Heptagon/BZR : compilation et synthèse de contrôleurs

Gwenaël Delaval

Université de Grenoble, Lig

Journées francophones de compilation — Annecy

Cycle de conception classique

Programmeur

Model-checking

Mon programme vérifie-t-il la propriété P ?

Non.

Mon programme (modifié) vérifie-t-il la propriété P ?

Non.

Mon programme (modifié) vérifie-t-il la propriété P ?

Non.

Cycle de conception avec BZR

Programmeur

Voici un programme non déterministe. Est-il possible de le contraindre afin qu'il vérifie la propriété P ?

BZR

Bien sûr : voici la contrainte.

Programmation synchrone en Heptagon/BZR

```
node alloc(r : bool) returns (g : bool)
let
  automaton
    state Idle
      do g = false until r then Alloc
    state Alloc
      do g = true until (not r) then Idle
  end
tel

node main(r0,r1 : bool) returns (g0,g1 : bool)
let
  g0 = inlined alloc(r0);
  g1 = inlined alloc(r1);
tel
```

```
node main(r0,r1 : bool) returns (g0,g1 : bool)



let
  g0 = inlined alloc(r0);
  g1 = inlined alloc(r1);
tel
```

```
node main(r0,r1 : bool) returns (g0,g1 : bool)
  contract
  assume not (r0 & r1)
  enforce not (g0 & g1)
let
  g0 = inlined alloc(r0);
  g1 = inlined alloc(r1);
tel
```

- mécanisme de contrats

```
node main(r0,r1 : bool) returns (g0,g1 : bool)
  contract
  assume not (r0 & r1)
  enforce not (g0 & g1) with (c0,c1:bool)
let
  g0 = inlined alloc(r0 & c0);
  g1 = inlined alloc(r1 & c1);
tel
```

- mécanisme de contrats
- non-déterminisme : variables contrôlables

```
node main(r0,r1 : bool) returns (g0,g1 : bool)


  var c0,c1:bool
let
  (c0,c1) = controller(r0,r1);
  g0 = inlined alloc(r0 & c0);
  g1 = inlined alloc(r1 & c1);
tel
```

- mécanisme de contrats
- non-déterminisme : variables contrôlables
- contrainte : contrôleur calculé par synthèse de contrôleurs

Plan

Synthèse de contrôleurs discrets : principe

But

Imposer une propriété temporelle Φ sur un système réactif (ne satisfaisant *a priori* pas Φ)

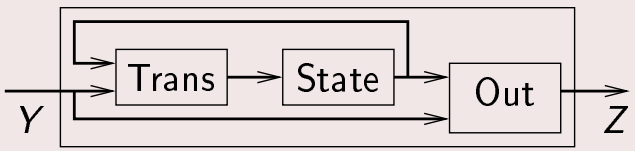
Synthèse de contrôleurs discrets : principe

But

Imposer une propriété temporelle Φ sur un système réactif (ne satisfaisant *a priori* pas Φ)

Principe (représentation équationnelle implicite)

State	mémoire
Trans	fonction de transition
Out	fonction de sortie



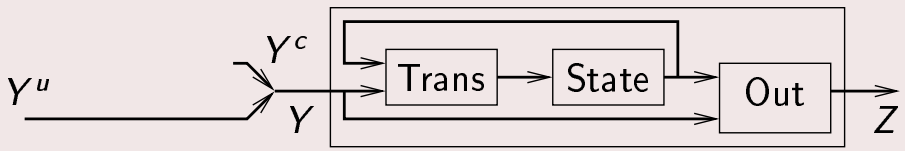
Synthèse de contrôleurs discrets : principe

But

Imposer une propriété temporelle Φ sur un système réactif (ne satisfaisant *a priori* pas Φ)

Principe (représentation équationnelle implicite)

State	mémoire
Trans	fonction de transition
Out	fonction de sortie



- Partition des entrées en contrôlables (Y^c) et incontrôlables (Y^u)

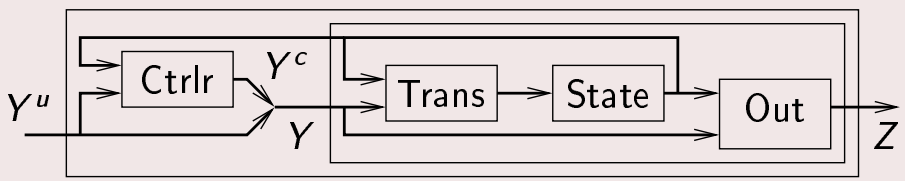
Synthèse de contrôleurs discrets : principe

But

Imposer une propriété temporelle Φ sur un système réactif (ne satisfaisant *a priori* pas Φ)

Principe (représentation équationnelle implicite)

State mémoire
Trans fonction de transition
Out fonction de sortie



- Partition des entrées en contrôlables (Y^c) et incontrôlables (Y^u)
- Calcul d'un contrôleur tel que le système contrôlé satisfait Φ

