{P \llbracket X \rrbracket Q \xmapsto{\tau} P \llbracket X \rrbracket \Omega}\ \llbracket X \rrbracket_5$$

$$\frac{}{\Omega \llbracket X \rrbracket \Omega \xmapsto{\checkmark} \Omega}\ \|_6$$

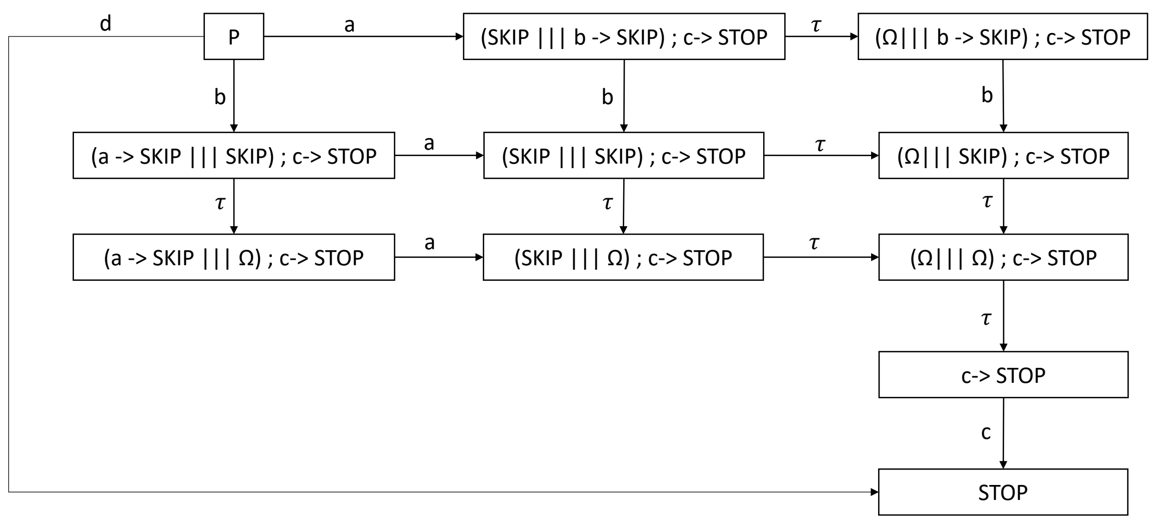
$$\frac{C \qquad P \xmapsto{\sigma} P'}{C \;\&\; P \xmapsto{\sigma} P'}\ \&_1$$

3.2 Exemples de preuves

Soit

```
P = ((a -> SKIP ||| b -> SKIP) ; c-> STOP ) [] d -> STOP
```

Voici le graphe de transition de `P`.



Prouvons la transition $\texttt{P} \overset{a}{\longmapsto} \texttt{P'}$, où

```
P' = ((SKIP ||| b -> SKIP) ; c-> STOP )
```

Voici l'arbre de preuve de cette transition.

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{\texttt{a -> SKIP} \overset{a}{\longmapsto} \texttt{SKIP}}\;\rightarrow \qquad a \notin \{\} \cup \{\checkmark\}}{\texttt{(a -> SKIP ||| b -> SKIP)} \overset{a}{\longmapsto} \texttt{(SKIP ||| b -> SKIP)}}\;[\![X]\!]_1 \qquad a \neq \checkmark}{\texttt{(a -> SKIP ||| b -> SKIP) ; c-> STOP} \overset{a}{\longmapsto} \texttt{(SKIP ||| b -> SKIP) ; c-> STOP}}\;\mathring{9}_1}{\texttt{P} \overset{a}{\longmapsto} \texttt{P'}}\;\square_3
$$

4 Bisimulation

Definition 1 La fermeture transitive et reflexive de la relation de transition $\longmapsto$, notée $\underset{*}{\longmapsto}$, est un sous-ensemble de l'espace $\mathcal{BE} \times (\Sigma^{\tau\checkmark})^* \times \mathcal{BE}$ et elle est définie comme suit. Soient $B, B' \in \mathcal{BE}$ des expressions de processus, $s \in (\Sigma^{\tau\checkmark})^*$ une séquence et $\mu \in \Sigma^{\tau\checkmark}$ une action.

