Vérification formelle, compositionnelle et automatique de systèmes de composants

Nicolas Ayache

CEA-LIST Laboratoire de Sûreté du Logiciel
Directrice : Christine Paulin — LRI / INRIA Proval
Encadrant : Loïc Correnson — CEA-LIST LSL
Encadrant : Franck Védrine — CEA-LIST LMEASI

5 janvier 2010

Contexte

criticité : la vie d'êtres humains ou d'importantes sommes d'argent en dépendent

Description	Que fait le système ?
Spécification	Que doit faire le système ?
Vérification	Description satisfait Spécification ?

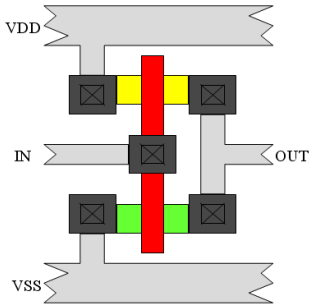
Contexte

criticité : la vie d'êtres humains ou d'importantes sommes d'argent en dépendent

Description	Que fait le système ?
Spécification	Que doit faire le système ?
Vérification	Description satisfait Spécification ?

Description de Systèmes

Autrefois : Dessin au micron



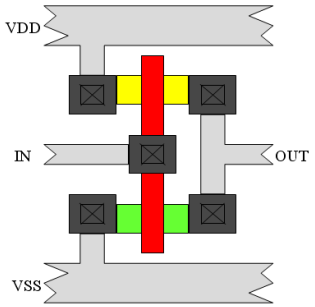
Aujourd'hui : Langages de Programmation

SystemC

- Récent (standardisé en 2005)
- Standard de fait
- Peu d'outils de vérification formelle

Description de Systèmes

Autrefois : Dessin au micron

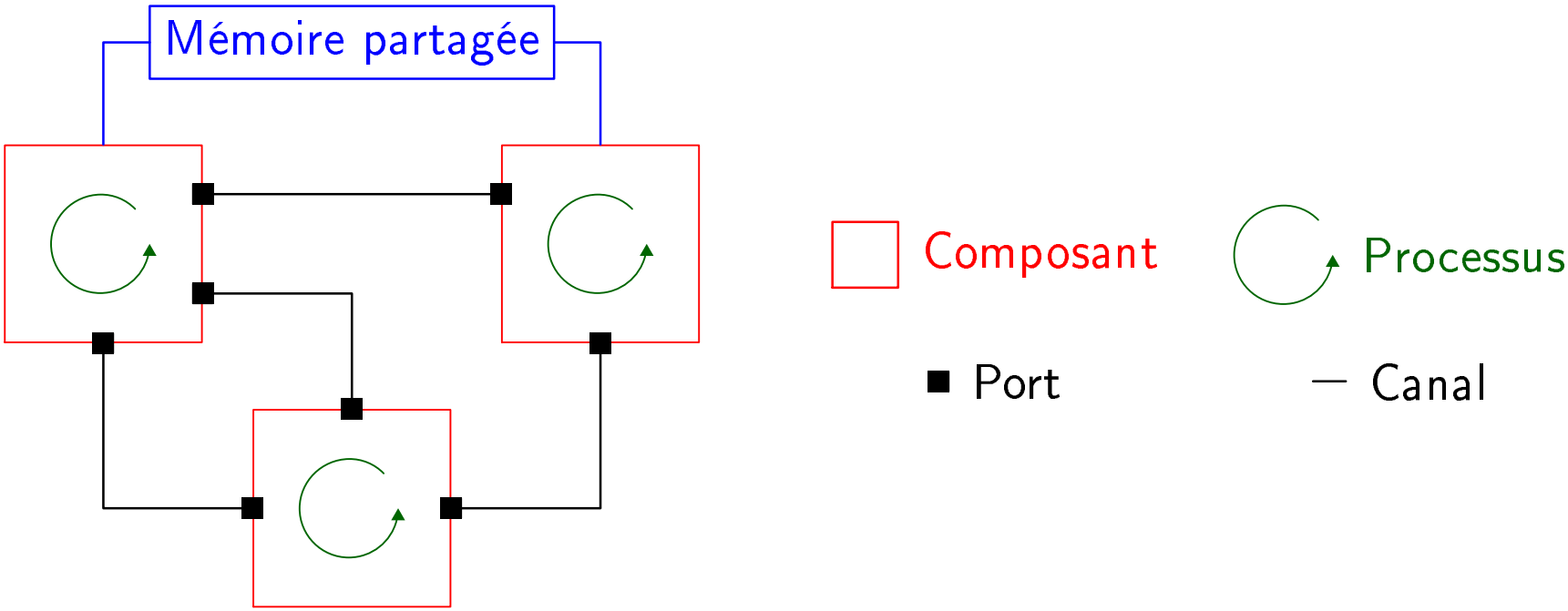


Aujourd'hui : Langages de Programmation

SystemC

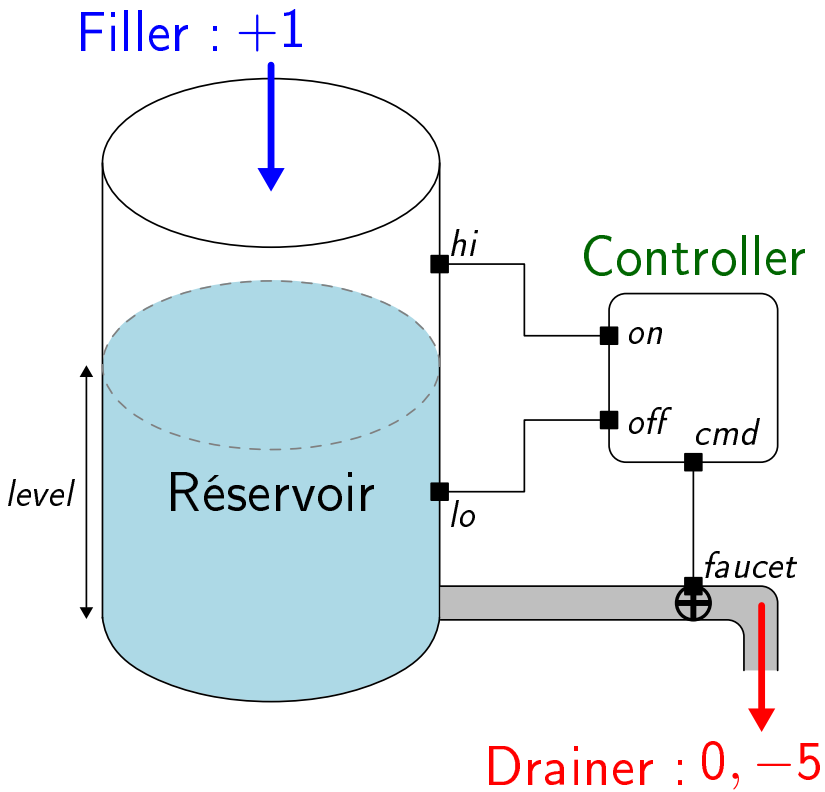
- Récent (standardisé en 2005)
- Standard de fait
- Peu d'outils de vérification formelle

Schéma d'un Système



Hypothèse Synchrone : les composants évoluent à la même vitesse

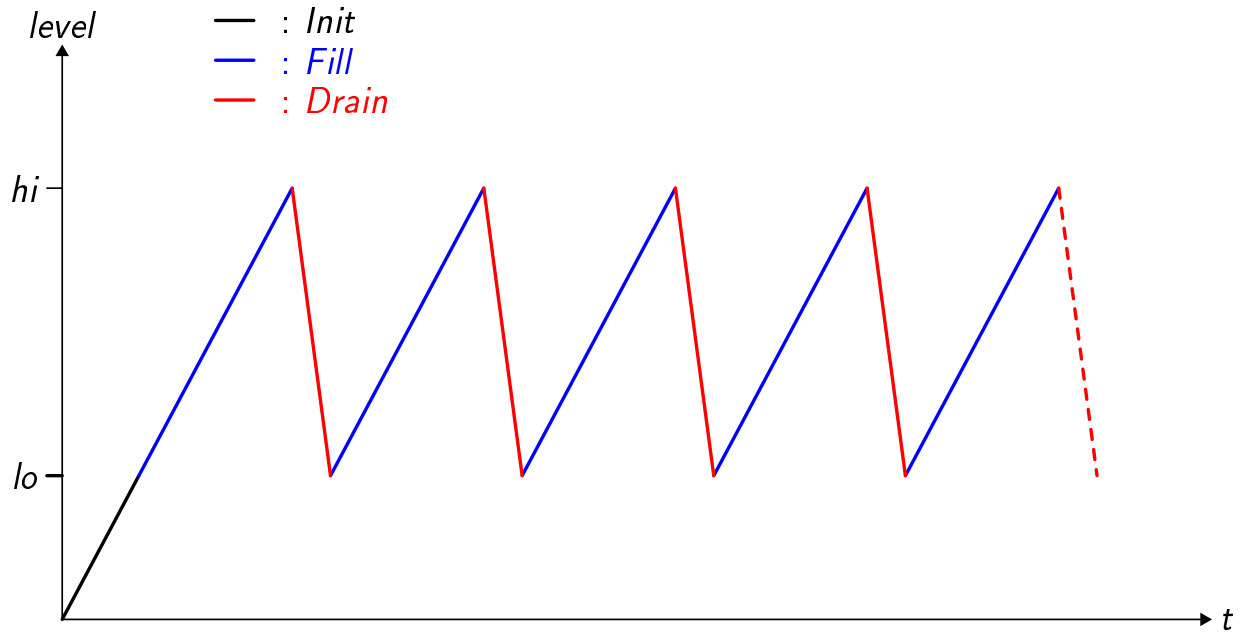
Système de la Pompe



3 processus :
Filler, Controller, Drainer

Évolution de *level* ?

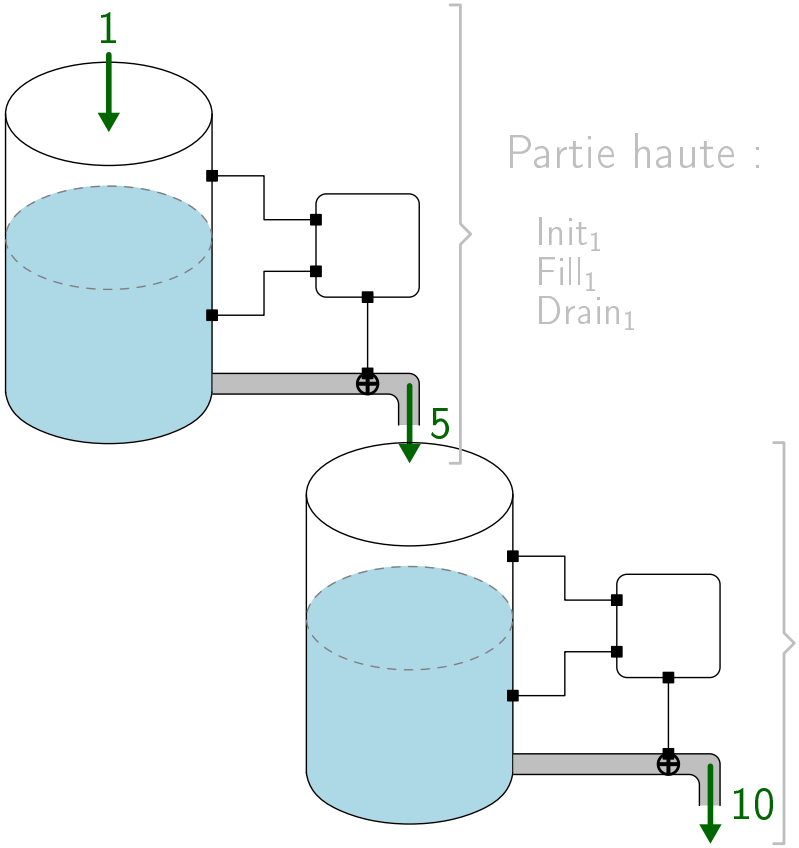
Évolution



$level\ (\pm\delta) \in [0\ ;\ hi]$?

Après initialisation, $level\ (\pm\delta) \in [lo\ ;\ hi]$?

Système des Pompes



Partie haute :

Init_1
Fill_1
Drain_1

Partie basse :

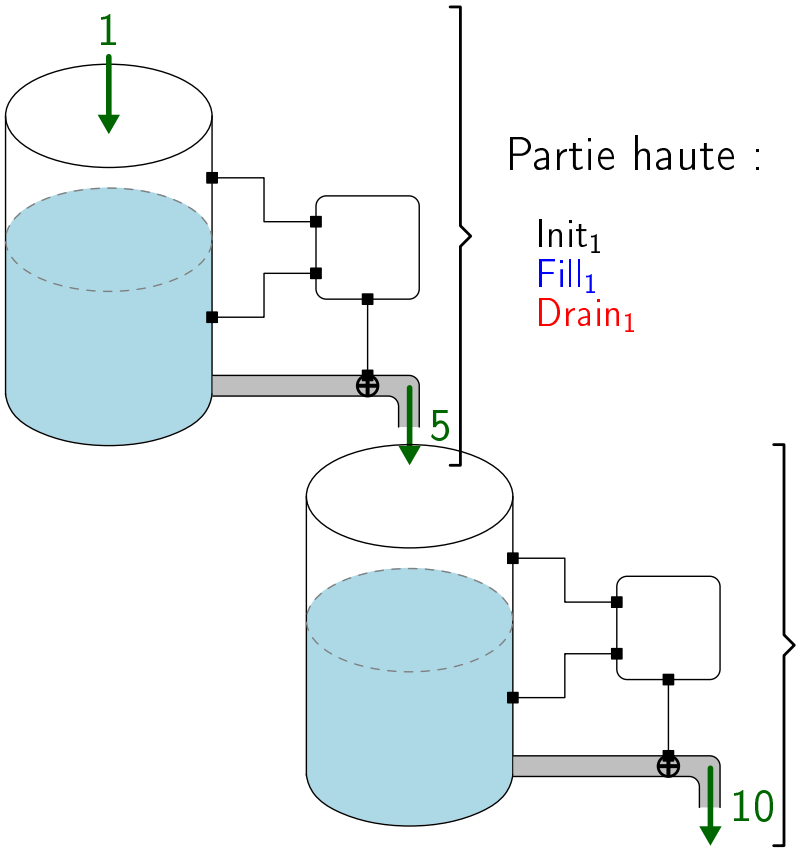
Init_2
Fill_2
Drain_2

Système complet :

Init_1	Init_2
Init_1	Fill_2
Init_1	Drain_2
Fill_1	Init_2
Fill_1	Fill_2
Fill_1	Drain_2
Drain_1	Init_2
Drain_1	Fill_2
Drain_1	Drain_2

Assemblage de systèmes $\Rightarrow$ Produit du nombre d'états

Système des Pompes



Partie haute :

Init_1
Fill_1
Drain_1

Partie basse :