BZR : contrats et SdC

$f(x_1, \ldots, x_n) = (y_1, \ldots, y_p)$

assume e_A

enforce e_G

with $c_1, \ldots, c_q$

$y_1 = f_1(x_1, \ldots, x_n, c_1, \ldots, c_q)$

$\ldots$

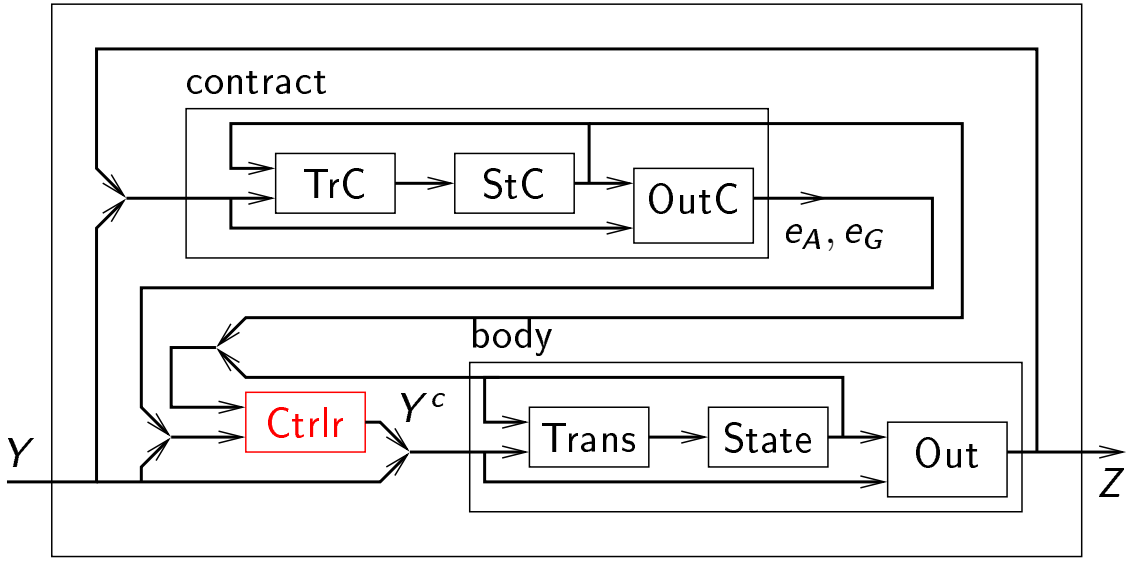
$y_p = f_p(x_1, \ldots, x_n, c_1, \ldots, c_q)$

- Extension d'Heptagon ($\simeq$ Lustre + automates à états)
- À chaque contrat, associe des variables contrôlables, locales au nœud

BZR : contrats et SdC

$f(x_1, \ldots, x_n) = (y_1, \ldots, y_p)$
assume e_A
enforce e_G
with $c_1, \ldots, c_q$

$y_1 = f_1(x_1, \ldots, x_n, c_1, \ldots, c_q)$
$\ldots$
$y_p = f_p(x_1, \ldots, x_n, c_1, \ldots, c_q)$



- Extension d'Heptagon ($\simeq$ Lustre + automates à états)
- À chaque contrat, associe des variables contrôlables, locales au nœud
- Produit modulairement un contrôleur local pour chaque nœud muni de contrat

