

## CHAP. 6 : TEST DE PROCESSUS

**Exercice Préliminaire** : comment tester si une implémentation satisfait la spécification ci-dessous?

---

```

process Pack2StupideAvecControle
  [input, output, control] (Max : Nat) :
exit :=
  ( control?NewMax:Nat [NewMax = 0] ; exit
    [ ]
    control?NewMax:Nat [NewMax > 0] ;
      Pack2StupideAvecControle[input, output,
                                control] (NewMax )
    [ ]
    input?M1:Message ; input?M2:Message ;
      ([taille(M1) + taille(M2) ≤ Max] ->
        output!ajouter(ajouter(vide, M1), M2);
        Pack2StupideAvecControle[input, output,
                                  control] (Max )
      [ ]
        [taille(M1)+taille(M2) > Max] ->
          output!ajouter(vide,M1)!ajouter(vide,M2);
          Pack2StupideAvecControle[input, output,
                                    control] (Max)
      )
    )
endproc (*Pack2StupideAvecControle*)
  
```

---

## LE PROBLÈME

- On a une spécification Lotos SP
- On a une implémentation I (de SP ???)
- Comment tester I pour vérifier qu'elle est conforme à SP ?

## LES QUESTIONS

- Quelles données choisir?
- Comment les soumettre?
- Comment décider si tout s'est bien passé?  
(et il peut y avoir de l'indéterminisme!)

## UNE PARTIE DES RÉPONSES

- On teste I par des processus “**testeurs**” que l'on construit à partir de la spécification
- On exécute I et chaque testeur T en parallèle

**I || T**

et on observe ce qui se passe (blocages, actions observables)

## PRINCIPES DU TEST À PARTIR D'UNE SPÉCIFICATION FORMELLE

- Pour construire les testeurs et décider du succès de  $I \parallel T$ , on a besoin de définir ce qu'est une implémentation correcte :  $I \text{ conf } SP$
- À partir de la définition de conf et de SP, on cherche à définir un jeu de test "exhaustif", tel que :

*$I \text{ passe } TestExhaust(SP) \text{ avec succès} \Leftrightarrow I \text{ conf } SP$*



- $I$  doit être "testable" :

*$I \text{ testable} \Rightarrow (I \text{ passe } TestExhaust(SP) \text{ avec succès} \Leftrightarrow I \text{ conf } SP)$*

NB : dans le cas des processus, testeur = test

## EXEMPLE D'HYPOTHÈSES DE TESTABILITÉ

- L'implémentation sous test est déterministe (ou raisonnablement non déterministe)
- Les actions de la spécification sont toutes implémentées, et de manière atomique

### VALIDITÉ ET NON BIAIS

#### Validité :

$I \text{ testable} \Rightarrow (I \text{ passe } TestExhaust(SP) \text{ avec succès} \Rightarrow I \text{ conf } SP)$

"On détecte toutes les implémentations non conformes"

#### Non Biais

$I \text{ testable} \Rightarrow (I \text{ passe } TestExhaust(SP) \text{ avec succès} \Leftarrow I \text{ conf } SP)$

"On accepte toutes les implémentations conformes"

## COMMENT SÉLECTIONNER UN SOUS-ENSEMBLE FINI DE TESTS?

- On fait des hypothèses plus fortes sur l'implémentation sous test : *hypothèses de sélection*
- On préserve la validité et le non biais (NB : sélection d'un sous-ensemble de TestExhaust(SP) => le non biais est préservé)
- **Exemples : uniformité, régularité, équité, etc**
  - Comment les trouver?
    - règles ou normes de programmation
    - expériences précédentes dans les mêmes conditions
    - autres tests, statiques ou dynamiques

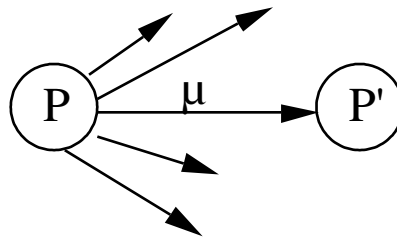
## DÉFINITIONS PRÉLIMINAIRES (LOTOS DE BASE)