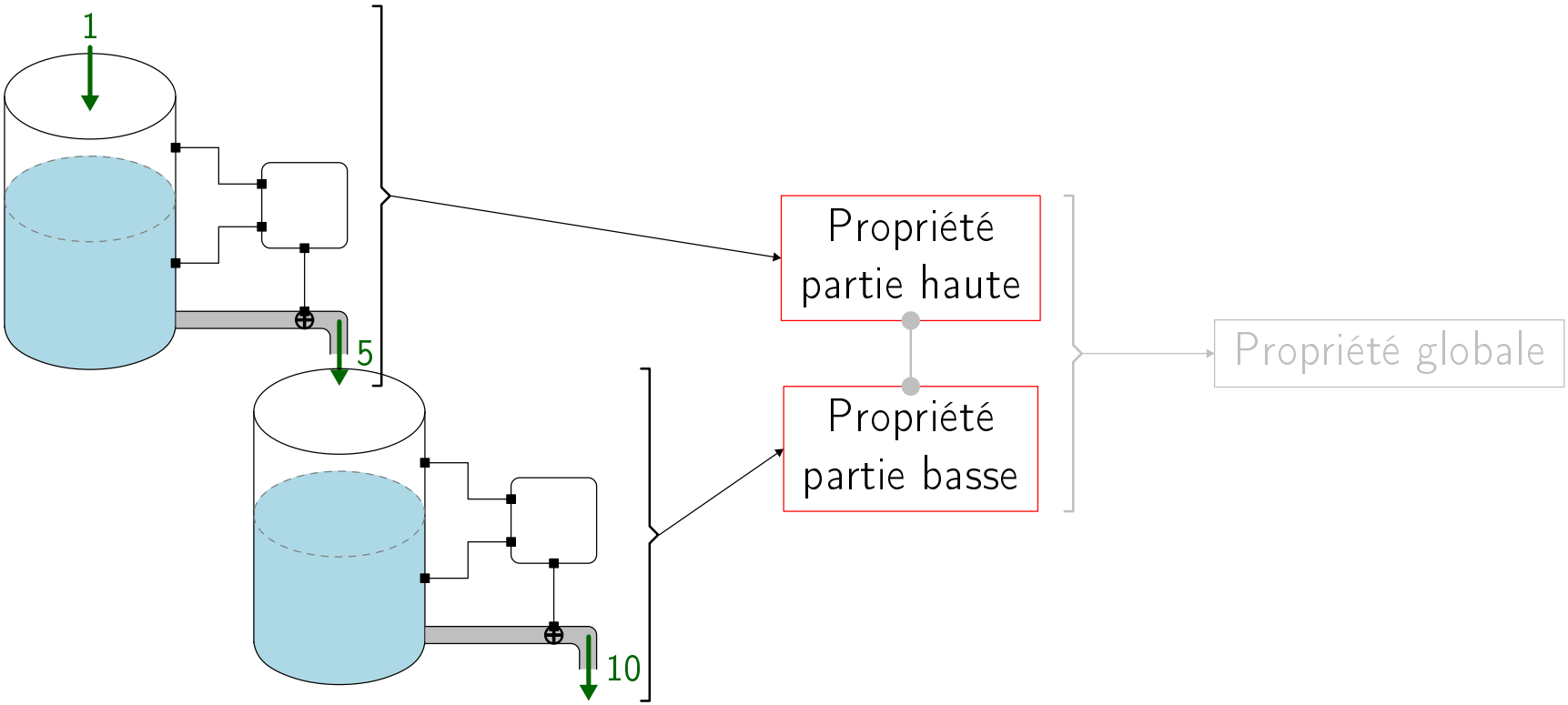
Init_2
Fill_2
Drain_2

Système complet :

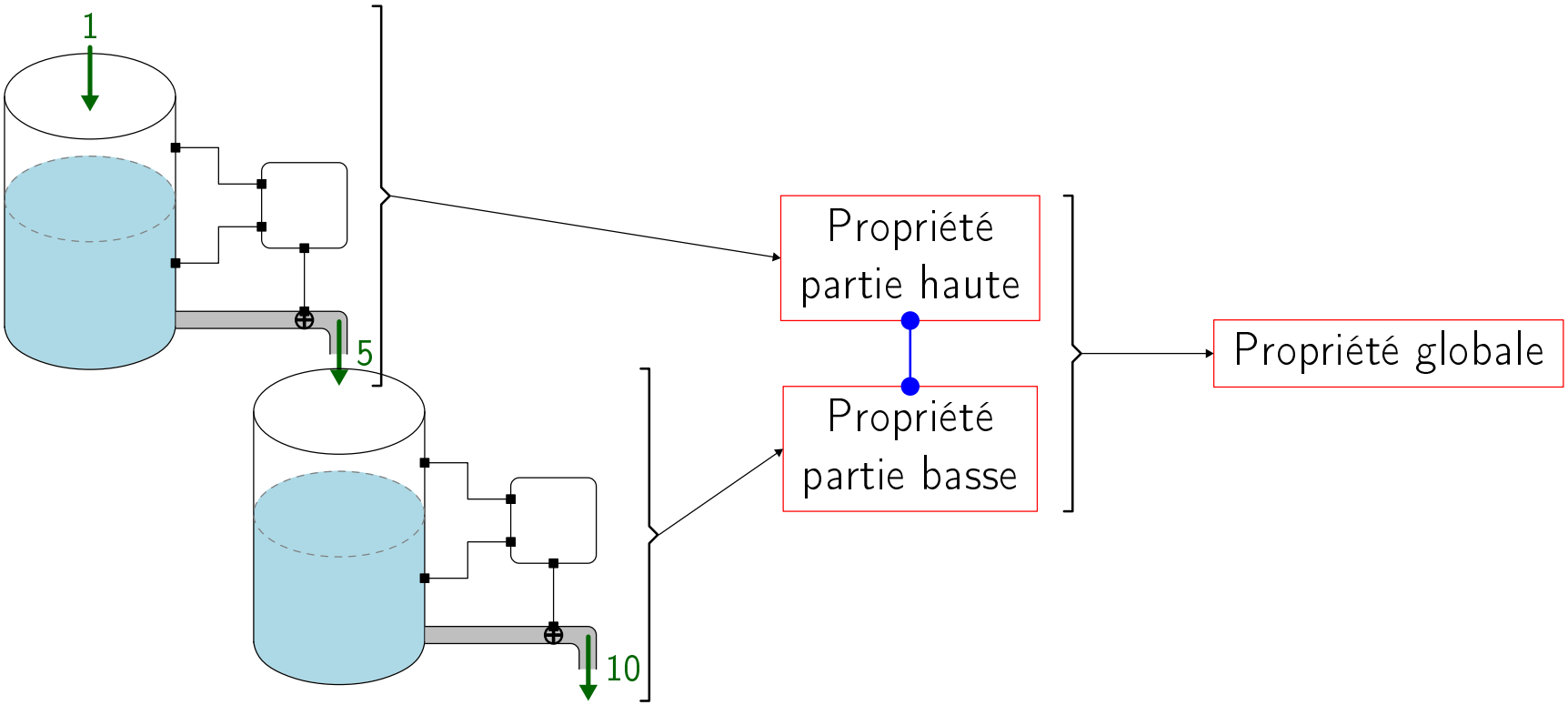
Init_1	Init_2
Init_1	Fill_2
Init_1	Drain_2
Fill_1	Init_2
Fill_1	Fill_2
Fill_1	Drain_2
Drain_1	Init_2
Drain_1	Fill_2
Drain_1	Drain_2

Assemblage de systèmes $\Rightarrow$ Produit du nombre d'états

Vérification Compositionnelle



Vérification Compositionnelle



Objectif

Description SystemC + Spécification
Vérification Compositionnelle

Accessibles à l'Ingénieur

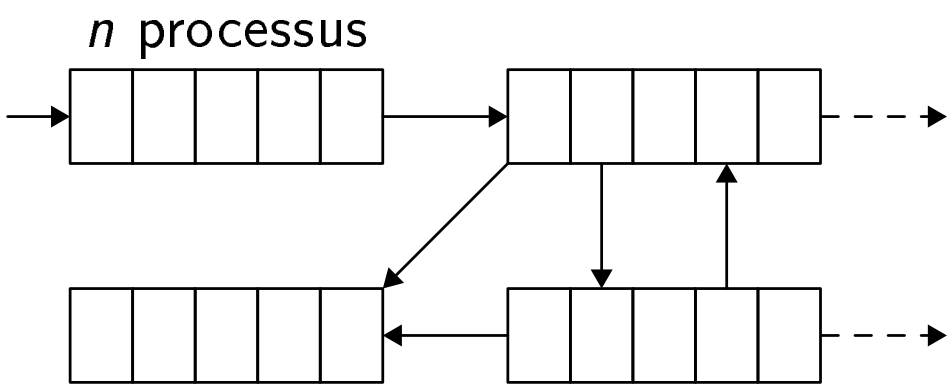
Difficultés liées à SystemC :

- Bibliothèque C++
- Aspects Logiciels et Matériels
- Synchrone et Asynchrone
- Indéterminisme

Model Checking

Système : séquences d'états

Spécification : proposition sur chemins d'états
(Logiques Temporelles)

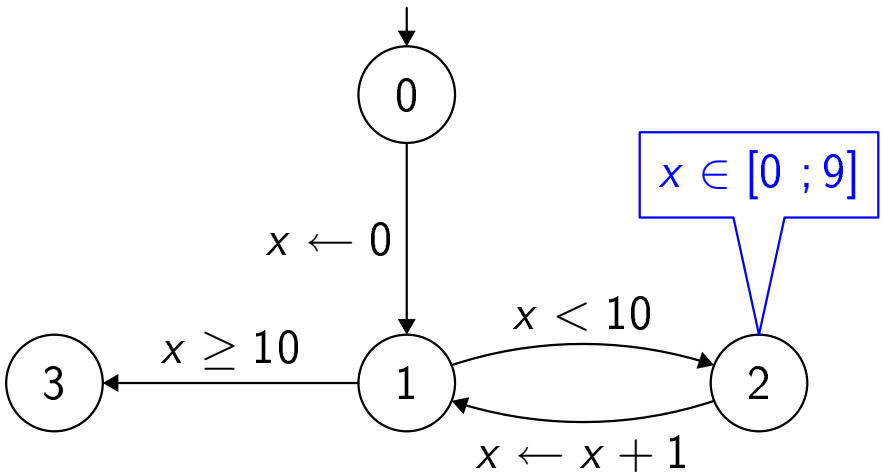


Problème : explosion combinatoire

Interprétation Abstraite

Théorie du Point Fixe

Programme : graphe de flot de contrôle (CFG)
Analyse : point de contrôle $\longrightarrow$ sur-ensemble environnements



Problème : parallélisme et concurrence

Sources d'Inspiration

Matthieu Moy (ACSD'05) : LusSy (+Nbac)

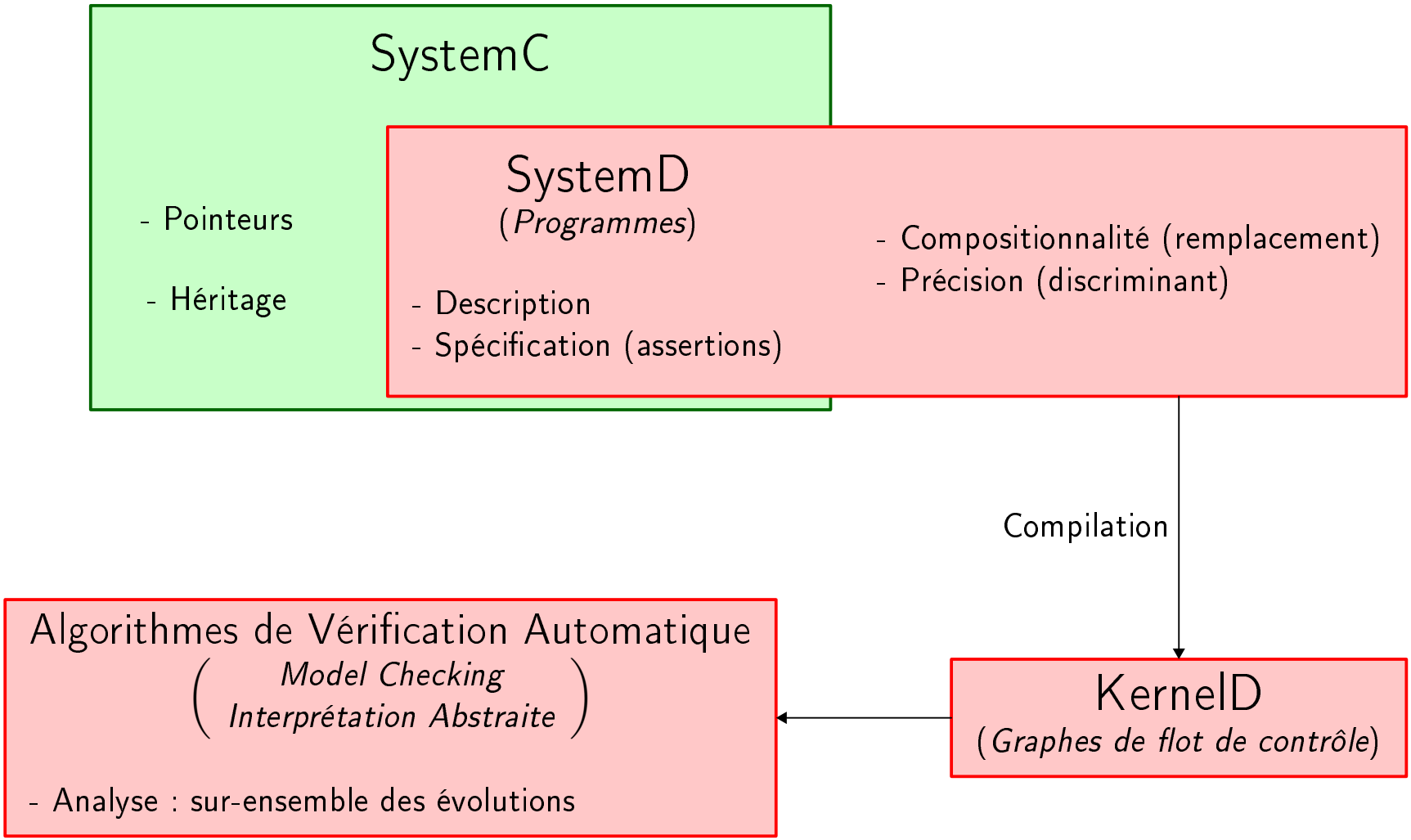
- SystemC
- Spécification (assertions)
- Vérification automatique

Pas de compositionnalité

Halbwachs et al. (AMAST'93)

- Cadre théorique
- Spécification (observateurs)
- Vérification compositionnelle

Synchrone et Déterministe


SystemC
- Pointeurs
- Héritage
SystemD
(*Programmes*)
- Description
- Spécification (assertions)
- Compositionnalité (remplacement)
- Précision (discriminant)
Compilation
Algorithmes de Vérification Automatique
(*Model Checking*
Interprétation Abstraite)
- Analyse : sur-ensemble des évolutions
KernelD
(*Graphes de flot de contrôle*)