BZR : contrats et SdC

$f(x_1, \ldots, x_n) = (y_1, \ldots, y_p)$

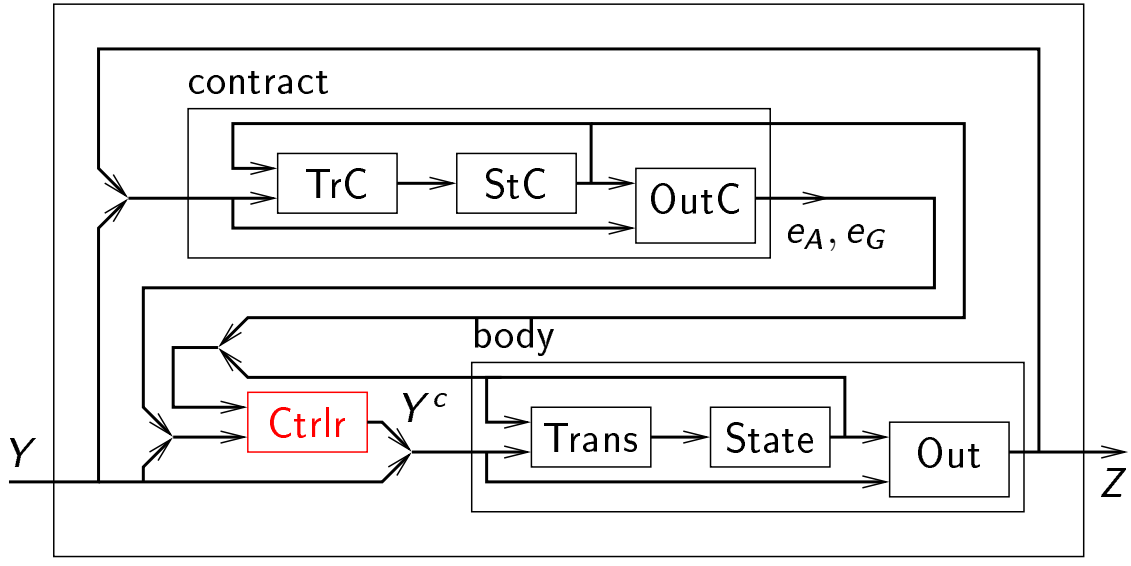
assume e_A

enforce e_G

$y_1 = f_1(x_1, \ldots, x_n)$

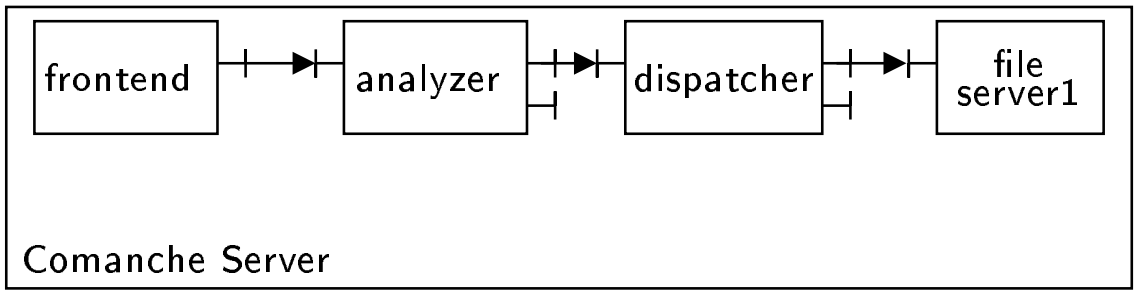
$\ldots$

$y_p = f_p(x_1, \ldots, x_n)$

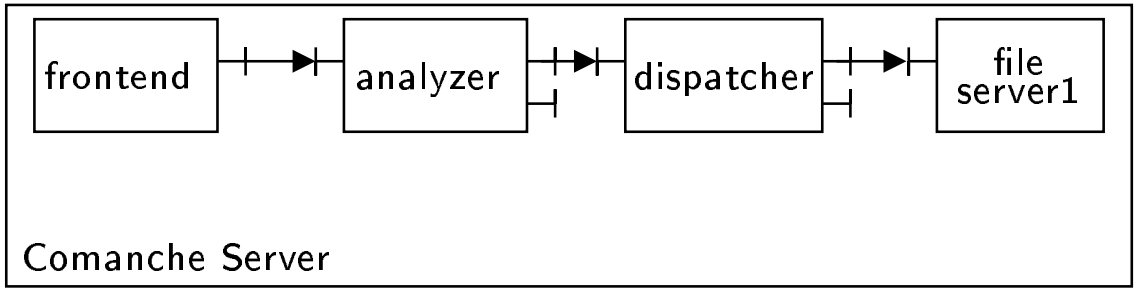


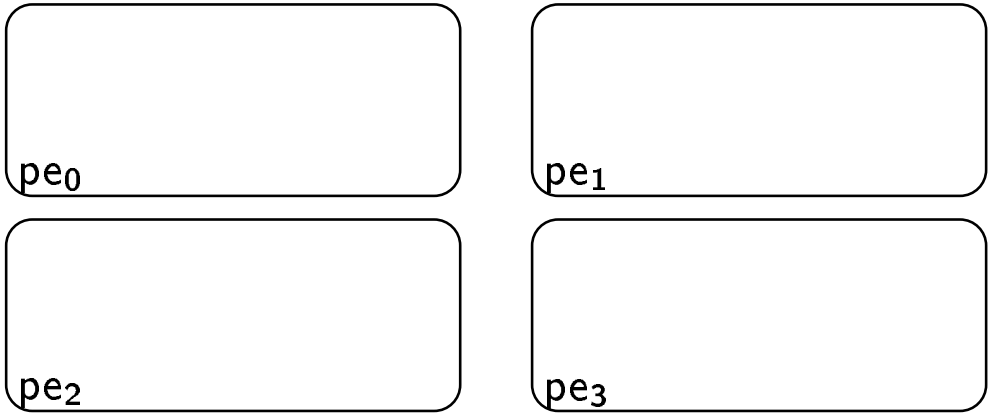
- Extension d'Heptagon ($\simeq$ Lustre + automates à états)
- À chaque contrat, associe des variables contrôlables, locales au nœud
- Produit modulairement un contrôleur local pour chaque nœud muni de contrat

Exemple : Serveur HTTP Comanche [Bouhadiba et al., EMSOFT'11]



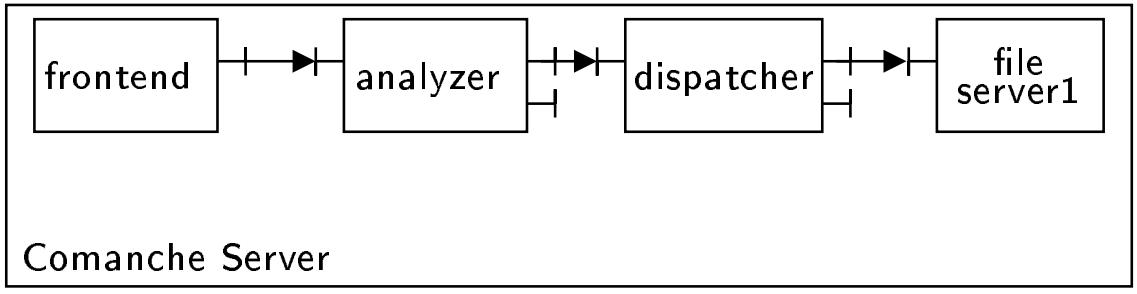
Exemple : Serveur HTTP Comanche [Bouhadiba et al., EMSOFT'11]





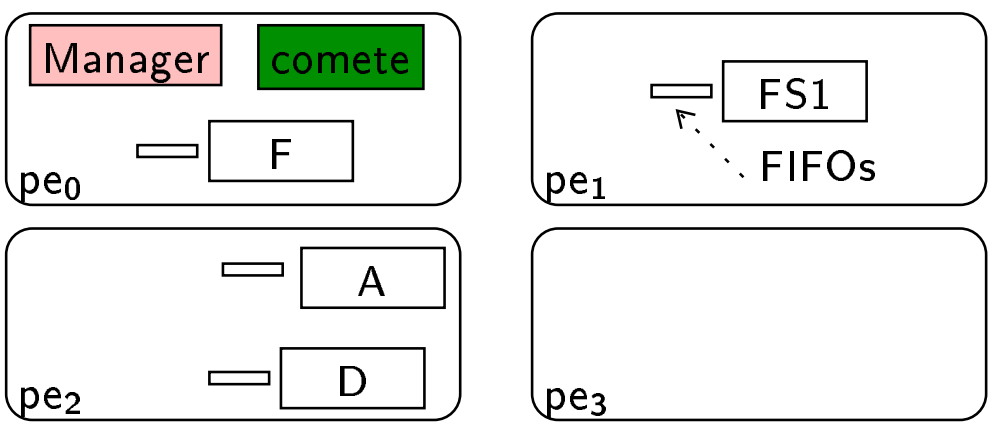
Plateforme d'exécution (4 unités de calcul)

Exemple : Serveur HTTP Comanche [Bouhadiba et al., EMSOFT'11]