1. $B \overset{\epsilon}{\underset{*}{\longmapsto}} B$

2. $B \overset{\mu \rightarrow s}{\underset{*}{\longmapsto}} B' \quad \Leftrightarrow \quad \exists\, B'' : B \overset{\mu}{\longmapsto} B'' \;\wedge\; B'' \overset{s}{\underset{*}{\longmapsto}} B'$

□

Definition 2 La restriction d'une séquence s aux éléments d'un ensemble A est notée $s\lceil A$ et définie comme suit.

$$
\begin{array}{l}
\langle\rangle\lceil A = \langle\rangle \\
u \in A \;\Rightarrow\; (u{\rightarrow}s)\lceil A = u{\rightarrow}(s\lceil A) \\
u \notin A \;\Rightarrow\; (u{\rightarrow}s)\lceil A = s\lceil A
\end{array}
$$

□

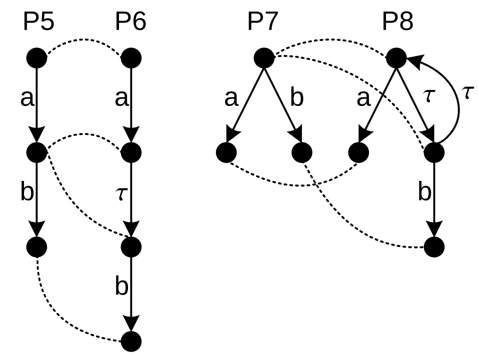


Figure 1: Exemples de bisimulation

Definition 3 La fermeture transitive et reflexive de la relation de transition $\longmapsto$ *restreinte* aux actions observables, notée $\underset{*}{\Longmapsto}$, est un sous-ensemble de l'espace $\mathcal{BE} \times (\Sigma^{\checkmark})^* \times \mathcal{BE}$ et elle est définie comme suit. Soient $B, B' \in \mathcal{BE}$ des expressions de processus et $s \in (\Sigma^{\checkmark})^*$.

$$B \overset{s}{\underset{*}{\Longmapsto}} B' \;\Leftrightarrow\; \exists\, s' \cdot s' \in (\Sigma^{\tau\checkmark})^* \wedge s = s' \lceil \Sigma^{\checkmark} \wedge B \overset{s'}{\underset{*}{\longmapsto}} B'$$

$\Box$

Definition 4 Une relation R sur $\mathcal{BE}$ est une *bisimulation faible* ssi elle satisfait la condition suivante.

$$\begin{array}{l}
\forall\, B_1, B_2 \in \mathcal{BE}, s \in (\Sigma^{\checkmark})^* : \\
\quad (B_1, B_2) \in R \Rightarrow \\
\qquad (\forall\, B_1' : B_1 \overset{s}{\underset{*}{\Longmapsto}} B_1' \;\;\Rightarrow\;\; \exists\, B_2' : B_2 \overset{s}{\underset{*}{\Longmapsto}} B_2' \;\wedge\; (B_1', B_2') \in R) \\
\quad\; \wedge \\
\qquad (\forall\, B_2' : B_2 \overset{s}{\underset{*}{\Longmapsto}} B_2' \;\;\Rightarrow\;\; \exists\, B_1' : B_1 \overset{s}{\underset{*}{\Longmapsto}} B_1' \;\wedge\; (B_1', B_2') \in R)
\end{array}$$

$\Box$

Definition 5 Deux expressions de processus B_1, B_2 sont dites *équivalentes par observation*, notée $B_1 \approx B_2$, ssi il existe une bisimulation faible R telle que $(B_1, B_2) \in R$. $\Box$

La figure 1 illustre la notion de bisimulation faible. Les expressions P_5 et P_6 sont reliées par une bisimulation. Les paires de cette bisimulation sont représentées par les noeuds reliés par une ligne pointillée. On peut donc conclure que P_5 est équivalent par observation à P_6. De la même manière, P_7 et P_8 sont aussi reliés par une bisimulation, et sont donc équivalent par observation. Toutefois, on note que P_8 peut diverger en exécutant une boucle infinie sur l'action interne τ. La notion d'équivalence par observation ne tient pas compte de ce phénomène.

Voici les lois de congruence pour l'équivalence par observation. Soient $B, C, D \in \mathcal{BE}$ des expressions de processus et $\mu^+ \in \Sigma^{\tau\checkmark}$ une action.

$$\begin{aligned}
B \Box C &\approx_c C \Box B && (1)\\
B \Box (C \Box D) &\approx_c (B \Box C) \Box D && (2)\\
B \Box B &\approx_c B && (3)\\
B \Box \mathit{STOP} &\approx_c B && (4)\\
\mu^+ \to \mathit{SKIP};\ B &\approx_c \mu^+ \to B && (5)\\
B \Box (\mathit{SKIP};\ B) &\approx_c \mathit{SKIP};\ B && (6)\\
\mu^+ \to (B \Box \mathit{SKIP};\ C) \;\Box\; \mu^+ \to C &\approx_c \mu^+ \to (B \Box \mathit{SKIP};\ C) && (7)
\end{aligned}$$

5 Raffinement

Definition 6 Un échec stable (*stable failure*) d'un processus P est une paire (s, X) telle que

1. $\exists\, P' \cdot P \overset{s}{\underset{*}{\Longmapsto}} P' \wedge P' \not\overset{\tau}{\longmapsto} \wedge P' \not\overset{\checkmark}{\longmapsto}$ (*ie*, P' est stable)
2. $\forall\, a \in X \cdot P' \not\overset{a}{\underset{*}{\longmapsto}}$ (*ie*, P' refuse chaque élément de X).