Plan

Plan

<< Model Checking pour le Contrôle >>

<< Interprétation Abstraite pour la Mémoire >>

SystemD
Programmes

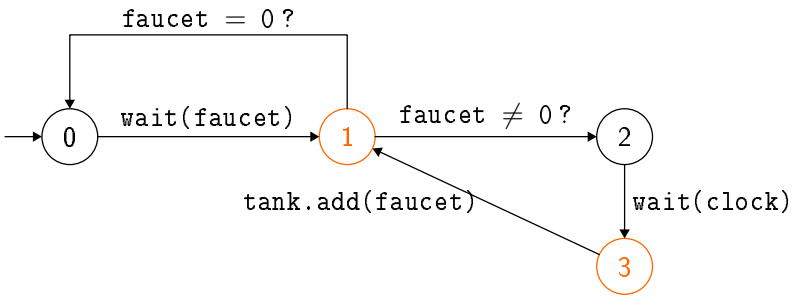
Compilation →

KernelD
CFGs de points d'attente

Filler/Drainer

```
while (true) {
 wait(faucet); (1)
 while (faucet != 0) {
  wait(clock); (3)
  tank.add(faucet); } }
```

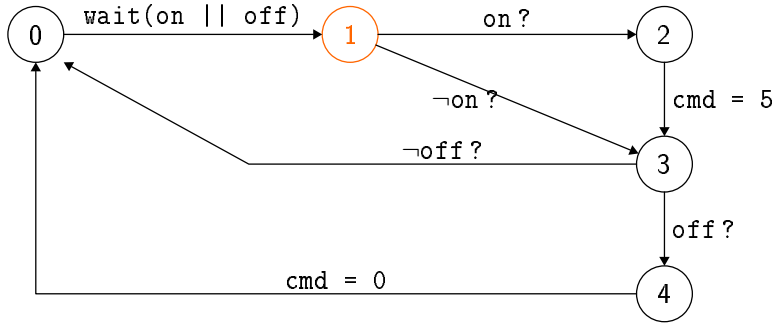
Filler/Drainer



Controller

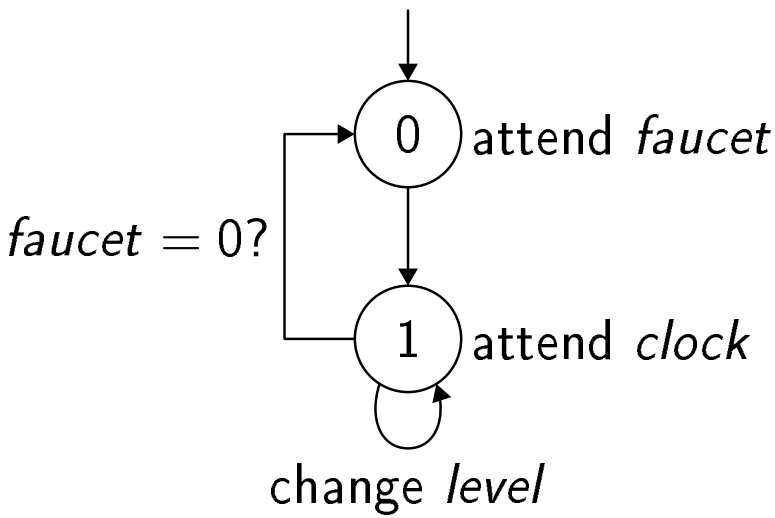
```
 while (true) {
  wait(on || off); (1)
  if (on) cmd = 5;
  if (off) cmd = 0; }
```

Controller

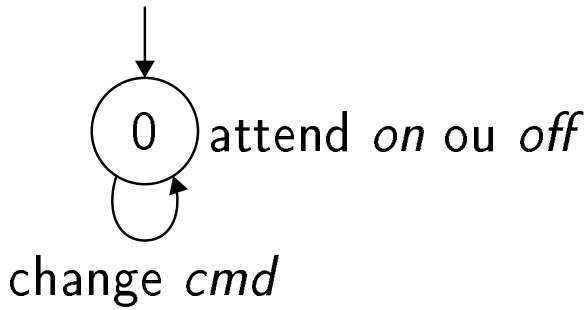


Points d'attente

Filler/Drainer



Controller



F_0 (resp. D_0) : Filler (Drainer) en 0 attend *faucet*
F_1 (resp. D_1) : Filler (Drainer) en 1 attend *clock*

Controller : un seul point d'attente, non mentionné dans la suite

État abstrait

État concret :

- Configuration : { point d'attente des processus ; évènements notifiés }
- Mémoire : valeur (ex : entier) pour chaque variable

Exemple : (F_1 D_0, $level = 300$)

État abstrait :

- Configuration
- Mémoire abstraite : valeur abstraite (ex : intervalles) pour chaque variable

Exemple : (F_1 D_0, $level \in [0\ ;1000]$)

État abstrait

État concret :

- Configuration : $\begin{cases} \text{point d'attente des processus} \\ \text{évènements notifiés} \end{cases}$
- Mémoire : valeur (ex : entier) pour chaque variable

Exemple : (F_1 D_0, $level = 300$)

État abstrait :

- Configuration
- Mémoire abstraite : valeur abstraite (ex : intervalles) pour chaque variable

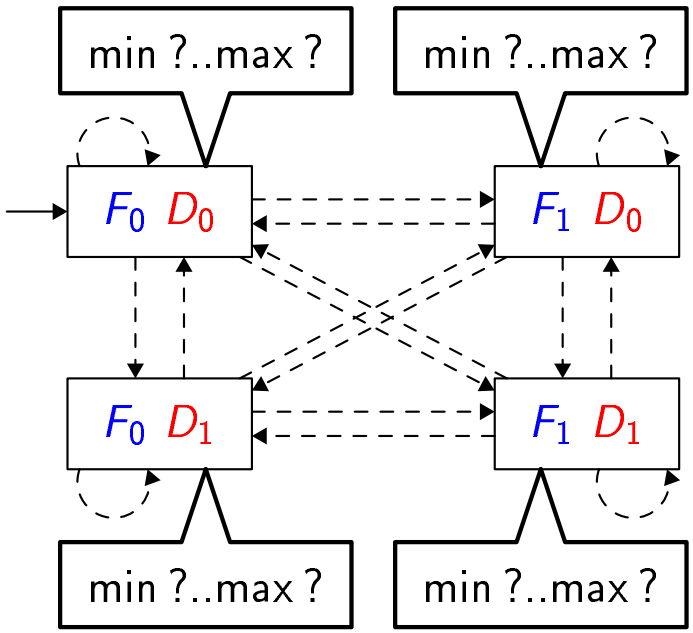
Exemple : (F_1 D_0, $level \in [0\,;1000]$)

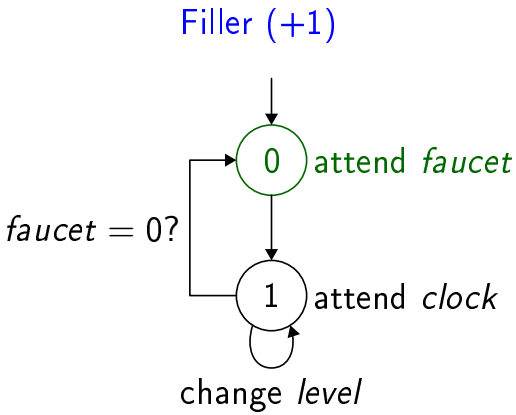
Graphe d'états abstraits

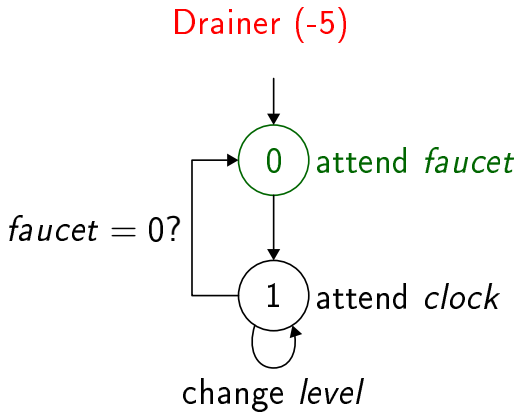
Sommet : configuration $\boxed{F_1\ D_0}$

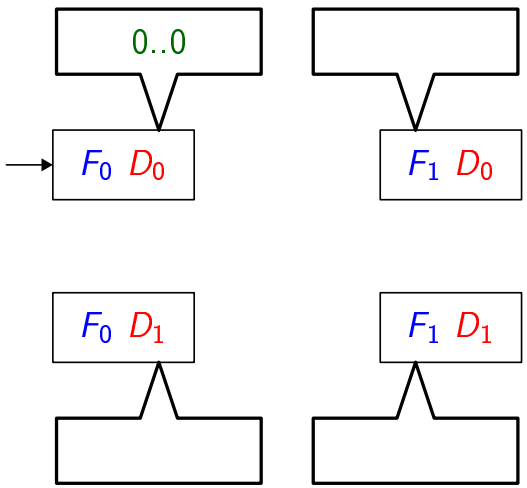
Analyse du système :

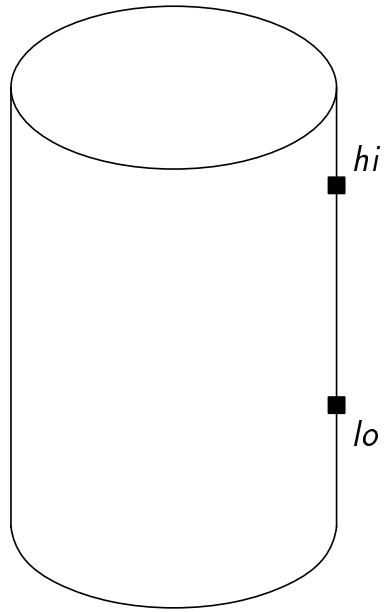
- Décoration : configuration $\longrightarrow$ mémoire abstraite
- Transitions