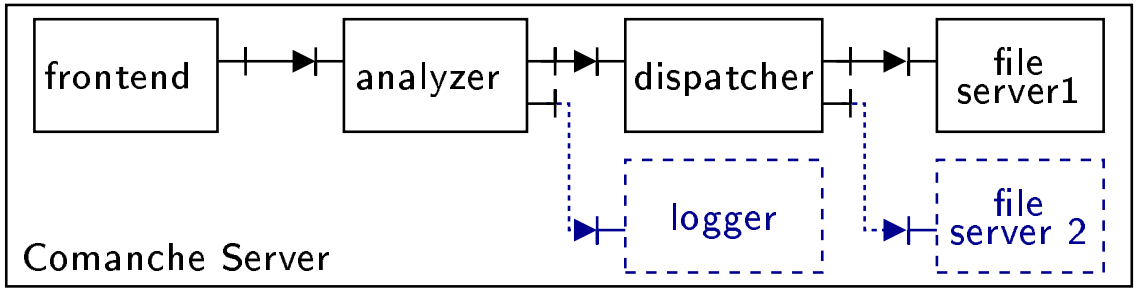
- Les composants sont déployés sur 4 unités de calcul par le middleware Comete
- Liaison $\Longrightarrow$ communications asynchrones par FIFOs

Déploiement



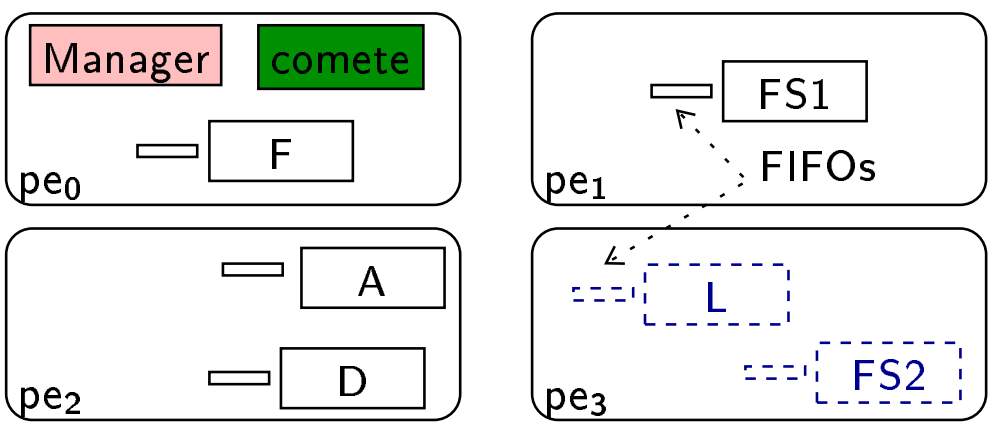
Plateforme d'exécution (4 unités de calcul)

Exemple : Serveur HTTP Comanche [Bouhadiba et al., EMSOFT'11]



- Les composants sont déployés sur 4 unités de calcul par le middleware Comete
- Liaison $\Longrightarrow$ communications asynchrones par FIFOs

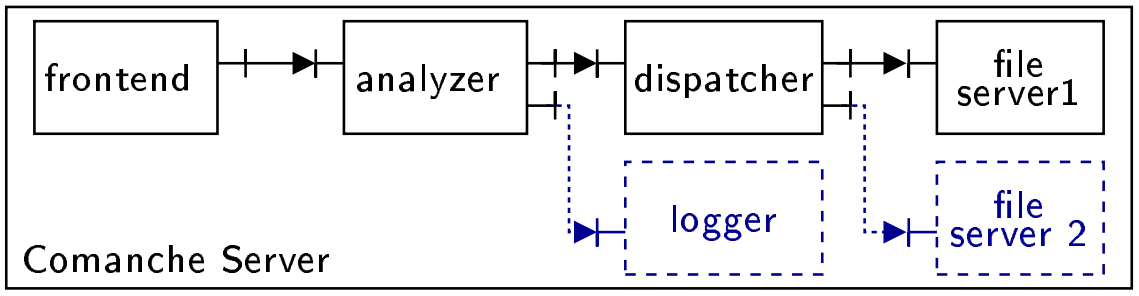
Déploiement



Plateforme d'exécution (4 unités de calcul)

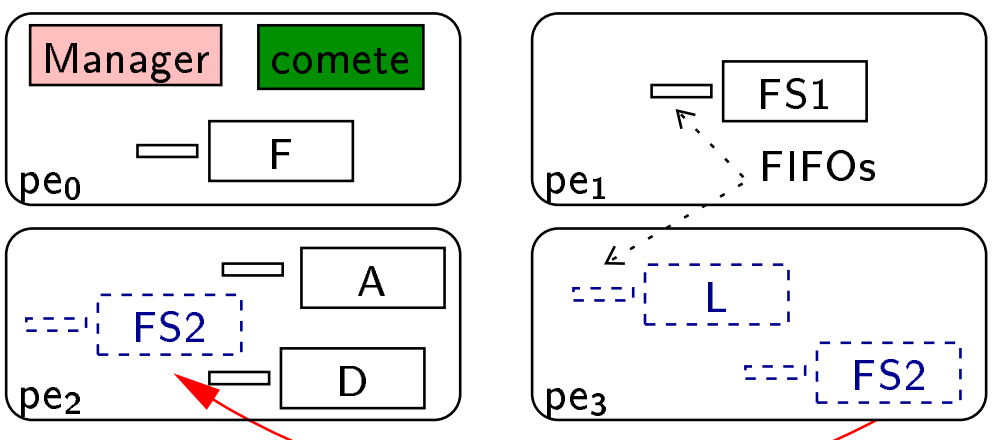
Reconfigurations possibles :
- Démarrer/Arrêter
- Connecter/Déconnecter

Exemple : Serveur HTTP Comanche [Bouhadiba et al., EMSOFT'11]