$\mu$  est une action quelconque

$a$  est une action observable

$s$  est une séquence d'actions observables

- $P \xrightarrow{\mu} P'$  :  $P$  peut faire  $\mu$  et être ensuite dans l'état  $P'$



- $=\varepsilon \Rightarrow$  fermeture réflexive et transitive de  $\mathbf{i}$   
- on a toujours  $P =\varepsilon \Rightarrow P$

- $P \xrightarrow{as} P'$  :

$\exists Q$  et  $Q'$  t.q.  $P =\varepsilon \Rightarrow Q \xrightarrow{a} Q' =\varepsilon \Rightarrow P'$

(on peut avoir des suites de  $\mathbf{i}$  partout)

- $P \xrightarrow{s} P'$  :  $s$  est exécutable depuis  $P$

$\exists P'$  t.q.  $P \xrightarrow{s} P'$

NB: il peut y en avoir plusieurs

- $P \neq /s \Rightarrow$  : *s n'est pas exécutable depuis P*

## TRACES ET REFUS

- $\text{Traces}(P) = \{s \in L^* \mid P \neq /s \Rightarrow\}$



où L est l'ensemble des actions observables

- $P \text{ after } s = \{P' \mid P \neq /s \Rightarrow P'\}$

*ensemble des états possibles après s*

- $\text{Ref}(P, s) = \{X \mid \exists P' \text{ dans } P \text{ after } s \text{ t.q.}$   
 $\forall a \in X, P' \neq /a \Rightarrow\}$

*C'est un ensemble d'ensembles d'actions*

- Soit  $X = \{a_1, \dots, a_n\}$

- Si  $X \in \text{Ref}(P, s)$  le processus

$$P \parallel s ; \left[ \right]_{i=1, \dots, n} a_i$$

*peut bloquer après s*



DONC

$$\text{Ref}(Q, a) = \{\emptyset, \{a\}, \{c\}, \{a,c\}, \{b\}, \{a,b\}\}$$

*“Q peut bloquer si on lui propose un de ces ensembles d’actions après a”*

$\text{Ref}(P, s)$  est *l’ensemble des refus* de P après s

## RELATION DE CONFORMITÉ

• **I conf SP** ssi

$$\forall s \in \text{Traces}(SP), \text{Ref}(I, s) \subseteq \text{Ref}(SP, s)$$

• autrement dit : *soit s une trace de SP ; si après avoir exécuté s, I peut bloquer sur un ensemble d’actions X, alors  $X \in \text{Ref}(SP, s)$*

- encore autrement dit :

Si une trace observable de SP,  $s$ , est exécutable par I et *peut* mener à un état où toutes les actions d'un ensemble A sont refusées

**Alors**

$s$  peut mener, à partir de SP, à un état où toutes les actions de A sont refusées

*“On n’a pas ajouté de blocage par rapport à la spécification”*

### EXEMPLE 1

- $SP := (a; b; \mathbf{stop} [ ] a; c; \mathbf{stop} )$
- $I := a; (b; \mathbf{stop} [ ] c; \mathbf{stop} )$
- $L = \{a, b, c\}$
- $\mathbf{Ref}(I, \varepsilon ) = \{\emptyset, \{b\}, \{c\}, \{b,c\}\} = \mathbf{Ref}(SP, \varepsilon )$
- $\mathbf{Ref}(I, a)$  et  $\mathbf{Ref}(SP, a)$

“SP **after** a” contient deux états

Pour  $\mathbf{Ref}(SP, a)$ , voir page 11 :  $\{\emptyset, \{a\}, \{c\}, \{a,c\}, \{b\}, \{a,b\}\}$

“I **after** a” contient un état

$$\text{Ref}(I, a) = \{\emptyset, \{a\}\}$$

- On a donc bien  $\text{Ref}(I, a) \subseteq \text{Ref}(SP, a)$

C'est à vérifier pour toutes les traces de SP exécutables par I (ici, toutes), sans oublier  $\varepsilon$

- $\text{Ref}(I, a; b) = P(L) = \text{Ref}(SP, a; b)$
- $\text{Ref}(I, a; c) = P(L) = \text{Ref}(SP, a; c)$