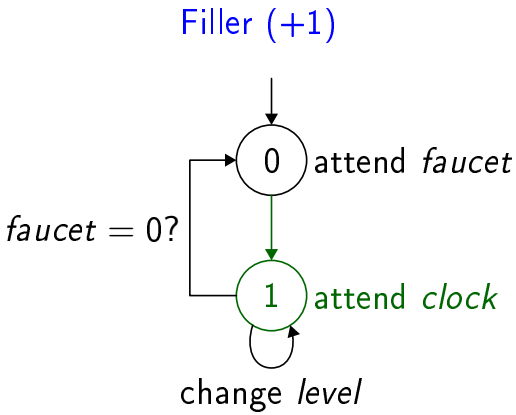


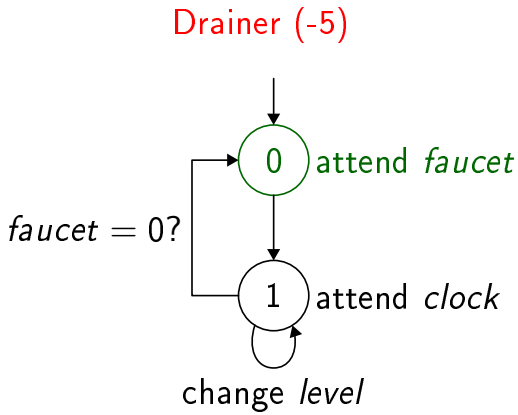

Filler (+1)
0
attend faucet
faucet = 0?
1
attend clock
change level

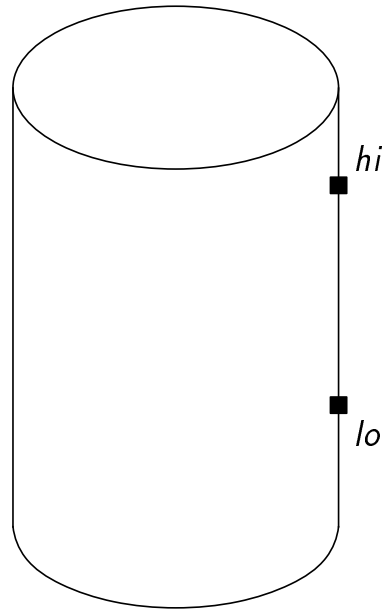

Drainer (-5)
0
attend faucet
faucet = 0?
1
attend clock
change level

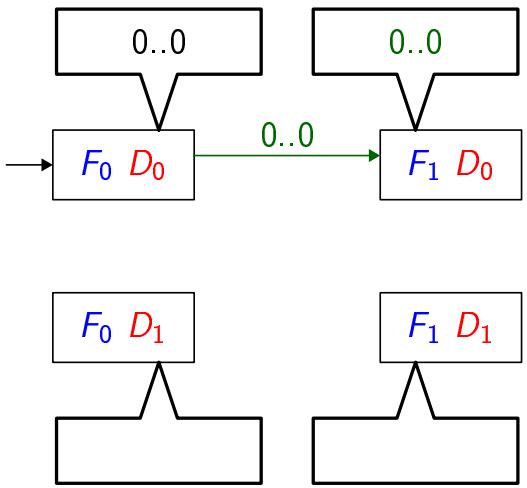

0..0
F0 D0
F1 D0
F0 D1
F1 D1

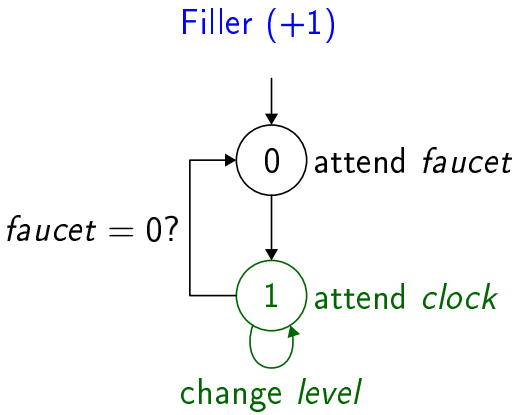

hi
lo

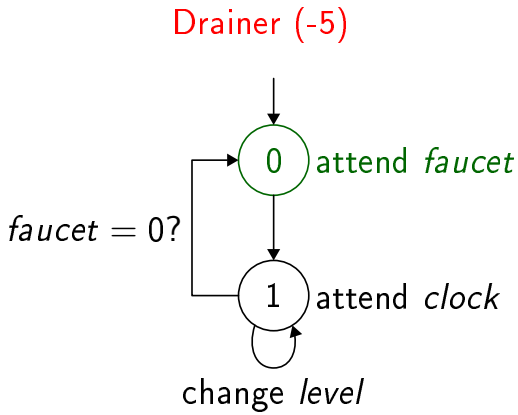

Filler (+1)
0
attend faucet
faucet = 0?
1
attend clock
change level

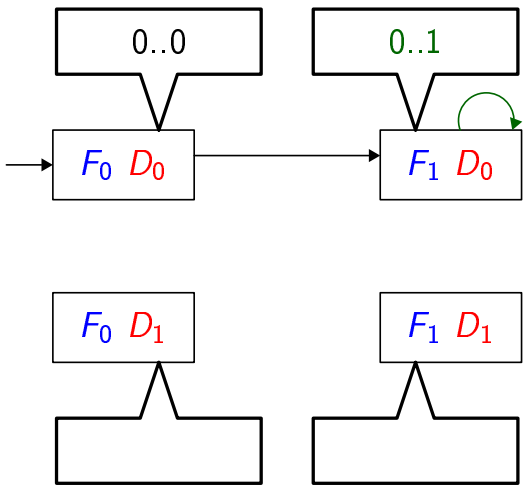

Drainer (-5)
0
attend faucet
faucet = 0?
1
attend clock
change level

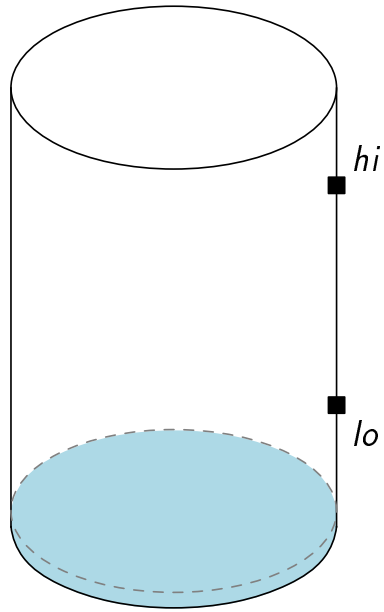

hi
lo

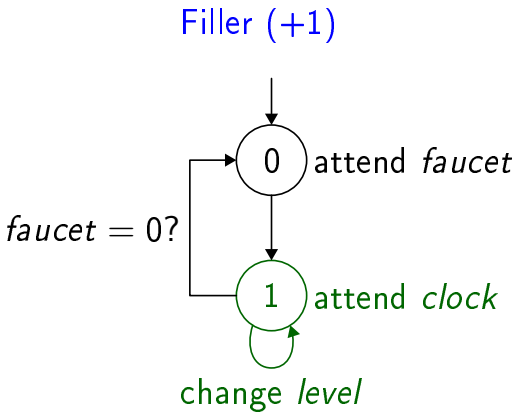

0..0
0..0
0..0
F_0 D_0
F_1 D_0
F_0 D_1
F_1 D_1

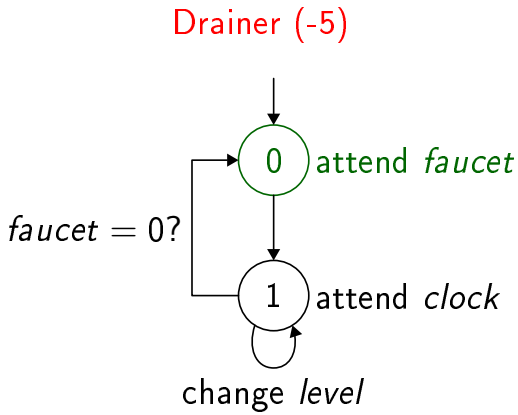

Filler (+1)
0
attend faucet
faucet = 0?
1
attend clock
change level

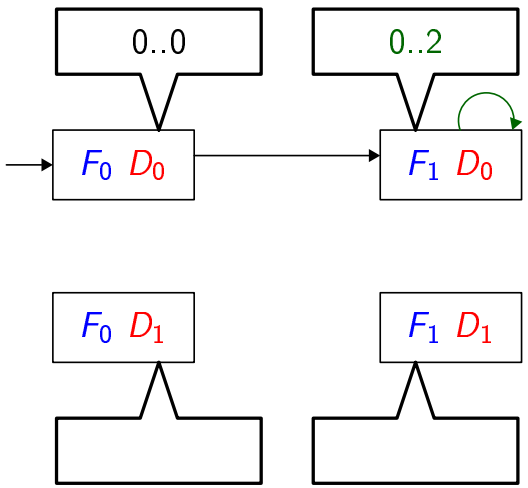

Drainer (-5)
0
attend faucet
faucet = 0?
1
attend clock
change level

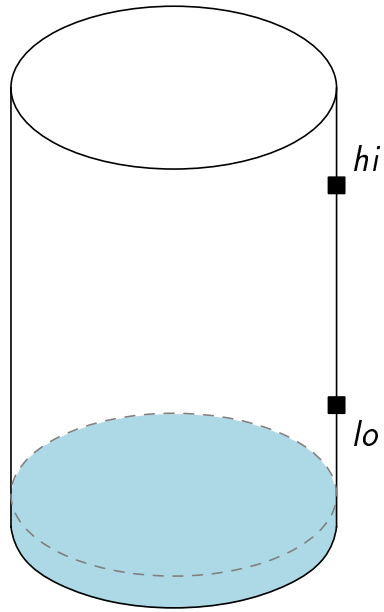

0..0
0..1
F_0 D_0
F_1 D_0
F_0 D_1
F_1 D_1

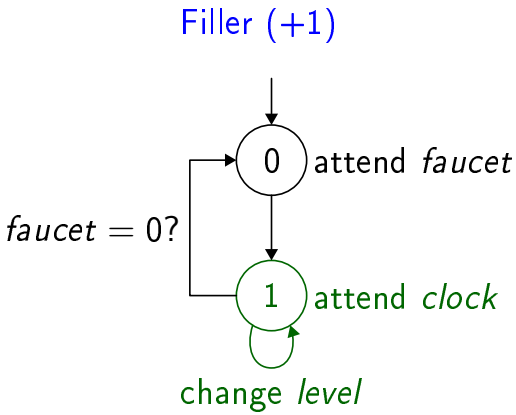

hi
lo

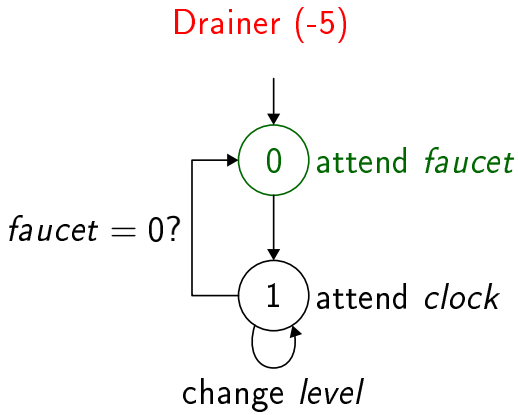

Filler (+1)
0
attend faucet
faucet = 0?
1
attend clock
change level

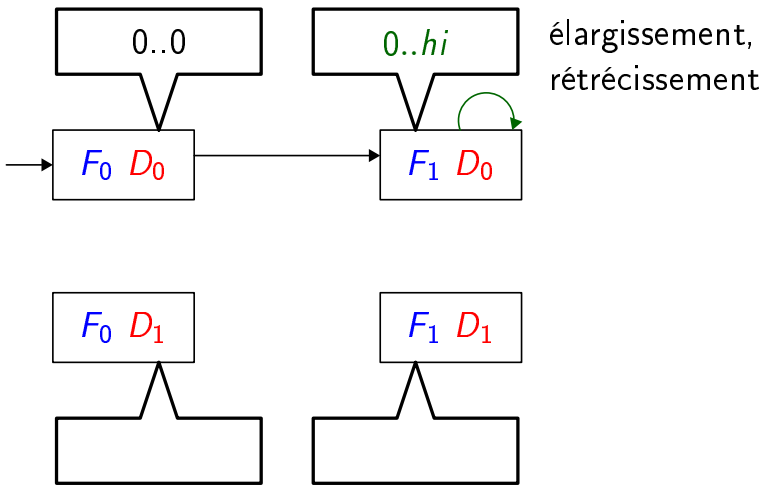

Drainer (-5)
0
attend faucet
faucet = 0?
1
attend clock
change level

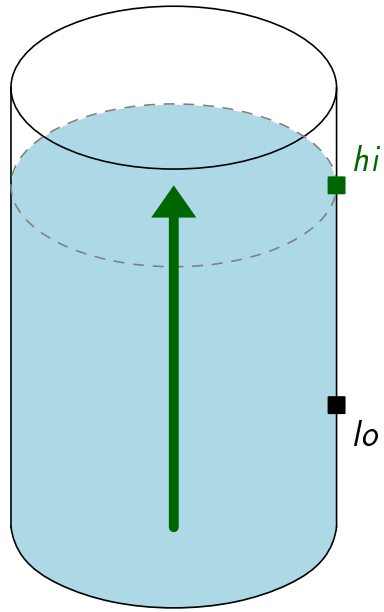

0..0
0..2
F_0 D_0
F_1 D_0
F_0 D_1
F_1 D_1

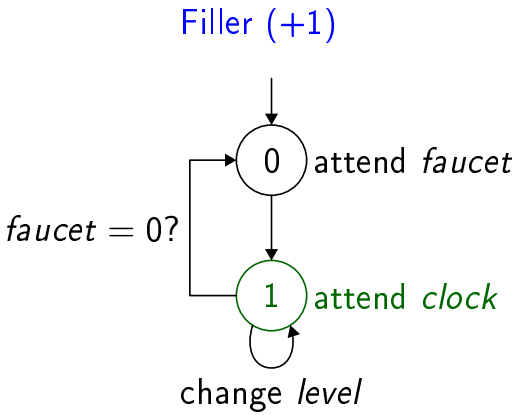

hi
lo

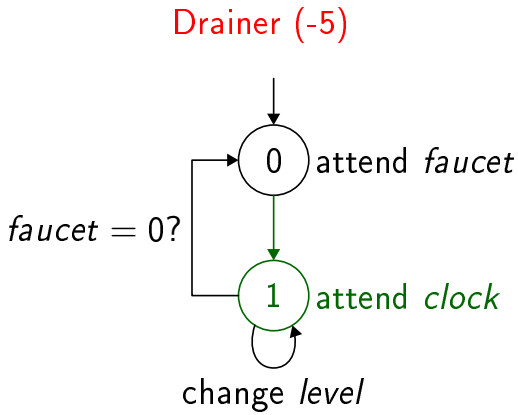

Filler (+1)
0
attend faucet
faucet = 0?
1
attend clock
change level

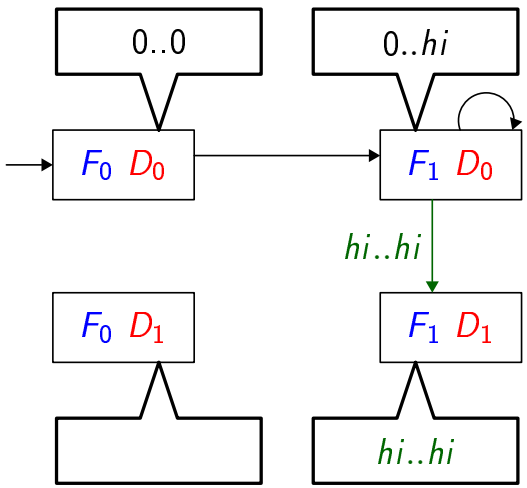

Drainer (-5)
0
attend faucet
faucet = 0?
1
attend clock
change level

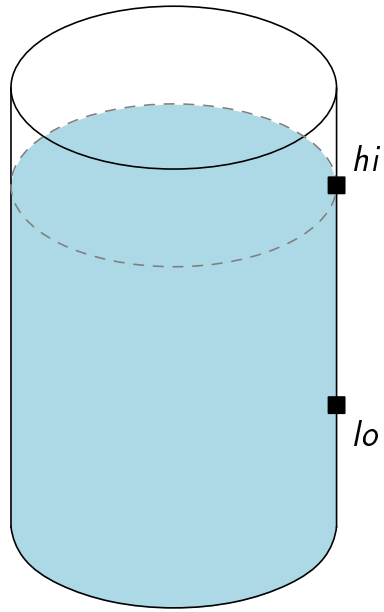

0..0
0..hi
élargissement,
rétrécissement
F0 D0
F1 D0
F0 D1
F1 D1

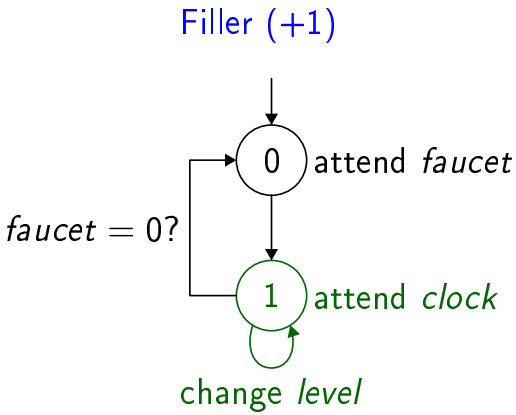

hi
lo

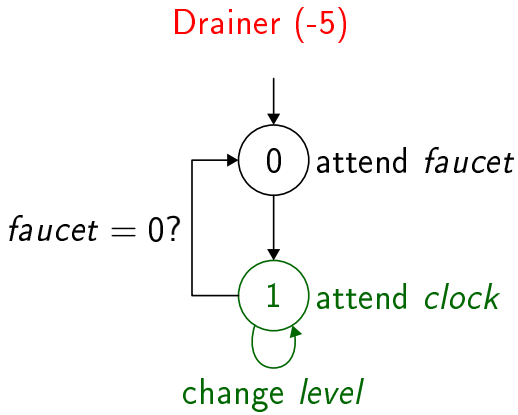

Filler (+1)
0
attend faucet
faucet = 0?
1
attend clock
change level

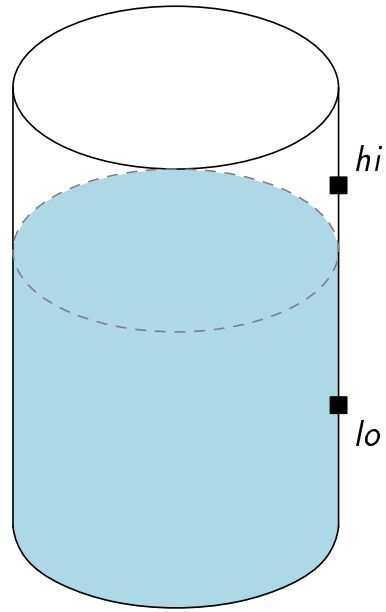