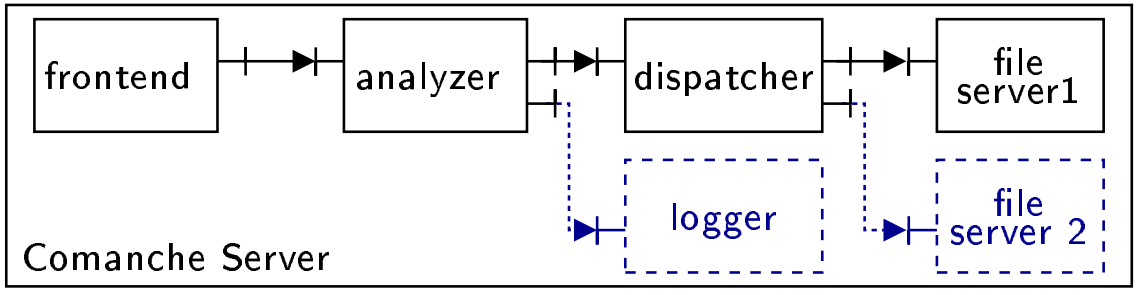
- Les composants sont déployés sur 4 unités de calcul par le middleware Comete
- Liaison $\Longrightarrow$ communications asynchrones par FIFOs

Déploiement



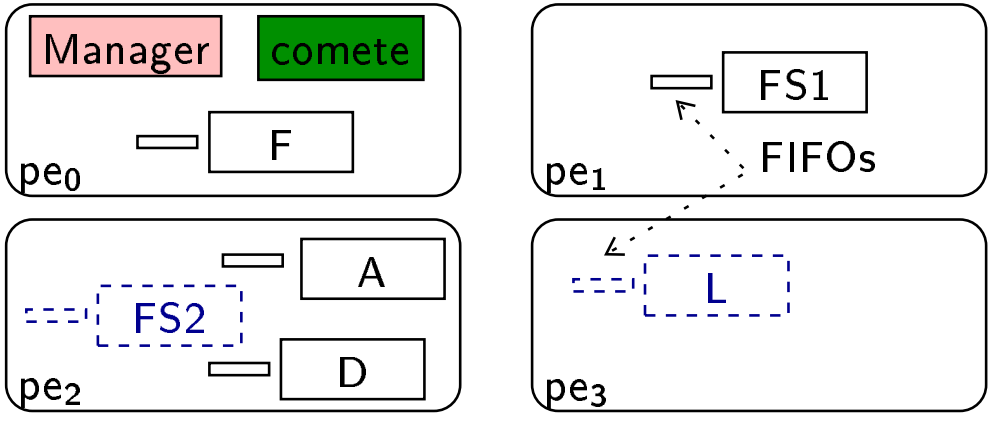
Reconfigurations possibles :
- Démarrer/Arrêter
- Connecter/Déconnecter
- Migrer des composants

Exemple : Serveur HTTP Comanche [Bouhadiba et al., EMSOFT'11]



- Les composants sont déployés sur 4 unités de calcul par le middleware Comete
- Liaison $\Longrightarrow$ communications asynchrones par FIFOs

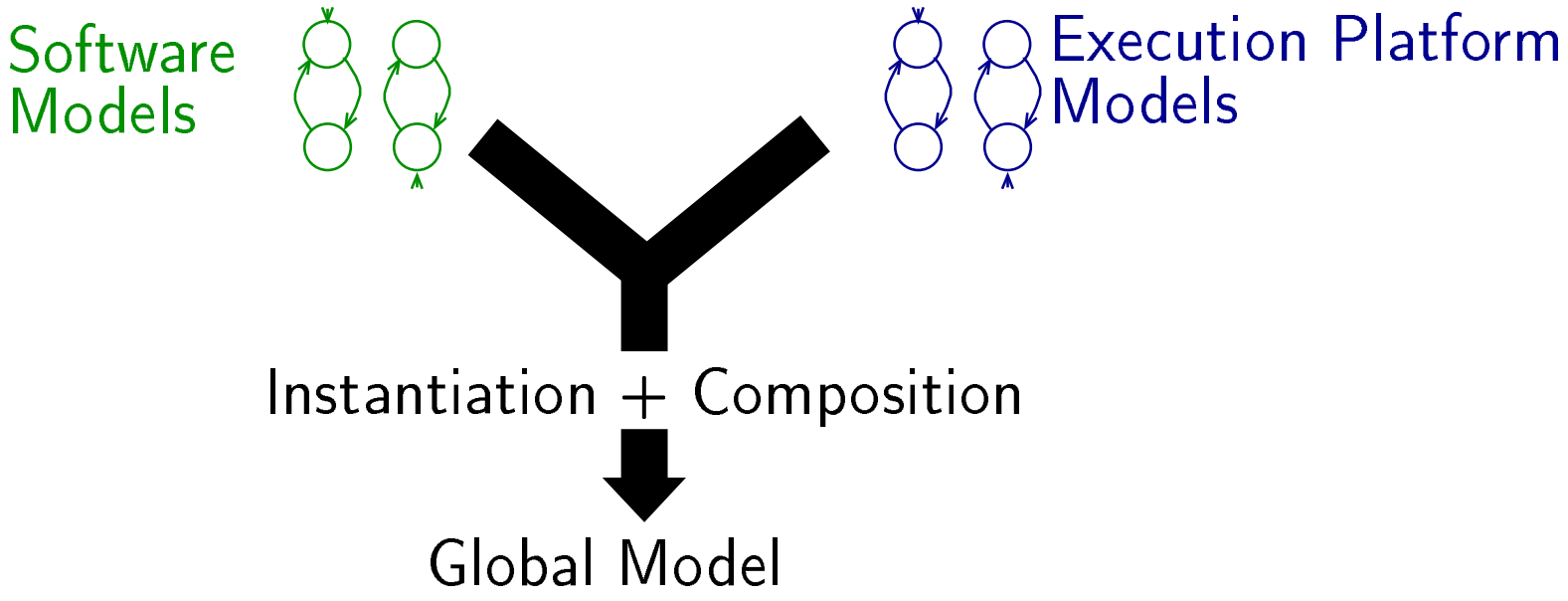
Déploiement



Plateforme d'exécution (4 unités de calcul)