On note *failures*$(P) = \{(s, X) \mid (s, X)$ est un échec stable de $P\}$.
On note *traces*$(P) = \{s \mid s \in \Sigma^{\checkmark} \wedge \exists\, P' \cdot P \overset{s}{\underset{*}{\Longmapsto}} P'\}$.
□

Definition 7 Raffinement: on dit que P est raffiné par traces par P', noté $P \sqsubseteq_T P'$, et défini comme suit:

$$P \sqsubseteq_T P' \quad\Leftrightarrow\quad traces(P) \supseteq traces(P') \ .$$

On dit que P est raffiné par échecs stables par P', noté $P \sqsubseteq_F P'$, et défini comme suit:

$$P \sqsubseteq_F P' \quad\Leftrightarrow\quad failures(P) \supseteq failures(P') \ .$$

On note

$$P_1 =_T P_2 \quad\Leftrightarrow\quad P_1 \sqsubseteq_T P_2 \wedge P_2 \sqsubseteq_T P_1$$

ainsi que

$$P_1 =_F P_2 \quad\Leftrightarrow\quad P_1 \sqsubseteq_F P_2 \wedge P_2 \sqsubseteq_F P_1$$

La figure 2 illustre le raffinement par trace et par échec stable. On note que P_1, P_2, P_3 et P_4 sont équivalents par trace (*ie*, ils ont les mêmes traces). Le processus P_0 raffine par trace chacun de ces processus.

$$P_1 =_T P_2 =_T P_3 =_T P_4 \sqsubseteq_T P_0$$

Quant au raffinement par échec stable, P_2, P_3 et P_4 sont équivalents, et ils sont raffinés par P_0 et P_1. Notons que $P_0 \not\sqsubseteq_F P_1$ et $P_1 \not\sqsubseteq_F P_0$, car leurs échecs stables sont différents.

$$P_2 =_F P_3 =_F P_4 \sqsubseteq_F P_0, P_1$$

On a *failures*(P_2) = *failures*(P_3) = *failures*(P_4) =

$$\Big\{(\langle\rangle, \Sigma - \{a\}), ([a], \Sigma - \{b\}), ([a], \Sigma - \{c\}), ([a, b], \Sigma), ([a, c], \Sigma)\Big\}$$

et *failures*(P_0) =

$$\Big\{(\langle\rangle, \Sigma - \{a\}), ([a], \Sigma - \{b\}), ([a, b], \Sigma)\Big\}$$

et *failures*(P_1) =

$$\Big\{(\langle\rangle, \Sigma - \{a\}), ([a], \Sigma - \{b, c\}), ([a, b], \Sigma), ([a, c], \Sigma)\Big\}$$

Notons que par convention et souci de concision, on ne mentionne pas dans l'ensemble un échec stable (t, S) si (t, S') et $S \subseteq S'$, par soucis de concision. On a donc

$$failures(P_2) = failures(P_3) = failures(P_4) \supseteq failures(P_0), failures(P_1)$$

On voit que P_0 raffine P_2, P_3 et P_4 en réduisant le nombre de traces, tout en ayant les mêmes refus pour chaque trace. On voit que P_1 raffine P_2, P_3 et P_4 en ayant les mêmes traces, mais en réduisant le nombre de refus, car $\Sigma - \{b, c\} \subseteq \Sigma - \{b\}, \Sigma - \{c\}$; P_1 est donc plus déterministe que P_2, P_3 et P_4.