Drainer (-5)
0
attend faucet
faucet = 0?
1
attend clock
change level

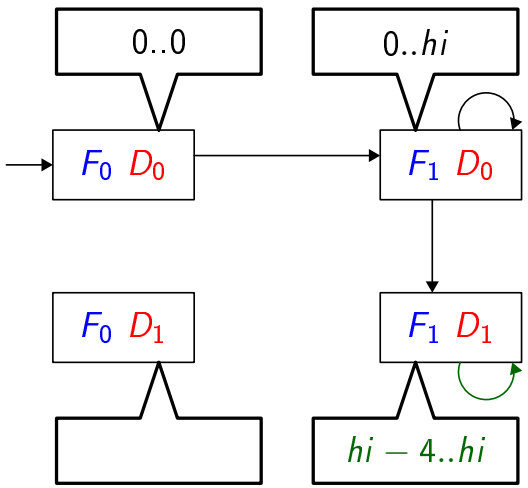

0..0
0..hi
F0 D0
F1 D0
hi..hi
F0 D1
F1 D1
hi..hi

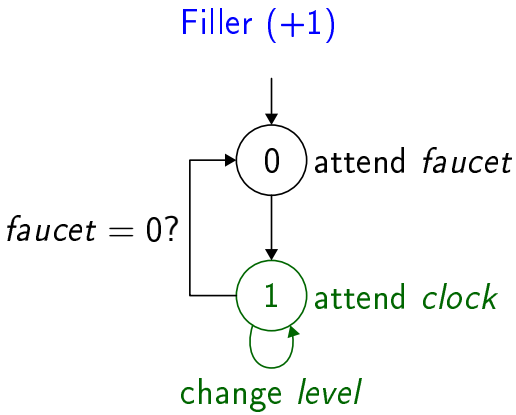

hi
lo

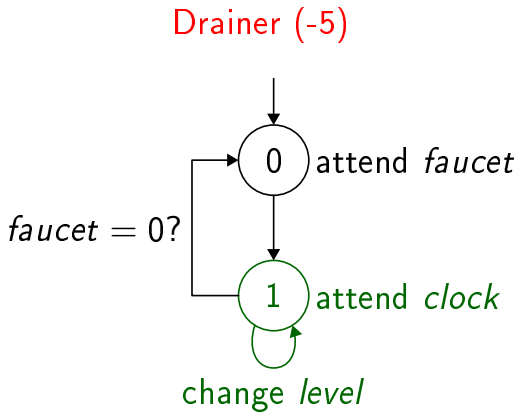

Filler (+1)
0
attend faucet
faucet = 0?
1
attend clock
change level

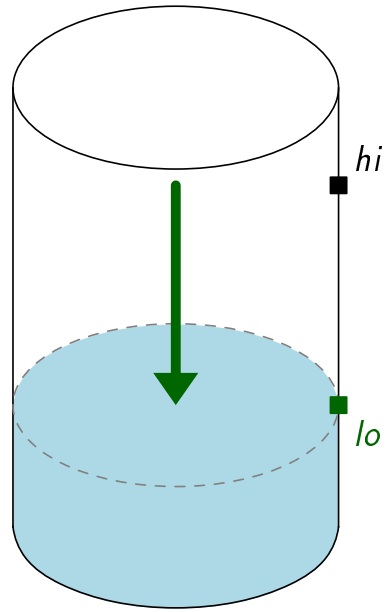

Drainer (-5)
0
attend faucet
faucet = 0?
1
attend clock
change level

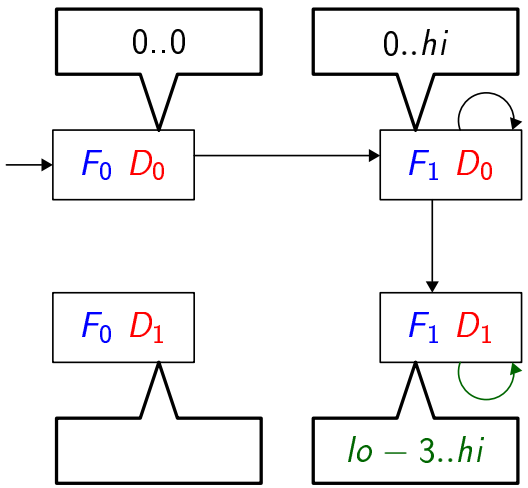

hi
lo


0..0
0..hi
F0 D0
F1 D0
F0 D1
F1 D1
hi − 4..hi

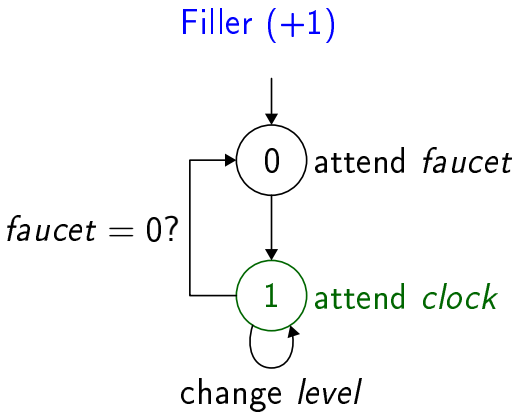


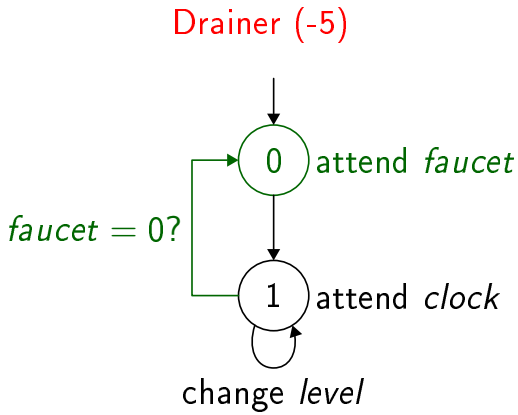


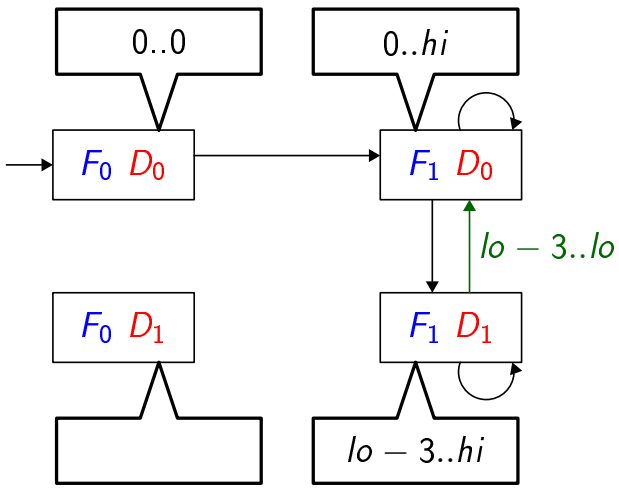


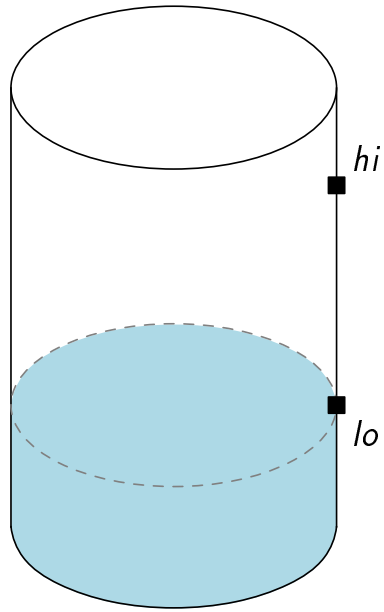


élargissement,
rétrécissement

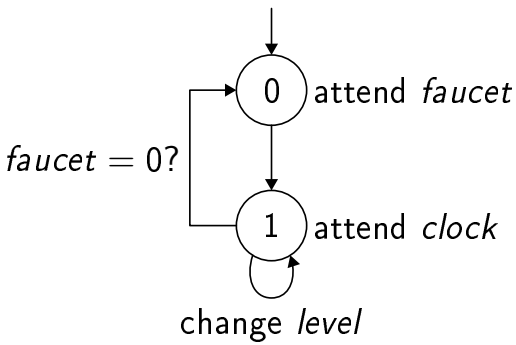

Filler (+1)
0
attend faucet
faucet = 0?
1
attend clock
change level


Drainer (-5)
0
attend faucet
faucet = 0?
1
attend clock
change level

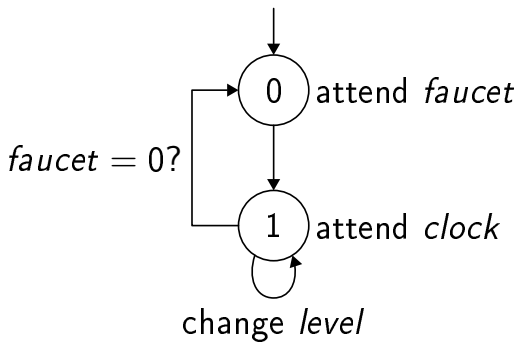

0..0
0..hi
F0 D0
F1 D0
lo − 3..lo
F0 D1
F1 D1
lo − 3..hi

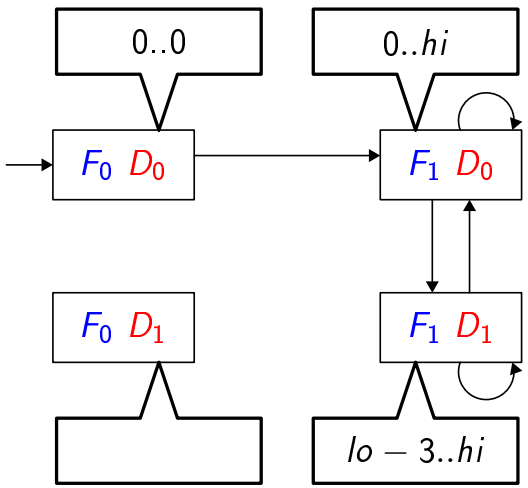

hi
lo

Filler (+1)

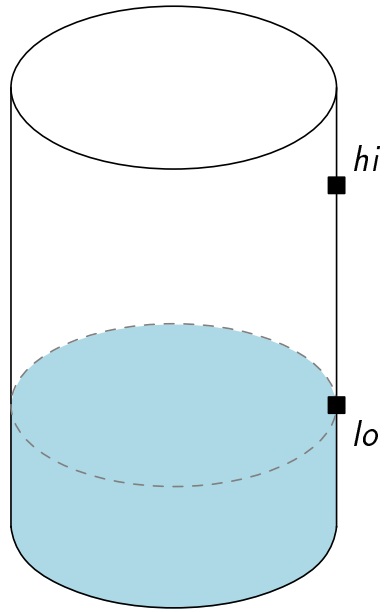


Drainer (-5)

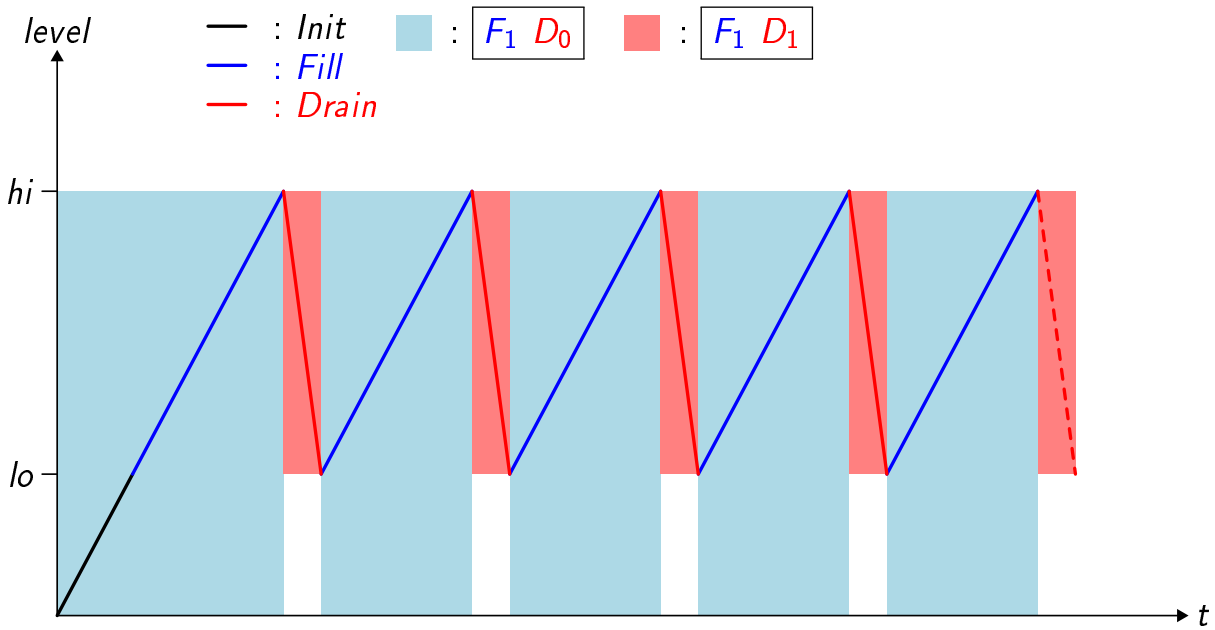




Point Fixe



Résultats



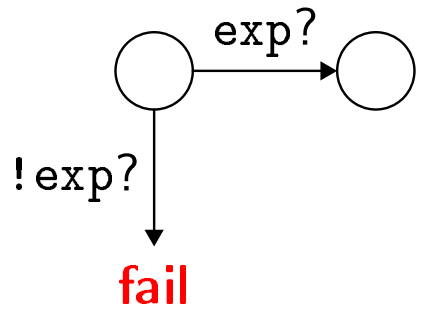
$level\ (\pm\delta) \in [0\ ;\ hi]$ ✓

Après initialisation, $level\ (\pm\delta) \in [lo\ ;\ hi]$ **X**

Plan

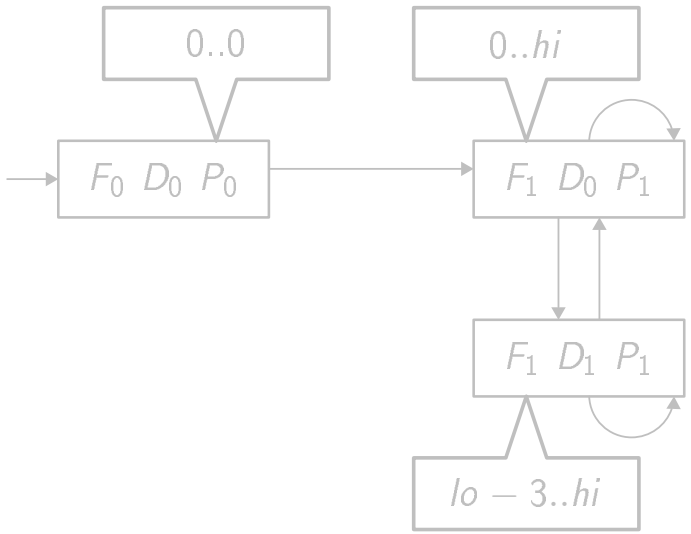
<< ***Une propriété est un système*** >>

`assert(exp) ;` —*Compilation*→

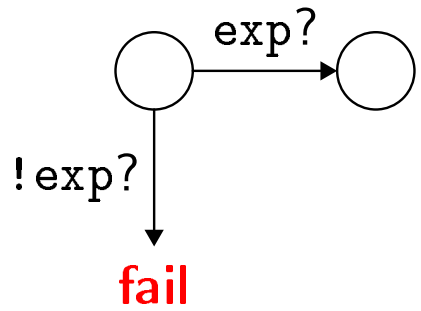


Vérification = atteignabilité de **fail**

```
property { (P0)
 while (true) {
  assert(  0 <= level &&
           level <= hi ) ; (fail)
  wait(clock) ; (P1) } }
```

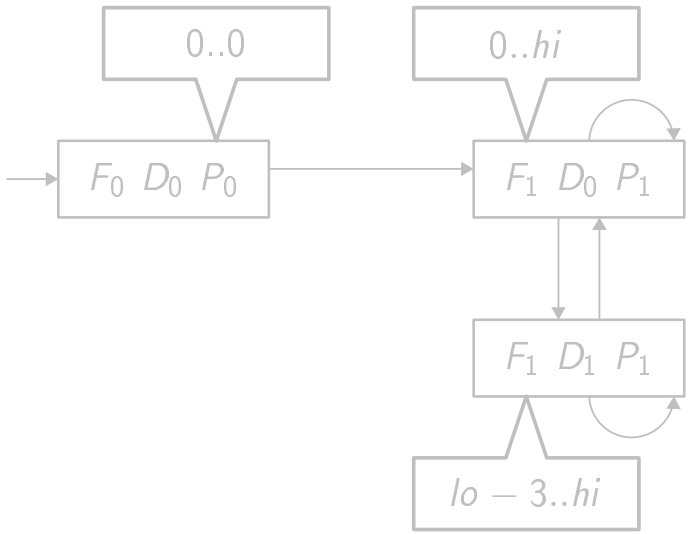


`assert(exp) ;` —*Compilation*→



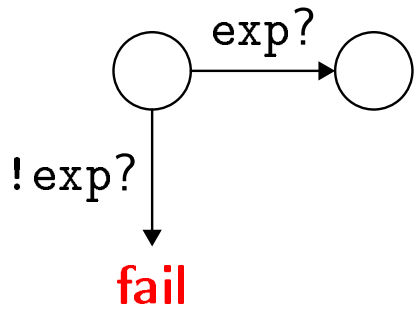
Vérification = atteignabilité de **fail**

```
property { (P0)
 while (true) {
  assert(  0 <= level &&
           level <= hi ) ; (fail)
  wait(clock) ; (P1) } }
```