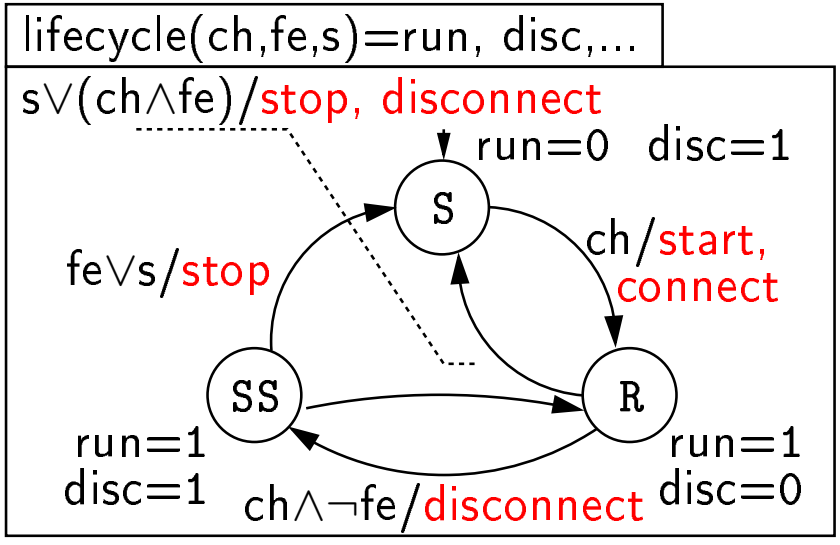
Reconfigurations possibles :
- Démarrer/Arrêter
- Connecter/Déconnecter
- Migrer des composants

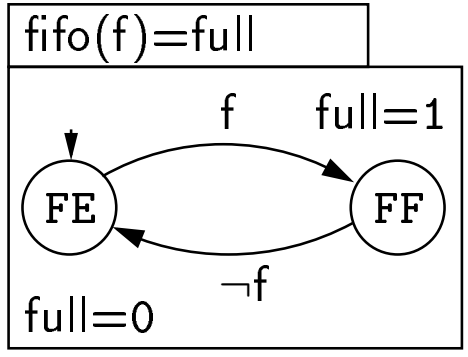
Modélisation des composants



- Approche modulaire
- Distinction modèles matériel/logiciel
- Modèle global = composition synchrone des instances

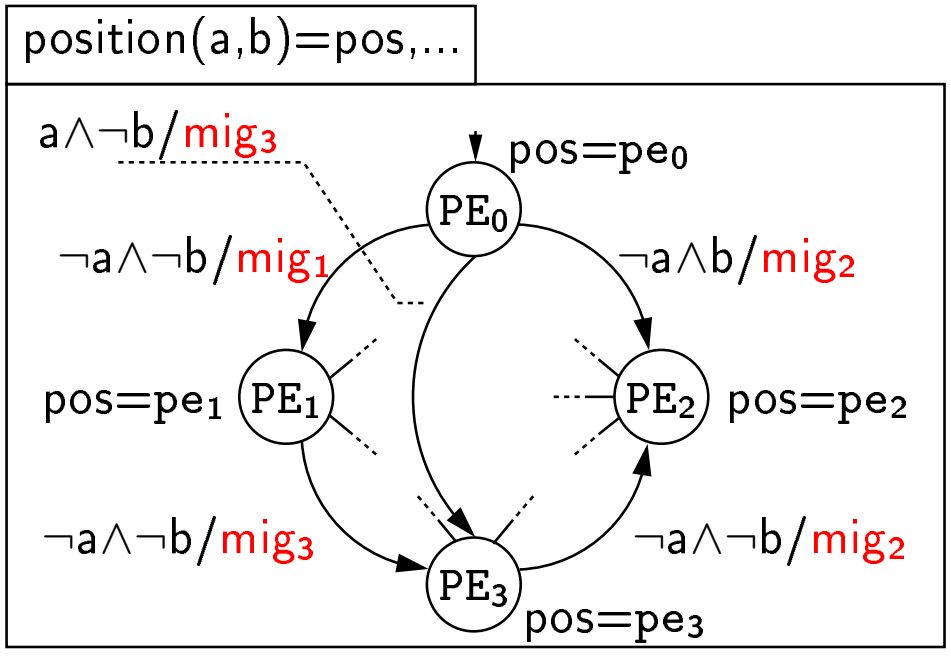
Modèles des composants logiciels

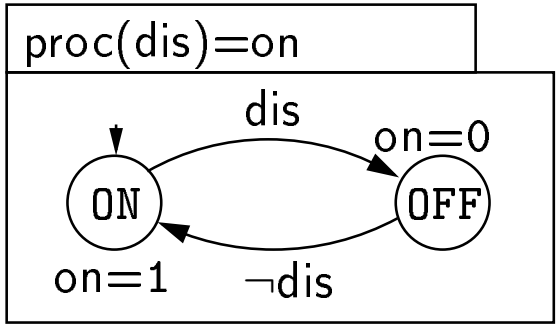




- Le modèle du cycle de vie permet de contrôler l'état du composant. Il déclenche les commandes de reconfiguration du logiciel (start, stop, connect, disconnect).
- Le modèle des files d'attentes reçoit ses entrées de l'application. La sortie (`full`) décrit l'état de la file.