DONC I **conf** SP,

mais pas l'inverse car  $\text{Ref}(I, a) \neq \text{Ref}(SP, a)$

## Exemple 2

- $SP := i; a; \text{stop}$
- $I := (b; \text{stop} [ ] i; a; \text{stop} )$
- $L = \{a, b, c\}$

ATTENTION : I **after**  $\varepsilon = \{I, I'\}$  avec

$$I' = a; \text{stop}$$

$\text{Traces}(SP) = \{\varepsilon, a\}$ , toutes exécutables par I

- $\text{Ref}(I, \varepsilon) = \text{ce qu'on refuse en } I \text{ et ce qu'on refuse en } I' = \{\emptyset, \{c\}\} \cup \{\emptyset, \{b\}, \{c\}, \{b,c\}\} = \{\emptyset, \{b\}, \{c\}, \{b,c\}\} = \text{Ref}(SP, \varepsilon)$
- $\text{Ref}(I, a) = P(L) = \text{Ref}(SP, a)$

DONC  $I \text{ conf } SP$ , ET l'inverse!

### EXEMPLE 3

- $SP := (b;c; \text{stop } [ ] i; a; \text{stop})$
- $I := i; a; \text{stop}$
- $L = \{a, b, c\}$

Traces de SP exécutables par I :  $\{\varepsilon, a\}$

- $SP \text{ after } \varepsilon = \{SP, SP'\}$  avec  $SP' = a; \text{stop}$
  - $I \text{ after } \varepsilon = \{I, I'\}$  avec  $I' = a; \text{stop}$
- $\text{Ref}(SP, \varepsilon) = \{\emptyset, \{b\}, \{c\}, \{b,c\}\}$

$$\text{Ref}(I, \varepsilon) = \{\emptyset, \{b\}, \{c\}, \{b,c\}\}$$

- $\text{Ref}(I, a) = P(L) = \text{Ref}(SP, a)$

DONC  $I \text{ conf } SP$ , et l'inverse!

MAIS ON N'A PAS :

$(b; \text{stop } [ ] i; a; \text{stop}) \text{ conf}$

$(b; c; \text{stop } [ ] i; a; \text{stop})$

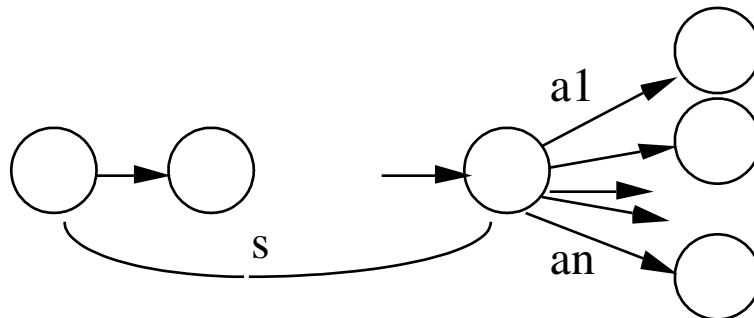
car après  $b$  le premier bloque sur  $c$

$\Rightarrow \text{conf}$  n'est pas transitive

## TEST EXHAUSTIF POUR conf

$$\text{exhaust}(SP) = \{s ; [ ]_{ai \in A} ai; \text{stop} \mid$$

$$s \in \text{Traces}(SP), A \subseteq L\}$$



**NB**, en Lotos complet:  $L = \{g!v \mid g \text{ port visible de } SP, v \text{ valeur d'un type de } SP\}$

**Rappel** :  $g!exp1 \parallel g!exp2$  s'exécute si  $exp1 = exp2$

**Verdict** de  $I \parallel T$  avec  $T \in \text{exhaust}(SP)$  :

- on atteint un **stop** de  $T \Rightarrow \text{succès}$
- on bloque avant
  - on bloque avant la fin de  $s \Rightarrow \text{succès}$   
( $s$  n'est pas exécutable par  $I$ ) ou verdict *inconclusif*
  - on bloque à la fin de  $s$ 
    - si  $SP$  peut bloquer à la fin de  $s$