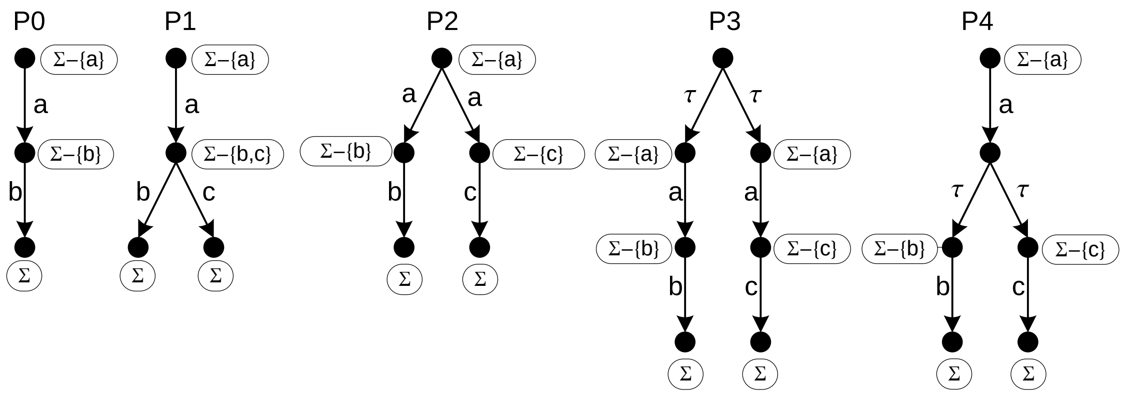


Figure 2: Processus avec identifications des échecs

6 Vérification de propriétés avec FDR2

L'outil FDR2 peut vérifier des assertions de raffinement par traces et par échecs stables. Pour vérifier une propriété de sûreté pour un processus P en représentant la propriété par un autre processus, disons *Prop*, et en montrant que $Prop \sqsubseteq_T P$. Le processus *Prop* représente les traces acceptables selon la propriété et on vérifie ainsi que P n'a que des traces acceptables selon cette propriété. Si la propriété ne porte que sur des événements de A, on vérifie alors

$$Prop \sqsubseteq_T P \setminus (\Sigma - A) \ .$$

Pour vérifier une propriété de vivacité, on utilise le raffinement par échec stable. Par exemple, supposons que l'on veuille prouver la propriété suivante.

> "Le système peut faire un b immédiatement après un a."

On peut spécifier cette propriété en CSP comme suit:

```
Spec = a -> (b -> STOP [] CANREFUSE({a,c,d}))

CANREFUSE(S) = (|~|x:S @ x -> STOP ) |~| STOP
```

L'arbre de transition du processus *Spec* est illustré dans la figure 3. Les refus sont indiqués dans un ovale pour chaque état stable (*ie*, état sans transition sortante sur τ) Pour la trace $[a]$, on a les échecs stables suivants:

1. $([a], \Sigma - \{b\})$
2. $([a], \Sigma - \{b, a\})$
3. $([a], \Sigma - \{b, c\})$
4. $([a], \Sigma - \{b, d\})$

Donc, un raffinement de ce processus doit accepter b immédiatement après un a, puisque chaque ensemble de refus après $[a]$ ne comprend pas b (*ie*, on ne refuse jamais b). Il est possible de refuser a, c ou d (d'où l'intérêt d'utiliser `CANREFUSE({a,c,d})` dans *Spec*). Notons aussi qu'un raffinement de *Spec*:

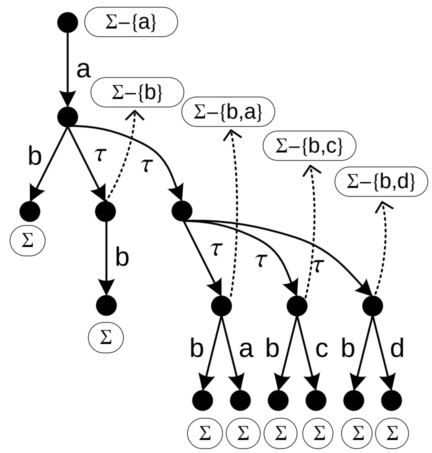


Figure 3: Propriété "a immédiatement suivi par b"
.

- doit accepter a dès l'état initial, car on a le refus $(\langle\rangle, \Sigma - \{a\})$
- peut accepter les traces $[a, a]$, $[a, b]$, $[a, c]$, $[a, d]$, mais il n'est pas obligé; il peut accepter seulement $[a, b]$ s'il le veut.

Le processus *Spec* est raffiné par échec stable par `Main2` et `Main3`, mais pas par `Main1`.

```
Main1 = a -> STOP
Main2 = a -> (b -> STOP [] c -> STOP)
Main3 = a -> b -> STOP
assert Spec [F= Main1 -- FAUX : Main1 ne peut accepter b
assert Spec [F= Main2 -- VRAI : Main2 peut accepter b et autre chose
assert Spec [F= Main3 -- VRAI : Main3 peut accepter b
```

Considérons maintenant la propriété suivante:

> "Le système peut faire un b après un a."