Modèle de la plateforme d'exécution



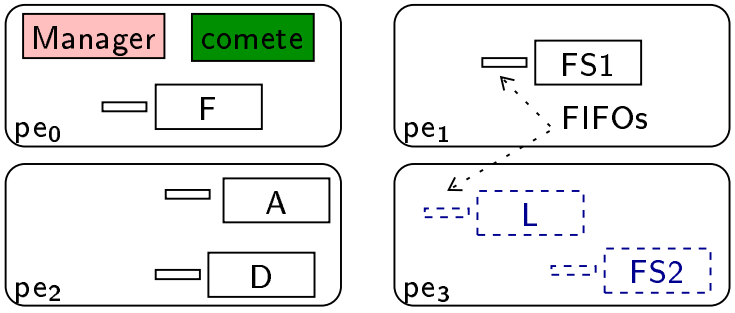


- le nœud `position` permet de contrôler la localisation d'un composant sur la plateforme d'exécution
- le nœud `proc` modélise la disponibilité d'une unité de calcul

Conception du manager

Modélisation :

- Disponibilités des unités de calcul
- Cycle de vie et localisation des composants
- Taille des FIFOs
- Charge des unités de calcul



Description des objectifs (propriétés booléennes) :

- Quand le serveur de fichier 1 est en surcharge, démarrer le serveur 2
- Garder la charge de chaque unité de calcul inférieure à X unités
- Aucun composant ne peut être exécuté sur une unité de calcul non disponible

Sémantique de traces

$[\![\cdot]\!]$: fonction des expressions vers un ensemble de traces

$$\begin{aligned}
\mathcal{N} &: \mathcal{V}^\infty \to \mathcal{P}(\mathcal{V}^\infty \times \mathcal{V}^\infty \times \mathcal{V}^\infty) \\
N &: \mathit{NodeEnv} = \mathit{Var} \to \mathcal{N} \\
\rho &: \mathit{TraceEnv} = \mathit{Var} \to \mathcal{P}(\mathcal{V}^\infty) \\
[\![\cdot]\!] &: \mathit{Exp} \times \mathit{NodeEnv} \times \mathit{TraceEnv} \to \mathcal{P}(\mathcal{V}^\infty)
\end{aligned}$$

$$[\![i]\!]^N_\rho = \{i.i.\ldots.\} \quad \text{(Imm)}$$

$$[\![\mathrm{op}(e)]\!]^N_\rho = \{\mathrm{op}^\infty(s) | s \in [\![e]\!]^N_\rho\} \quad \text{(Op)}$$

(1)

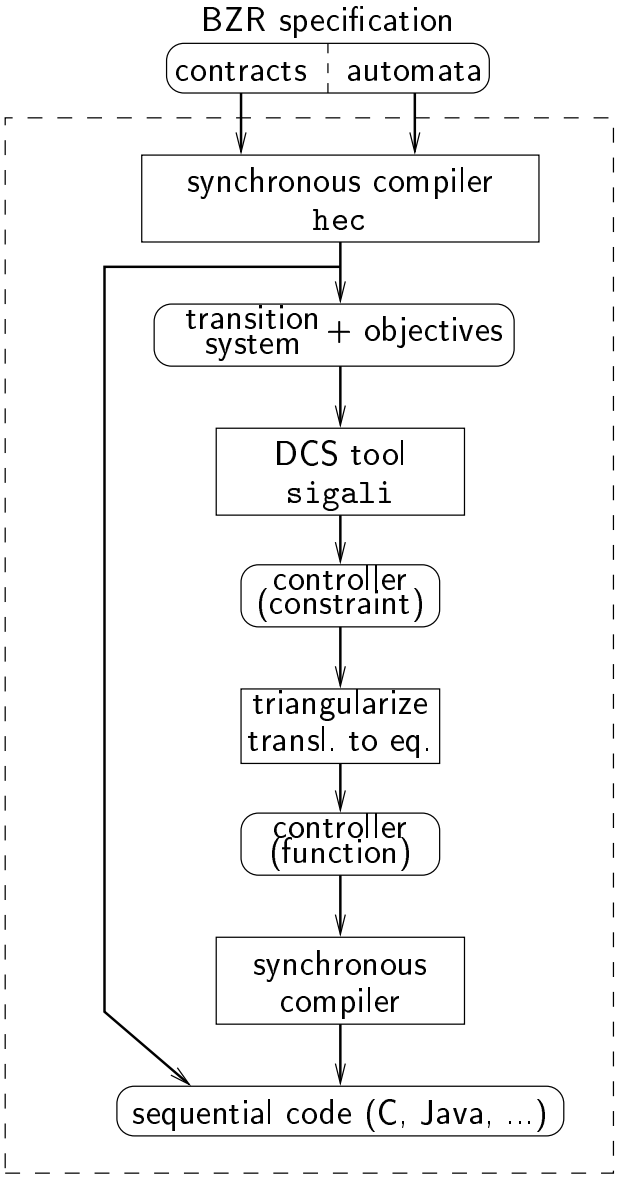
Sémantique des contrats

$$[\![f(e)]\!]^N_\rho = \left\{ \begin{array}{ll} s \text{ s.t.} & (s, s_A, s_G) \in N(f)([\![e]\!]^N_\rho) \\ & \wedge s_A = s_G = [\![\texttt{true}]\!]^N_\rho \end{array} \right\} \qquad \text{(App)}$$

$$[\![\begin{array}{l} \texttt{node } f(x) \texttt{ returns } (y) \\ \quad \texttt{contract } (D_1, e_A, e_G) \\ \quad \texttt{with } c \\ \quad \texttt{let } D_2 \texttt{ tel} \end{array}]\!]^N$$

$$= \lambda s. \left\{ \begin{array}{l} (s_y, s_A, s_G) \text{ s.t. } \exists s_c, \\ \quad \rho = fix(\lambda\rho.([\![D_1 \,;\, D_2]\!]^N_\rho) \\ \qquad\qquad (\{x \mapsto s, c \mapsto s_c\})) \\ \quad s_y \in \rho(y) \\ \quad s_A = [\![e_A]\!]^N_\rho \\ \quad s_G = [\![e_G]\!]^N_\rho \\ \quad True(s_A) \Rightarrow True(s_G) \end{array} \right\} \qquad \text{(NodeC)}$$

Implémentation



Compilation d'Heptagon/BZR :

- «masque» la synthèse de contrôleurs, transparente du point de vue du programmeur (même interface/API)
- le contrôleur synthétisé est lui-même en Heptagon : indépendant du langage cible