$\Rightarrow$  succès

- sinon  $\Rightarrow$  échec

## AMÉLIORATION DE EXHAUST, TESTABILITÉ

- Les tests tels que  $A \in \text{Ref}(SP, s)$  sont toujours des succès!

$$\text{exhaust}'(SP) = \{s ; [ ]_{ai \in A} ai; \text{stop} \mid s \in \text{Traces}(SP), A \subseteq L, A \notin \text{Ref}(SP, s)\}$$

- il y a aussi des redondances sur les sous-séquences qui ... ne bloquent pas

*Que signifierait le succès de tous les tests de exhaust(SP) ?*

**Cela implique que I conf SP seulement si**

- I a les mêmes actions observables que SP (sinon on a des blocages non significatifs)

- Le non-déterminisme de I est “équilibré” et on passe chaque test “un nombre suffisant de fois”...

*Exemple d’Hypothèse de testabilité* : Après p exécutions d’un test tous les comportements possibles ont été couverts.

p dépend de |s| et du nb max de choix possibles à chaque action de I.

*Autre exemple* : I prend toujours le même choix...

## OPTIMISATION DES TESTS

On factorise :

Exemple  $SP := a ; b ; c ; \mathbf{stop}$

$\text{exhaust}'(SP) = \{$

*Après  $\varepsilon$  on doit faire  $a$  :*      **$a ; \mathbf{stop}$**  ,

*Après  $a$  on doit faire  $b$  :*    **$a ; b ; \mathbf{stop}$**  ,

*Après  $b$  on doit faire  $c$  :*    **$a ; b ; c ; \mathbf{stop}$**  }

Verdict : on peut bloquer partout, sauf sur les actions en gras

Jeu de test optimisé, en changeant le verdict :

**$\{ a ; b ; c ; \mathbf{stop} \}$**

**Verdict : il faut atteindre le  $\mathbf{stop}$**

### Règle de factorisation :

Toute trace qui est préfixe d'une autre trace présente dans le jeu de test peut être retirée du jeu de test en aménageant le verdict pour que

les absences de blocage qu'elle testait soit  
toujours testées

**Attention :  $\varepsilon$  est toujours une trace!**

## À PROPOS D' $\epsilon$

$SP := (i ; a ; \text{stop} [] i ; b ; \text{stop})$

$a ; \text{stop}$  avec verdict "il faut atteindre le **stop**"  
est un test biaisé : on peut refuser  $a$  après  $\epsilon$

$b ; \text{stop}$  est aussi biaisé avec ce verdict...

Test non biaisé :  $(a ; \text{stop} [] b ; \text{stop})$  avec  
verdict "il faut atteindre un **stop**" est un test  
non biaisé (et valide)

**Donc, on détermine**

Mais pas seulement dans le cas d' $\epsilon$  !

Autre exemple :

$SP := (i ; a ; c ; \text{stop} [] i ; a ; b ; \text{stop})$

$(a ; (c ; \text{stop} [] b ; \text{stop}))$  avec verdict "il faut  
atteindre un **stop**" est un test non biaisé (et  
valide)

$SP := (i ; a ; c ; \text{stop} [] i ; a ; (b ; \text{stop} [] i ; d ; \text{stop}))$

test : a; (c;stop[]b;stop[]d;stop))

## Exemple 1 de test : cas non disjoints, avec hypothèses d'uniformité

---

```

process P[g,h]: noexit:=
  (g?x:int[x<10]; g!x; stop
  [ ]
  g?y:int[y>5]; h!y; stop)
endproc

```

---

Traces(P) = { $\epsilon$  ,  $\langle g,i \rangle$ ,  $\langle g,i \rangle \langle g,i \rangle$   $i < 10$ ,  
 $\langle g,i \rangle \langle h,i \rangle$   $i > 5$  }

Ref(P,  $\epsilon$  ) = P({h!i | i:int})

Ref(P,  $\langle g,i \rangle$ ) =

- si  $5 < i < 10$ , P(tout sauf {h!i , g!i})
- si  $i \leq 5$ , P(tout sauf g!i)
- si  $i \geq 10$ , P(tout sauf h!i)