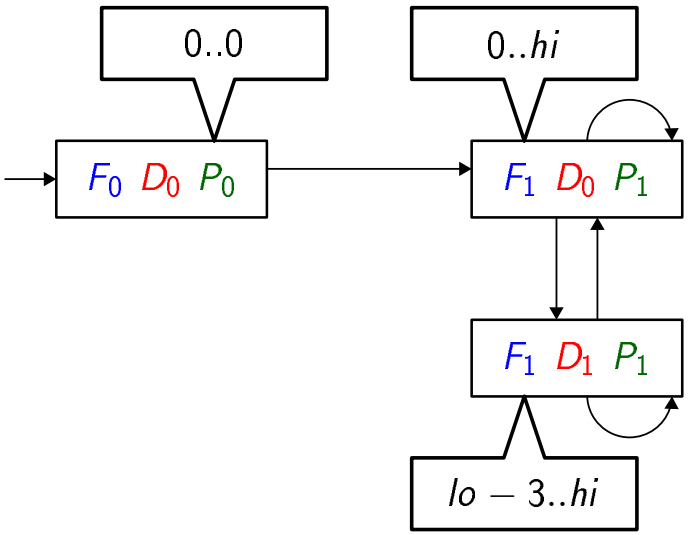
```
assert(exp) ;
```

Compilation →



Vérification = atteignabilité de **fail**

```
property { (P0)
 while (true) {
  assert(  0 <= level &&
           level <= hi ) ; (fail)
  wait(clock) ; (P1) } }
```



Définition : Trace et Sémantique

$$\mathcal{T} \triangleq \mathbb{N} \to \Sigma \qquad [\![S]\!] \in \mathcal{P}(\mathcal{T})$$

Définition : Sûreté locale

$$\mathrm{Safe}_S(t) \quad \triangleq \quad \forall p \in S,\ \forall i,\ t_i(p) \neq \mathit{fail}$$

Définition : Validité

$$S \models S' \quad \triangleq \quad \forall t \in [\![S \otimes S']\!],\ \mathrm{Safe}_S(t) \Rightarrow \mathrm{Safe}_{S'}(t)$$

Proposition

$$S \models S' \quad \Leftrightarrow \quad S \otimes S' \models_{\mathrm{MC}} \mathbf{A}\ (\mathbf{G}\ \mathrm{Safe}_S \Rightarrow \mathbf{G}\ \mathrm{Safe}_{S'})$$

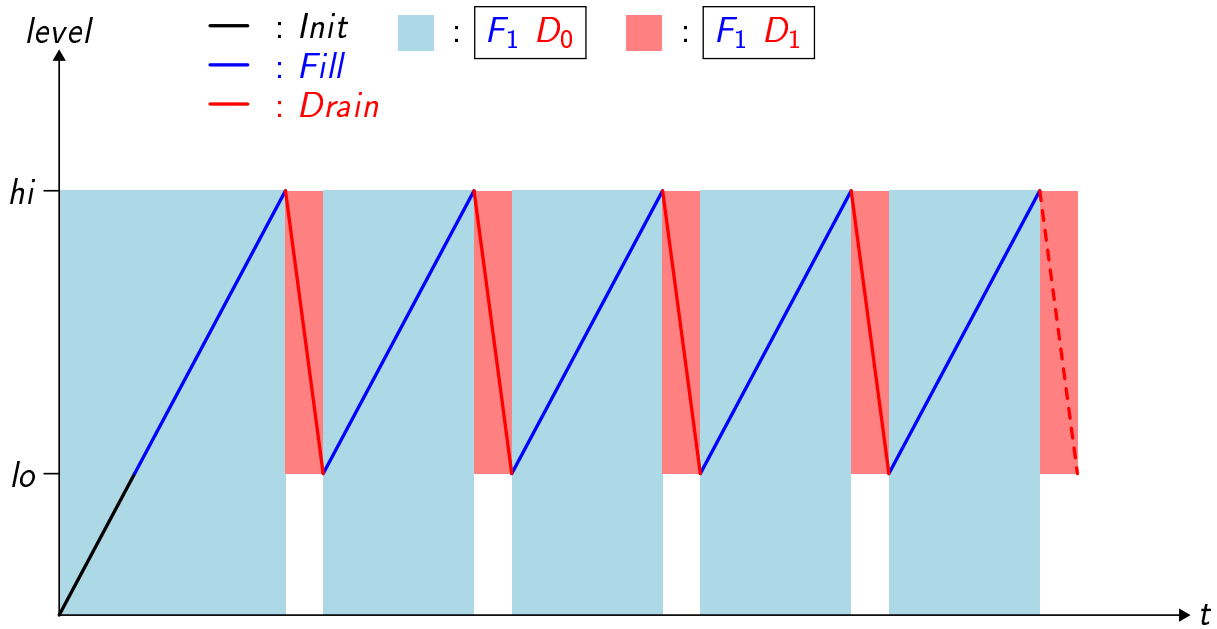
Propriété d'état : `assert(x < 10) ;`

Hypothèse : `if (exp) {wait(x) ; assert(x < 10) ;}`

Répétition : `while (exp) {wait(x) ; assert(x < 10) ;}`

Ensemble de valeurs : `x = rand(min, max) ;`

Conjonction $P_1 \wedge P_2$ **:** `if (rand()) {`P_1`} else {`P_2`}`



$level\ (\pm\delta) \in [0\ ;\ hi]$ ✓

Après initialisation, $level\ (\pm\delta) \in [lo\ ;\ hi]$ **X**

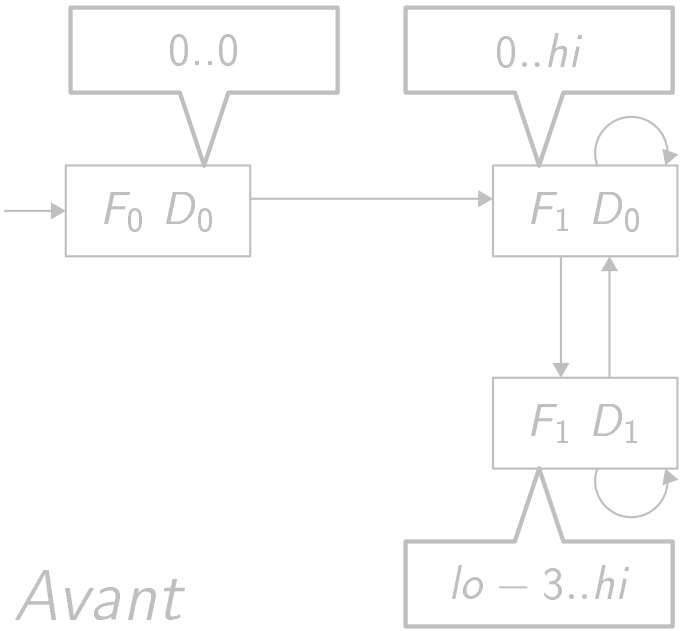
Phases *Init* et *Fill* fusionnées

Plan

<< ***Ajouter des processus pour gagner en précision*** >>

Plus de Configurations ⇒ Plus de Précision

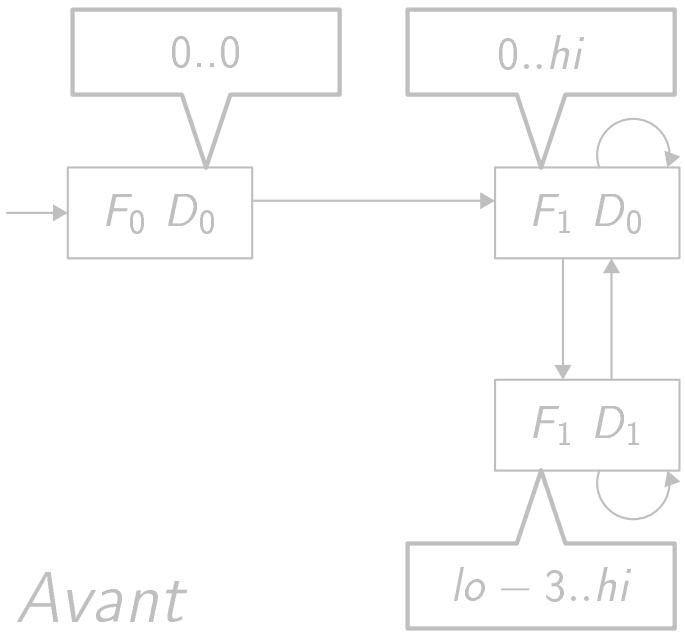
```
discriminate {
  wait(lo) ;   (d0)
  halt() ;     (d1) }
```



Avant

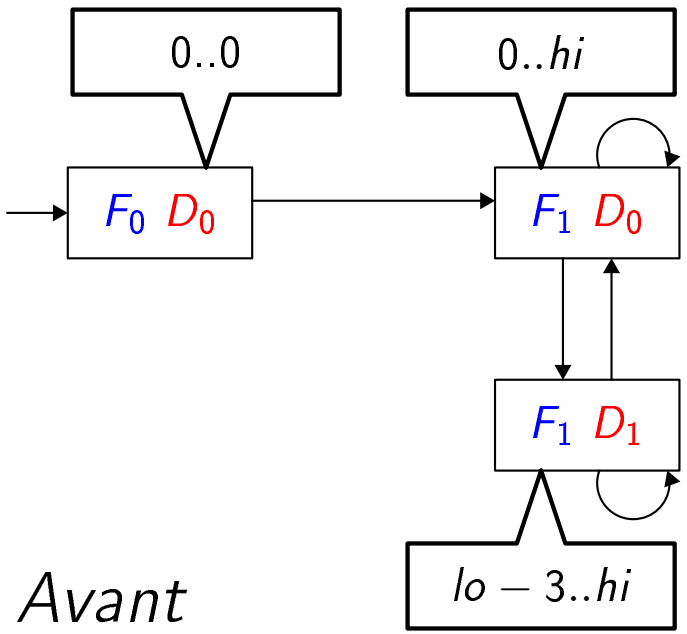
Plus de Configurations ⇒ Plus de Précision

```
discriminate {
  wait(lo) ;   (d0)
  halt() ;     (d1) }
```



Plus de Configurations $\Rightarrow$ Plus de Précision

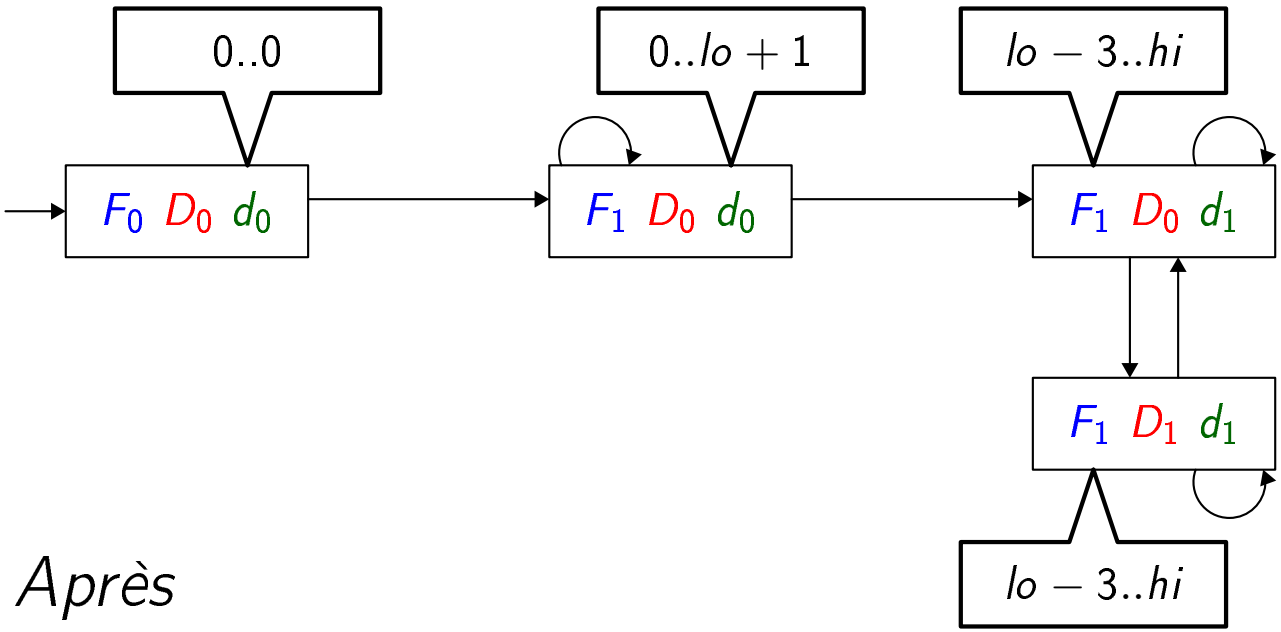
```
discriminate {
  wait(lo) ;   (d0)
  halt() ;     (d1) }
```



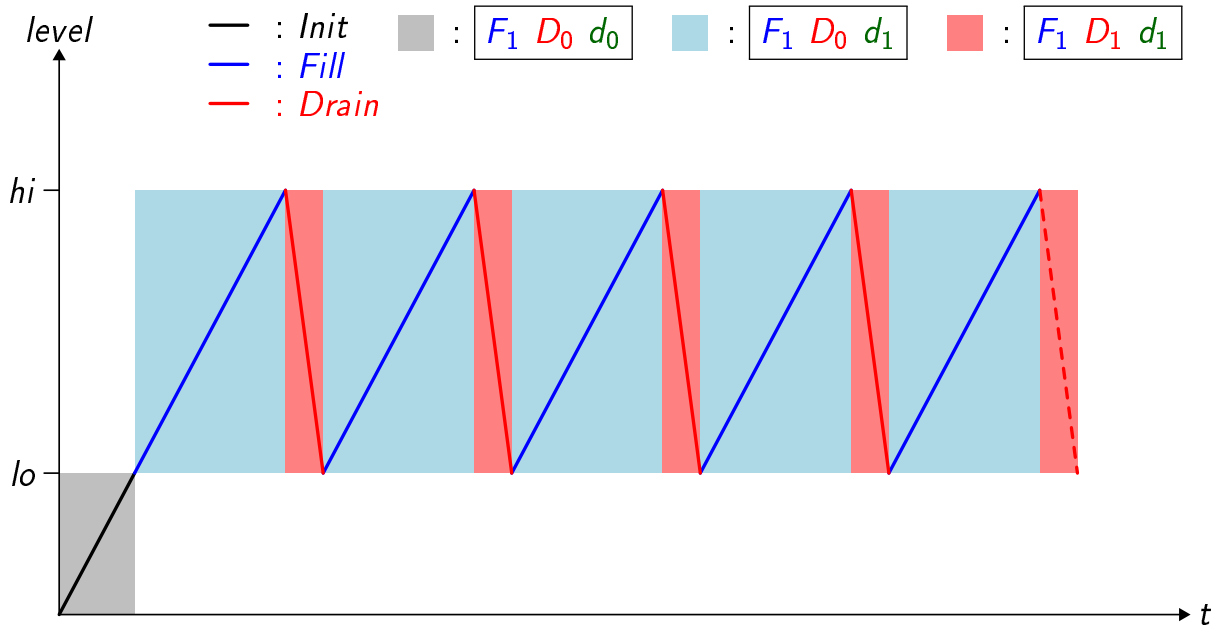
Avant

Plus de Configurations $\Rightarrow$ Plus de Précision

```
discriminate {
  wait(lo) ;   (d0)
  halt() ;     (d1) }
```