Notons que cette propriété n'impose pas que b surviennent immédiatement après a. Considérons le système suivant

```
Main4 = ((a -> P ; b -> STOP) ||| Q)
P = c -> P [] SKIP
Q = d -> d -> Q [] STOP
```

On peut spécifier cette propriété en CSP comme suit:

```
Spec2 = a -> b -> STOP
```

en vérifiant l'assertion suivante

```
assert Spec2  ||| CHAOS({d})  [F= Main4 \ {c}
```

On masque les événements survenant entre a et b, c'est-à-dire c, et on fait un entrelacement avec `CHAOS({d})` pour faire abstraction de `Q`. On note que la spécification de la propriété en CSP doit prendre en compte la structure du système à vérifier; ce n'est pas le cas lorsqu'on utilise une logique temporelle comme LTL ou CTL pour spécifier les propriétés, ou bien des prédicats sur les traces comme en Alloy.

Le patron à utiliser pour écrire un processus représentant une propriété découle de la *monotonicité* des opérateurs de CSP par rapport à $\sqsubseteq_T$ et $\sqsubseteq_F$. On peut l'exprimer comme suit: soit $\Xi(X)$ une expression de processus CSP comprenant la variable X qui dénote une expression de processus. L'expression $\Xi(P_1)$ dénote le remplacement de X par P_1 dans Ξ.

$$P_1 \sqsubseteq_T P_2 \;\Rightarrow\; \Xi(P_1) \sqsubseteq_T \Xi(P_2)$$

$$P_1 \sqsubseteq_F P_2 \;\Rightarrow\; \Xi(P_1) \sqsubseteq_F \Xi(P_2)$$

Cela signifie qu'en raffinant une composante (*ie*, un sous-processus) C d'un processus P, on raffine du même coup P.

On a aussi que, pour tout P et Q

$$CHAOS \sqsubseteq_F P$$

$$P \sqcap Q \sqsubseteq_F P$$

$$P \sqcap Q \sqsubseteq_F Q$$

On voit donc que *Spec* a été construit en considérant `Main2` et `Main3` et la monotonicité. Puisque

$$CANREFUSE(\{a, c, d\}) \sqsubseteq_F (\Box\, x : S \bullet x)$$

avec $S \subseteq \{a, c, d\}$, on a que

$$a \rightarrow (b \rightarrow STOP \;\Box\; CANREFUSE(\{a, c, d\}) \sqsubseteq_F Main2, Main3$$

L'expression suivante est un patron fréquent:

$$Prop \;|||\; CHAOS(B) \sqsubseteq_F P \setminus C$$

où B dénote les actions où *Prop* n'impose pas d'ordre particulier, et C sont les actions qui surviennent entre les actions de *Prop* dans P, mais dont on ne veut pas tenir compte dans *Prop*, par souci de simplicité.

On peut aussi vérifier qu'un processus P accepte une trace $s = [\sigma_1, \sigma_2, \ldots]$ de la manière suivante

$$P \sqsubseteq_T \sigma_1 \rightarrow \sigma_2 \rightarrow \ldots \rightarrow STOP$$

Cela ne garantit pas que cette trace est toujours possible, à cause du non-déterminisme qui pourrait survenir dans P (*ie*, choix interne, etc). Une possibilité est d'utiliser le patron suivant

```
Spec3 =
    ANYBUT(SIGMA,a)
[]  a -> (
    ANYBUT(SIGMA,b)
[]  b -> (
    ANYBUT(SIGMA,c)
[]  c -> CHAOS(SIGMA)
))

ANYBUT(S,x) = (|~| y : diff(S,{x}) @ y -> CHAOS(S)) |~| STOP
```

```
Main5 = a -> b -> c -> STOP [] b -> c -> STOP
Main6 = a -> b -> c -> STOP [] a -> c -> STOP
Main7 = a -> (b -> c -> STOP |~| d->STOP)

assert Spec3 [F= Main5 -- VRAI
assert Spec3 [F= Main6 -- FAUX : après a peut refuser b,
                       --  à cause du nondéterminisme de Main6
assert Spec3 [F= Main7 -- FAUX : après a peut refuser b,
                       --  à cause du nondéterminisme de Main7
```

Ces exemples sont disponibles dans le fichier fd-ref-exemple-v2.csp. Pour d'autres exemples, voir le fichier properties.csp qui donne des propriétés vérifiées sur la spécification de la bibliothèque. Notons finalement que FDR2 permet de vérifier qu'un processus ne contient pas d'impasse (*ie*, *deadlock*, un état où aucune transition n'est possible) et de boucle infinie sur τ (*ie*, *livelock*).