Ref(P,  $\langle g,i \rangle \langle g,i \rangle$   $i < 10$ ) = P(tout)

Ref(P,  $\langle g,i \rangle \langle h,i \rangle$   $i > 5$ ) = P(tout)

Jeu de test possible : 3 tests qui doivent terminer

---

- g!v1; (g!v1; **stop** [ ] h!v1; **stop**) (\*  $5 < v1 < 10$ \*)
  - g!v2; g!v2; **stop** (\*  $v2 \leq 5$ \*)
  - g!v3; h!v3; **stop** (\*  $v3 \geq 10$ \*)
- 

avec hypothèses d'uniformité sur v1, v2, v3

## Exemple 2 : cas disjoints avec blocage, hypothèses d'uniformité et de régularité

---

```

process Q[g]: noexit:=
    g?x:int; ( [x<10]—> g!x; stop (*ou exit*)
              [ ]
              [x>20]—> Q[g])
endproc

```

---

Traces(Q) =

{ $\epsilon$  ,  $\langle g, i \rangle$   $i$  quelconque,  $\langle g, i \rangle \langle g, i \rangle$   $i < 10$ ,  
 $\langle g, i \rangle s$   $i > 20$  et  $s \in \text{Traces}(Q)$  }

Jeu de test possible :  $n$  tests qui doivent terminer

- 
- $g!v1; g!v1; \mathbf{stop}$  (\* ou **exit**,  $v1 < 10^*$ )
  - $g!v2; g!v3; g!v3; \mathbf{stop}$  (\* ou **exit**,  $v2 > 20$ ,  
 $v3 < 10^*$ )
  - $g!v4; g!v5; g!v6; g!v6; \mathbf{stop}$  (\* etc\*)
- 

*Choix de  $n$*  : hypothèse de régularité

Hypothèses d'uniformité sur  $v1, v2, \dots$

*Forme générale des tests :*

**Tests(Q) =**

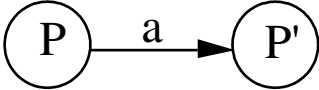
**{g!v1; g!v1; stop |v1<10 } ∪ {g!  
v2;T |v2>20, T∈Tests(Q) }**

## RÈGLES GÉNÉRALES POUR CONSTRUIRE Tests(P)

(en Lotos de base, se généralisent à Lotos complet)

**Tests(stop) = {∅}**

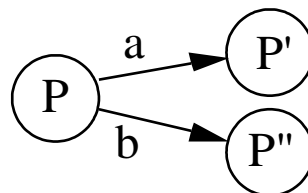
**Tests(exit) = {exit}**

$P := a; P'$   **Tests(P) = {a ; T | T∈ Tests(P')}**  
On ne doit pas bloquer sur a

$P := i; P'$   **Tests(P) = {Tests(P')}**

**Cas plus compliqué : [ ]**

1 - non-déterminisme contrôlable  $a \neq b$  :



$$\text{Tests}(a;P' [ ] b;P'') = \{a;T', b; T'' \mid T' \in \text{Tests}(P'), \\ T'' \in \text{Tests}(P'')\}$$

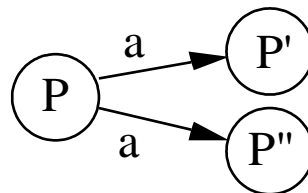
On ne doit bloquer ni sur a, ni sur b.

*Attention : l'implémentation ne doit pas pouvoir contrôler le testeur!*

- $a; T' [ ] b; T''$  n'est pas un test complet.
- $i; a; T' [ ] i; b; T''$  l'est, mais doit être passé "un nombre suffisant de fois" (donc à éviter).

## 2 - non-déterminisme non contrôlable :

### 2.1 : factoriser les cas communs



$$\text{Tests}(a;P' [ ] a;P'') = \{a; (T' [ ] T'') \mid T' \in \text{Tests}(P'), \\ T'' \in \text{Tests}(P'')\}$$

On ne doit pas bloquer sur a.