Après



$level\ (\pm\delta) \in [0\ ;\ hi]$ $\surd$

Après initialisation, $level\ (\pm\delta) \in [lo\ ;\ hi]$ $\surd$

Article VECoS [ACV08b]

Définition : **Système Discriminant**

$$S_{\mathcal{D}} \text{ discrimine } S \quad \triangleq \quad \begin{cases} S_{\mathcal{D}} \text{ n'écrit pas les variables de } S \\ S_{\mathcal{D}} \text{ ne bloque pas } S \end{cases}$$

Théorème : **Discriminant et Validité**

$$\frac{S \otimes S_{\mathcal{D}} \models S' \qquad S_{\mathcal{D}} \text{ discrimine } S \otimes S'}{S \models S'}$$

- $S_{\mathcal{D}}$ **n'écrit pas les variables de** S**.**
  Syntaxique.
- $S_{\mathcal{D}}$ **ne bloque pas** S**.**
  Critère plus général : *si S attend un temps non nul, $S_{\mathcal{D}}$ ne boucle pas avec des attentes sans temps et sur ses locaux.*

Définition : **Système Discriminant**

$$S_{\mathcal{D}} \text{ discrimine } S \quad \triangleq \quad \begin{cases} S_{\mathcal{D}} \text{ n'écrit pas les variables de } S \\ S_{\mathcal{D}} \text{ ne bloque pas } S \end{cases}$$

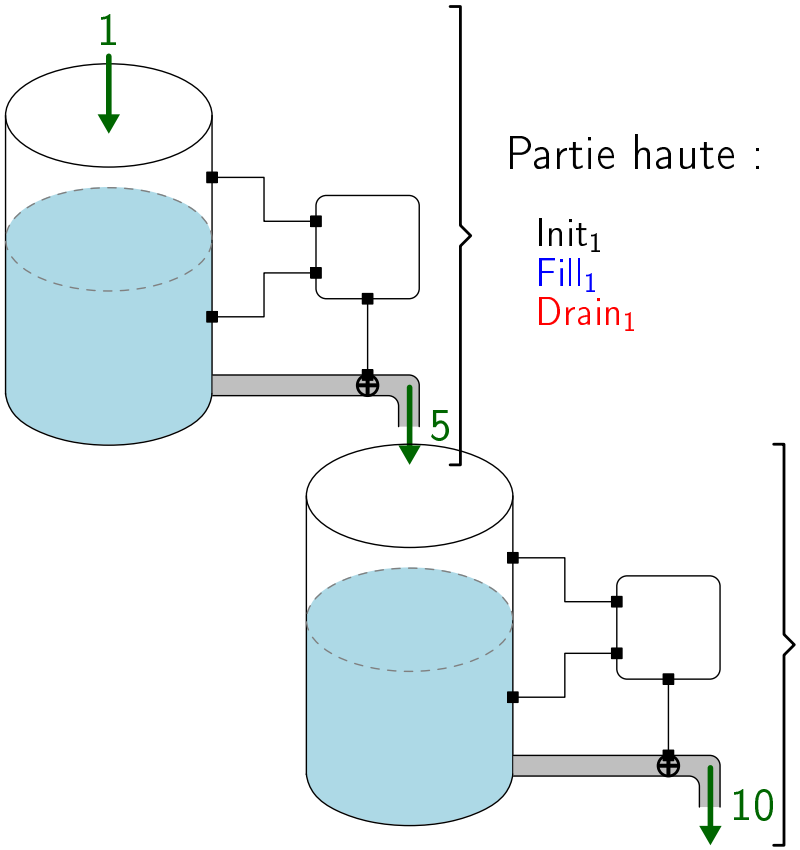
Théorème : **Discriminant et Validité**

$$\frac{S \otimes S_{\mathcal{D}} \models S' \qquad S_{\mathcal{D}} \text{ discrimine } S \otimes S'}{S \models S'}$$

- $S_{\mathcal{D}}$ **n'écrit pas les variables de** S**.**
  Syntaxique.
- $S_{\mathcal{D}}$ **ne bloque pas** S**.**
  Critère plus général : *si* S *attend un temps non nul,* $S_{\mathcal{D}}$ *ne boucle pas avec des attentes sans temps et sur ses locaux.*

Plan

<< ***Remplacer un système par un système plus simple*** >>



Partie haute :

Init_1
Fill_1
Drain_1

Partie basse :

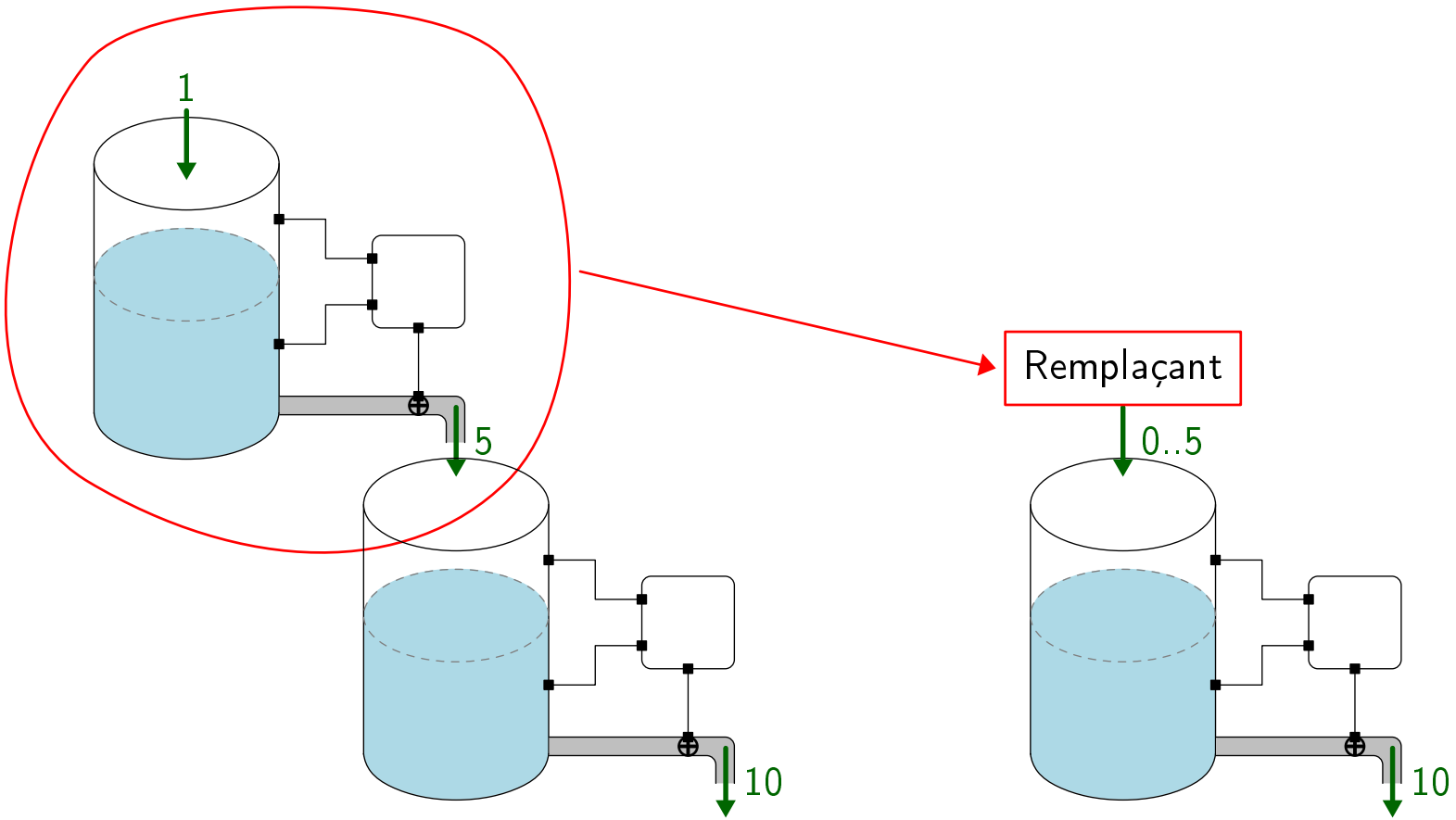
Init_2
Fill_2
Drain_2

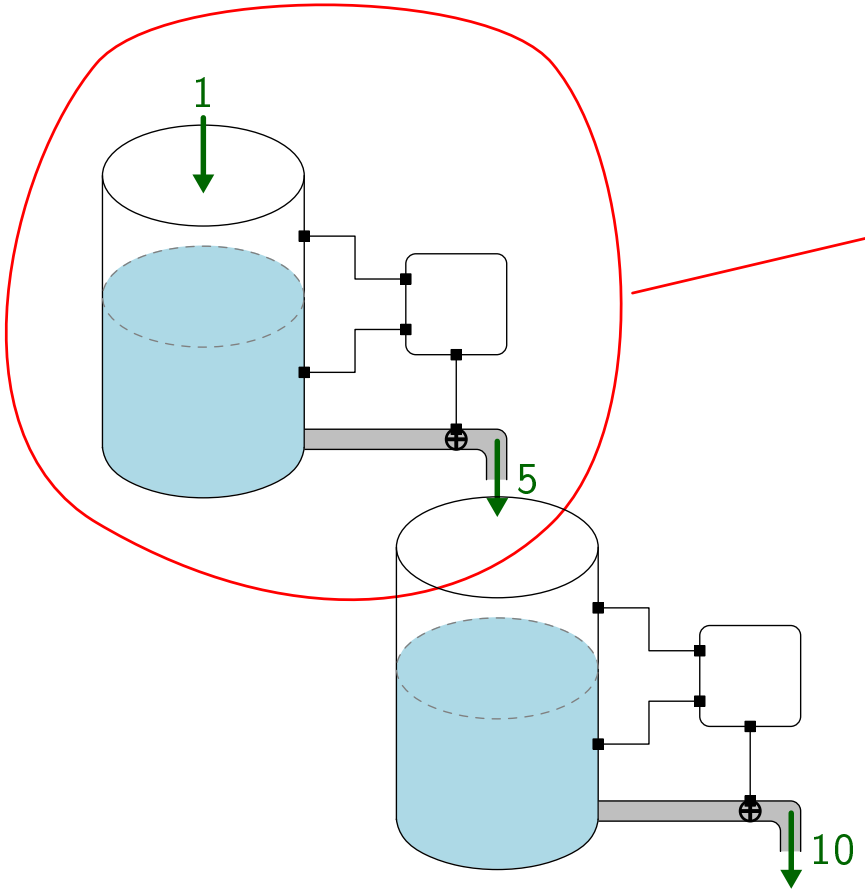
Système complet :

Init_1	Init_2
Init_1	Fill_2
Init_1	Drain_2
Fill_1	Init_2
Fill_1	Fill_2
Fill_1	Drain_2
Drain_1	Init_2
Drain_1	Fill_2
Drain_1	Drain_2

Assemblage de systèmes $\Rightarrow$ Produit du nombre de configurations

Compositionnalité : remplacer un système par un autre






```
abstraction PumpAbs(PumpSystem P) {

 port<int> out_drain;

 local : out_drain;
 link : out_drain = P.pump_out.drain;

 thread {
  while (true) {
  out_drain = rand(0, 5);
  wait(clock); } }
}
```

```
module Sys_Prop {

 PumpSerie PS with PumpAbs(PS.top);

 property {
   ... }
}

main_analysis { Sys_Prop P; }
```

`abstraction` : définit un système remplaçant

`link` : variables identifiées

Définition : Remplacement de S par S' (variables E exclues)

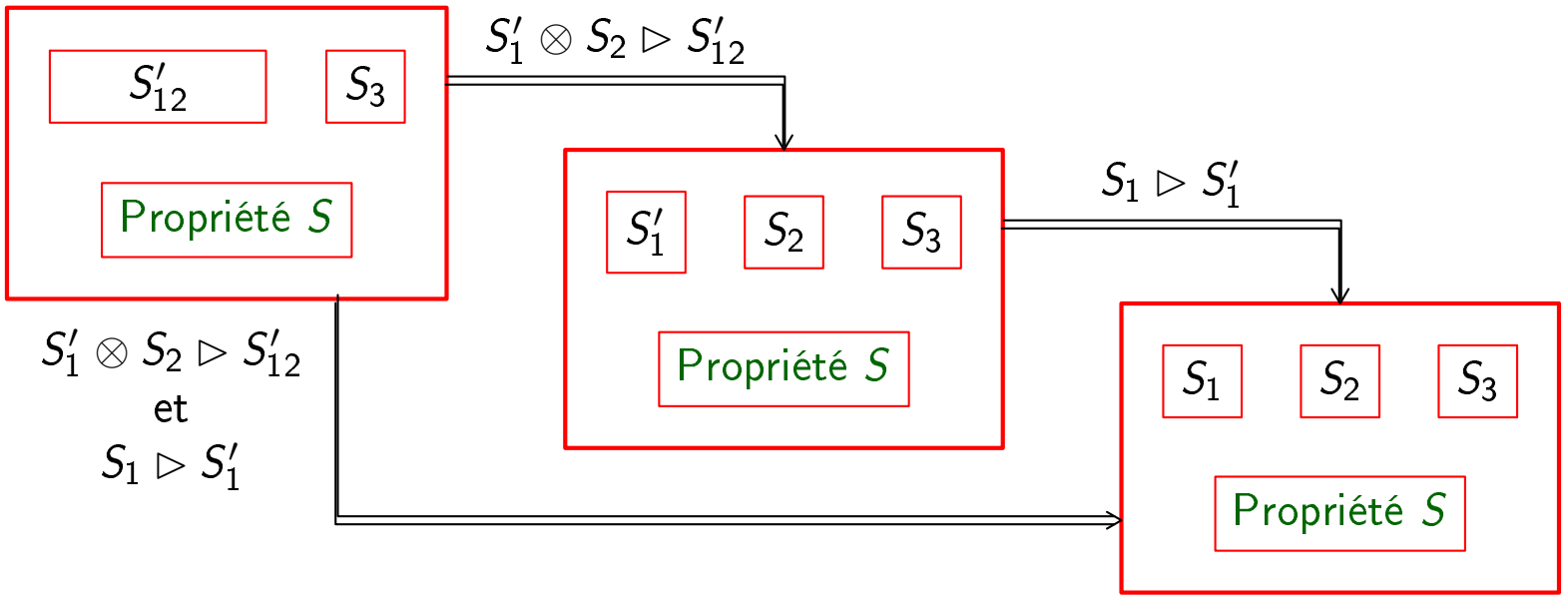
$\forall t \in [\![S]\!],\ \exists t' \in [\![S']\!]$ telle que t' s'obtient à partir de t en insérant des états où :