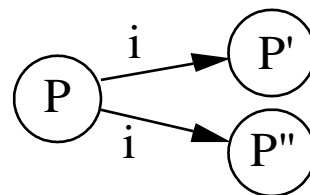
*mais aussi :*

$$\text{Tests}(a;P' [ ] i;a;P'') = \{a; (T' [ ] T'') \mid T' \in \text{Tests}(P'), \\ T'' \in \text{Tests}(P'')\}$$

**Condition :** les ensembles d'actions initiales de  $T'$  et  $T''$  sont disjoints et  $\neq i$ .

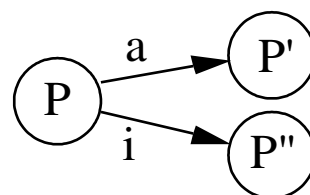
Sinon, on factorise comme ci-dessus, et on retire les  $i$  comme ci-dessous.

## 2.2 : supprimer les $i$



$\text{Tests}(i;P' [ ] i;P'') = \{(T' [ ] T'') \mid T' \in \text{Tests}(P'), T'' \in \text{Tests}(P'')\}$  si les ensembles d'actions initiales de  $T'$  et  $T''$  sont disjoints et  $\neq i$ . Ces tests autorisent l'implémentation à choisir arbitrairement une branche.  
Sinon, on factorise et on retire les  $i$ .

## Supprimer les $i$ (suite)



$\text{Tests}(a; P' [ ] i; P'') = \{(a; T' [ ] T''), T'' \mid T' \in \text{Tests}(P'), T'' \in \text{Tests}(P'')\}$

avec les mêmes conditions sur  $T'$  et  $T''$ .

**Attention :** si  $a$  n'est pas une action initiale de  $P''$ , elle peut être refusée  $\Rightarrow a; T'$  peut bloquer sur  $a$ . *Ces tests autorisent l'implémentation à refuser  $a$ .*

**NB :** un seul  $T''$  dans les tests  $(a; T' [ ] T'')$  suffit.

### Exemple 3 : où on a un peu de tout

---

**process**  $P[g, h]$ : **noexit**:=

$(g?x[x<20]; h!x+2; \text{stop} [ ] i; g?y[y>10]; h!y; \text{stop})$

**end proc**

---

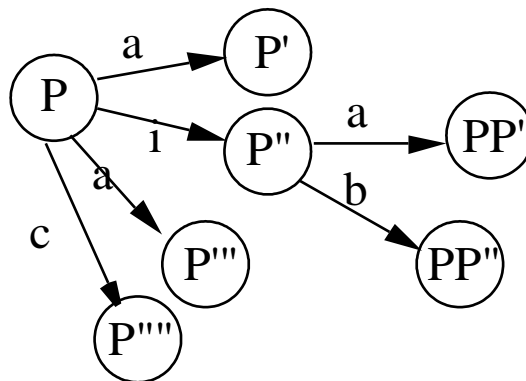
suppression de **i**, factorisation des cas communs :

---

- $(g!v1; h!v1+2; \text{stop} (* v1 \leq 10, g!v1 \text{ peut être refusé*})$   
 $[ ] g!v2; h!v2; \text{stop}) (* v2 \geq 20, \text{ toujours accepté*})$
  - $(g!v1; h!v1+2; \text{stop} (* v1 \leq 10, g!v1 \text{ peut être refusé*})$   
 $[ ] g!v3; (h!v3+2; \text{stop} [ ] h!v3; \text{stop})) (* 10 < v3 < 20*)$
  - $g!v2; h!v2; \text{stop} (* v2 \geq 20*)$
  - $g!v3; (h!v3+2; \text{stop} [ ] h!v3; \text{stop}) (* 10 < v3 < 20*)$
-

## Exemple récapitulatif 3bis

(heureusement, de tels cas sont rares en pratique!)



$P = (a; P' \mid i; (a; PP' \mid b; PP'') \mid a; P''' \mid c; P''''))$

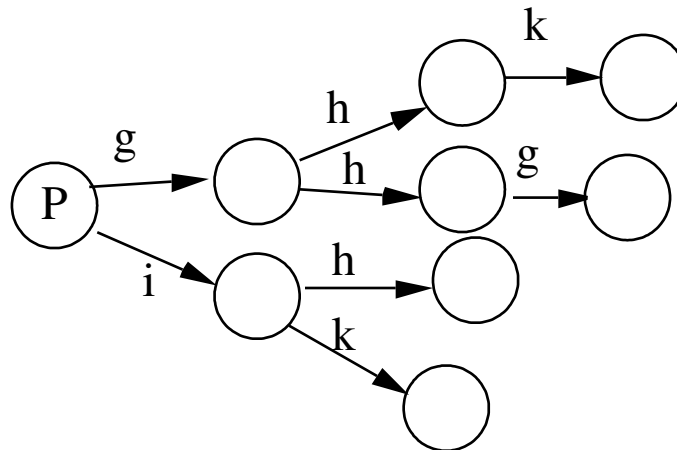
*Les actions observables possibles à partir de P sont a, b, c.*

- a est toujours acceptée et peut mener à P', PP' ou P'''
- b est toujours acceptée et mène à PP''
- c peut être refusée (en P''). Si elle est acceptée elle mène à P''''

$\text{Tests}(P) = \{ a; (\text{Tests}(P') \mid \text{Tests}(PP') \mid \text{Tests}(P''')),$   
 $b; \text{Tests}(PP''),$   
 $(c; \text{Tests}(P'''')) \mid$   
 $a; (\text{Tests}(P') \mid \text{Tests}(PP') \mid \text{Tests}(P''')),$   
 $(c; \text{Tests}(P'''')) \mid b; \text{Tests}(PP'') \} )$

**NB** : 3 tests suffisent. Un des 2 derniers est redondant.

## Exemple récapitulatif 3ter




---

$P[g, h, k] := (g; (h; k; \mathbf{stop}[]h; g; \mathbf{stop})$   
 $[]$   
 $i; (h; \mathbf{stop}[]k; \mathbf{stop}) )$

---

*Actions initiales possibles : g, h, k*

- g peut être refusée
- h doit être factorisée après g

Tests(P), version 1 :

---

$h; \mathbf{stop}$

$k; \mathbf{stop}$

$(g; h; (k; \mathbf{stop}[]g; \mathbf{stop}) []h; \mathbf{stop})$

---

Tests(P), version 2 :

---

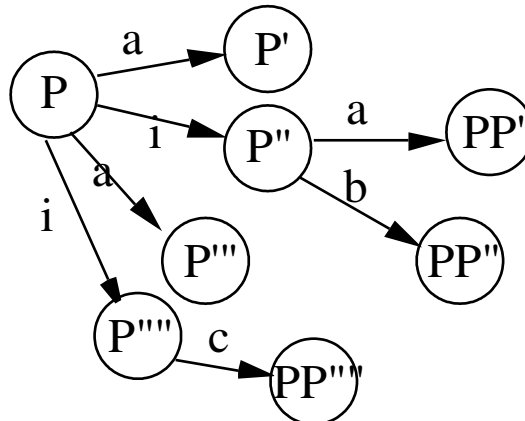
$h; \mathbf{stop}$

---

**k; stop**

**(g; h; (k; stop [ ] g; stop) [ ] k; stop)**

## exercice



$P = (a; P' [ ] i; (a; PP' [ ] b; PP'') [ ] a; P''' [ ] i; c; P''''))$

*Les actions observables possibles à partir de P sont a, b, c.*

- a peut être refusée (en P''''), mais alors c est acceptée) ; si elle est acceptée elle mène à P', PP' ou P'''
- b peut être refusée (en P''''), mais alors c est acceptée); si elle est acceptée elle mène à PP''
- c peut être refusée (en P'', mais alors a ou b sont acceptées); si elle est acceptée elle mène à PP''''

Tests possibles :

$$\begin{aligned} \text{Tests}(P) = \{ & (a; ( \text{Tests}(P') [ ] \text{Tests}(PP') [ ] \text{Tests}(P''')) \\ & [ ] c; \text{Tests}(PP'''')), \\ & (b; \text{Tests}(PP'') [ ] c; \text{Tests}(PP'''')), \end{aligned}$$

(c; Tests(PP''''') [ ] a; Test(PP')) }