- $\forall x \notin E$, la valeur de x est préservée ;
- $\forall p \notin S'$ tel que p ne lit pas des évènements de E, le statut d'attente de p est le même ;
- $\text{Safe}_S(t) \Rightarrow \text{Safe}_{S'}(t')$

Notation $S \rhd_E S'$: remplacement de S par S'

Théorème : Remplacement et Validité

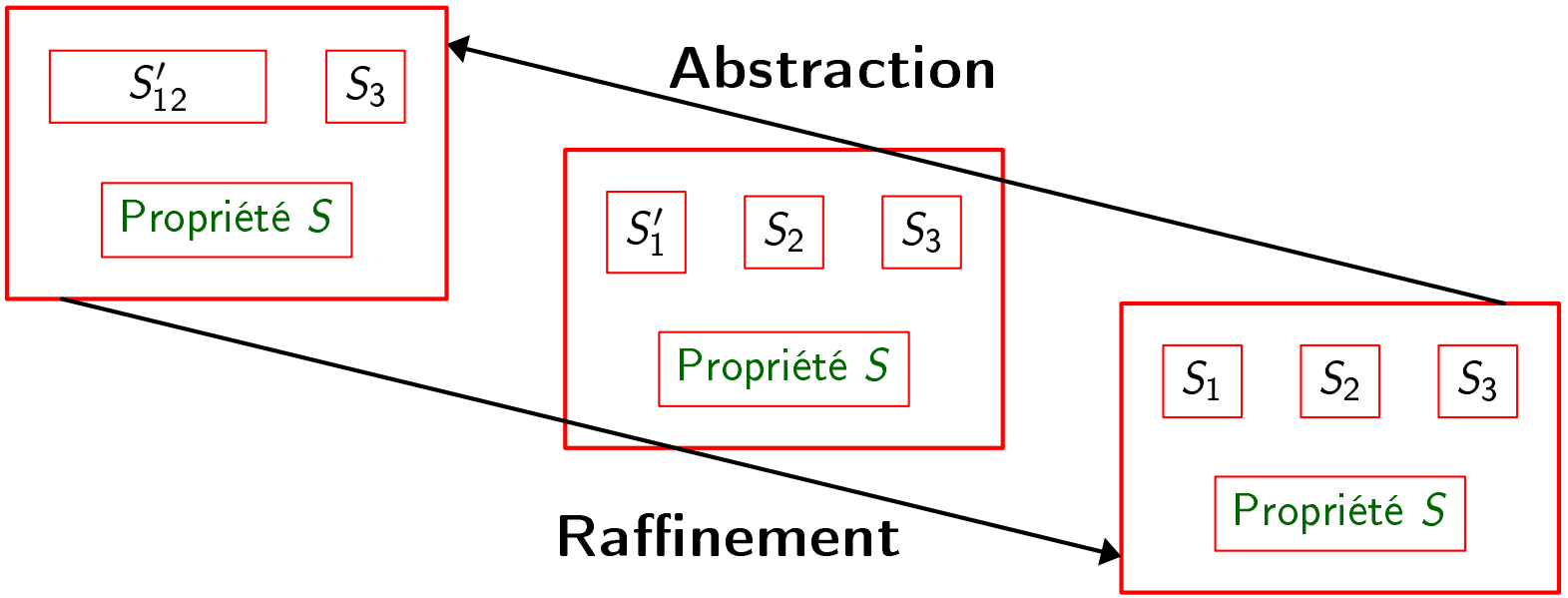
$$\frac{S_1 \rhd_E S_1' \qquad S_1' \otimes S_2 \models S}{S_1 \otimes S_2 \models S} \text{ Si } Read(S_2) \cap E = Read(S) \cap E = \emptyset$$



Article DATICS [ACV08a]

Théorème : Remplacement et Validité

$$\frac{S_1 \rhd_E S_1' \qquad S_1' \otimes S_2 \models S}{S_1 \otimes S_2 \models S} \text{ Si } Read(S_2) \cap E = Read(S) \cap E = \emptyset$$



Article DATICS [ACV08a]

`abstraction A(System1 S1) { System2 S2; ... }`

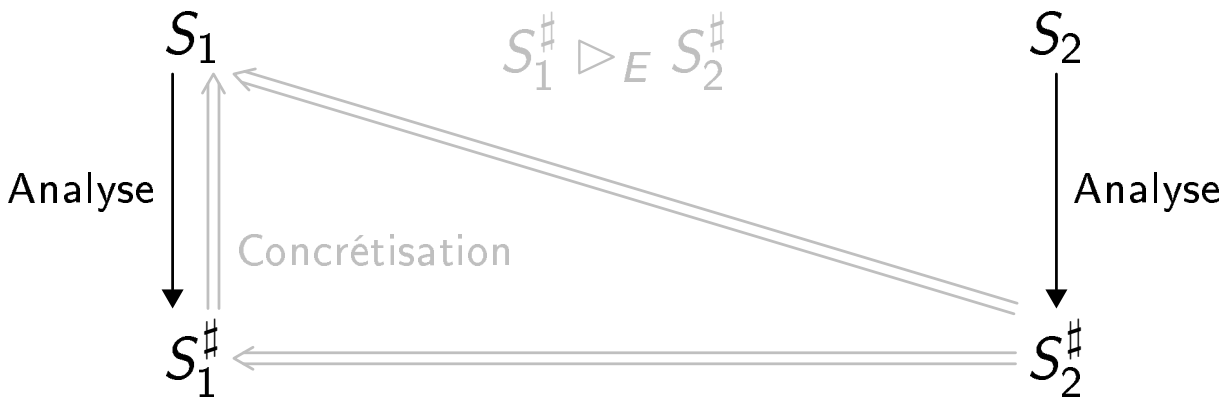
=

`S1` remplacé par abstraction de `S2`

Théorème : Validité par Remplacement et Abstraction

$$\frac{S_1^\sharp \rhd_E S_2^\sharp \qquad S_2^\sharp \models S}{S_1 \models S} \text{ Si } Read(S) \cap E = \emptyset$$

où $S_1^\sharp$ = abstraction de S_1, $S_2^\sharp$ = abstraction de S_2



`abstraction A(System1 S1) { System2 S2; ... }`

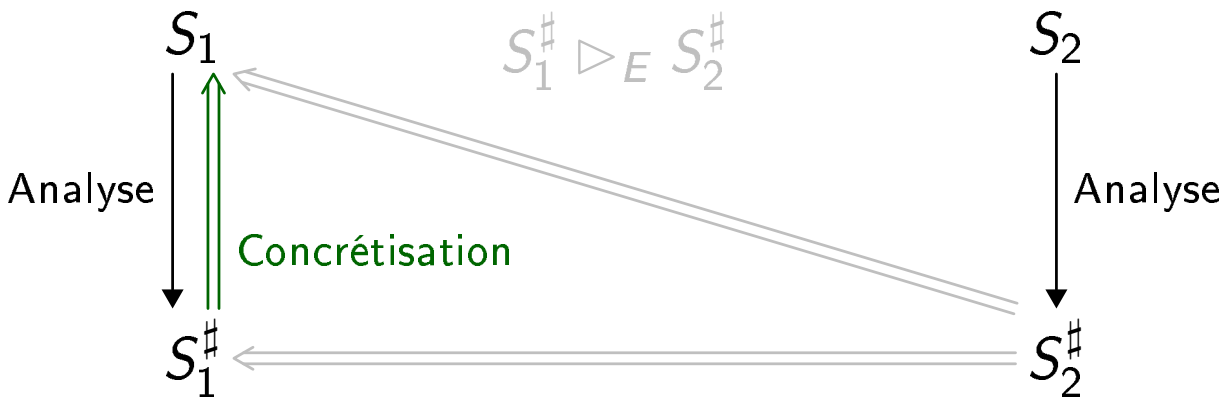
=

`S1` remplacé par abstraction de `S2`

Théorème : Validité par Remplacement et Abstraction

$$\frac{S_1^\sharp \rhd_E S_2^\sharp \qquad S_2^\sharp \models S}{S_1 \models S} \text{ Si } Read(S) \cap E = \emptyset$$

où $S_1^\sharp$ = abstraction de S_1, $S_2^\sharp$ = abstraction de S_2



`abstraction A(System1 S1) { System2 S2; ... }`

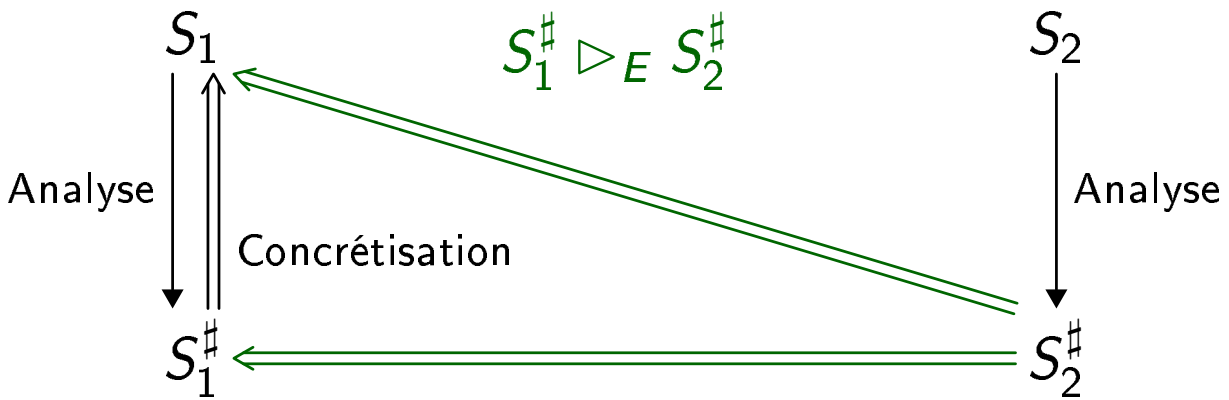
=

S1 remplacé par abstraction de S2

Théorème : Validité par Remplacement et Abstraction

$$\frac{S_1^\sharp \rhd_E S_2^\sharp \qquad S_2^\sharp \models S}{S_1 \models S} \text{ Si } Read(S) \cap E = \emptyset$$

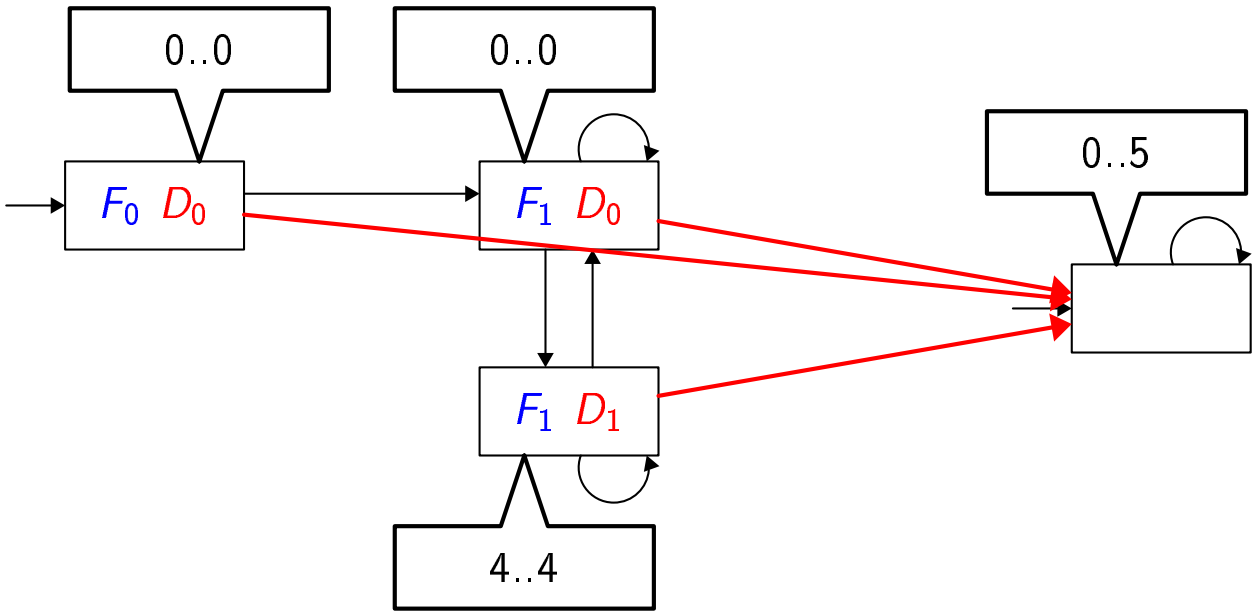
où $S_1^\sharp$ = abstraction de S_1, $S_2^\sharp$ = abstraction de S_2



$S_1^\sharp \rhd_E S_2^\sharp$: **Critère de remplacement**

Projection des configurations de $S_1^\sharp$ vers celles de $S_2^\sharp$ où :

- pour chaque variable, plus de valeur en $S_2^\sharp$;
- sûreté locale en $S_1^\sharp \Rightarrow$ sûreté locale en $S_2^\sharp$;
- transitions préservées.



Conclusion

SystemD :

- Description, Spécification et Vérification compositionnelle
- Accessible à l'Ingénieur
- Vérification statique et automatique

Implantation :

- KernelD + moteur de simulation (OCaml)
- Preuve formelle de quelques thérorèmes (Coq)
  Ex : $[\![S_1 \otimes S_2]\!] = [\![S_1]\!] \cap [\![S_2]\!]$

Conclusion

SystemD :

- Description, Spécification et Vérification compositionnelle
- Accessible à l'Ingénieur
- Vérification statique et automatique

Implantation :

- KernelD + moteur de simulation (OCaml)
- Preuve formelle de quelques thérorèmes (Coq)
  Ex : $[\![S_1 \otimes S_2]\!] = [\![S_1]\!] \cap [\![S_2]\!]$

Conclusion

Perspectives :

- Exemple “réel” de grande taille
- Réduction par Remplacement
- Remplacement Automatique
- Génie Logiciel

Merci pour votre attention

Bibliographie


Nicolas Ayache, Loïc Correnson, and Franck Védrine.
Scenarios for validating SystemC descriptions.
In *ICC'08 : Proceedings of the 12th WSEAS international conference on Circuits*, pages 468–473, Stevens Point, Wisconsin, USA, 2008. World Scientific and Engineering Academy and Society (WSEAS).


Nicolas Ayache, Loïc Correnson, and Franck Védrine.
Verifying SystemC with Scenario.
In *2nd International Workshop on Verification and Evaluation of Computer and Communication Systems (VECoS 2008)*, eWiC, Leeds, UK, 2008. British Computer Society.