Performances

Example name	# state vars	# inputs	# cont.	total # vars	synthesis time (s)	contr. size (# C loc)
bzradmin [ICAS'12]	20	7	3	30	0.11	494
bzrlang	47	6	5	58	0.52	2502
cellphone [ICESS'09]	29	17	6	52	0.61	2346
radiotrans	14	13	2	29	0.10	428
migration [EMSOFT'11]	220	7	8	235	1188.06	49660
httpserver [CBSE'10]	36	2	3	41	0.11	659
robot arm [IFAC'11]	31	23	3	57	0.13	422
provadm [GCM'10]	11	5	2	18	0.01	197
prog2 [LCTES'10]	8	4	2	14	0.01	217
prog4 [LCTES'10]	16	8	4	28	0.06	506
prog6 [LCTES'10]	24	12	6	42	0.40	986
prog8 [LCTES'10]	32	16	8	56	74.07	1535
prog10 [LCTES'10]	40	20	10	70	0.97	2134
prog12 [LCTES'10]	48	24	12	84	2.75	3051
prog14 [LCTES'10]	56	28	14	98	5.73	3976
prog16 [LCTES'10]	64	32	16	112	2990.58	6341
prog18 [LCTES'10]	72	36	18	126	64709.32	9091
prog20 [LCTES'10]	80	40	20	140	947.27	7454

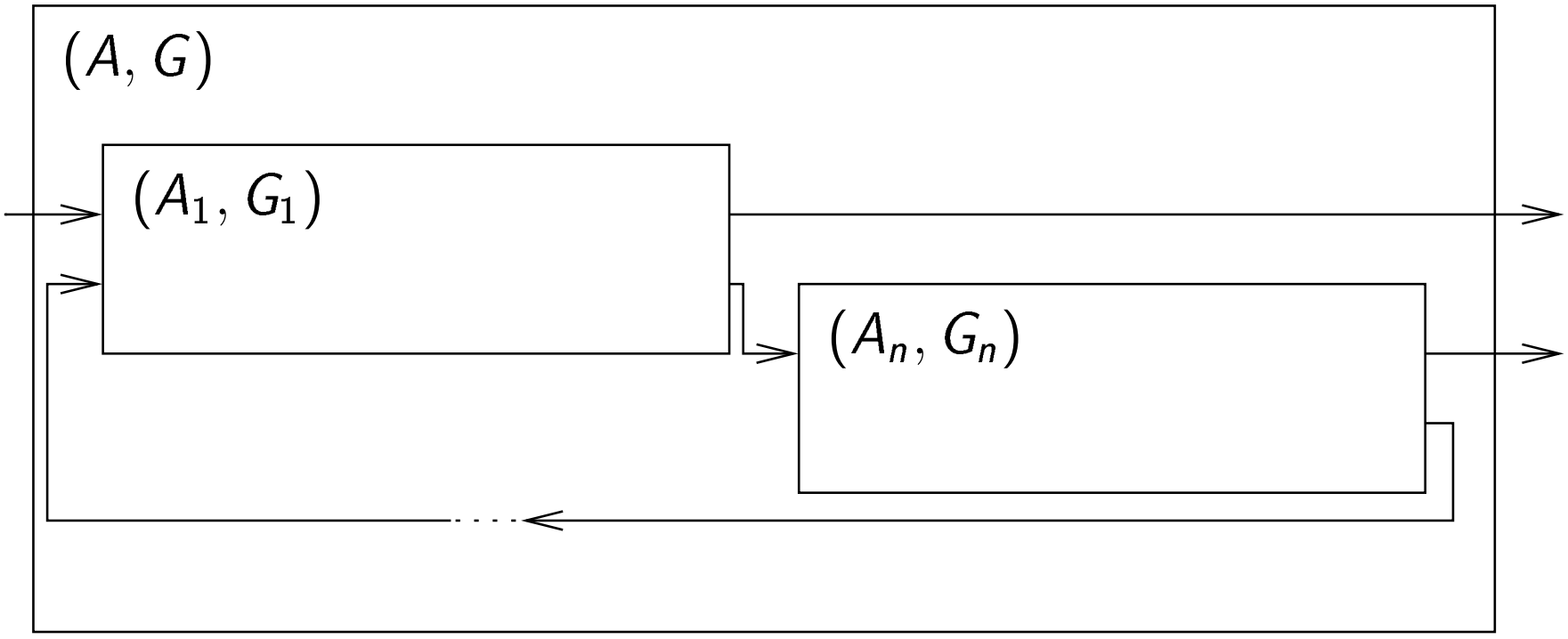
BZR : motivations pour la modularité

- Passage à l'échelle

- Utilisation de composants abstraits/blocs IP/...

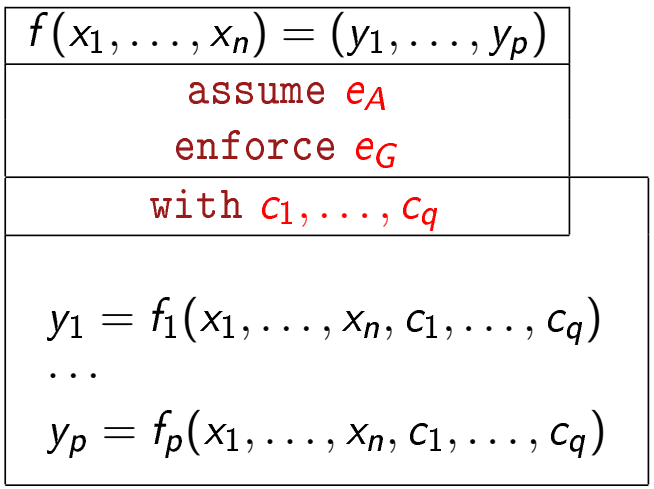
Méthode : utilisation des contrats dans un processus de compilation modulaire...

Contrats et validation



À partir de l'hypothèse d'environnement $\Box A_i \Rightarrow \Box G_i$, $i \in \{1, \ldots, n\}$,
vérifier que $\Box A \Rightarrow \Box G$

Proposition : contrats et SdC



- Associer, à chaque contrat, des variables contrôlables locales

Proposition : contrats et SdC

$f(x_1, \ldots, x_n) = (y_1, \ldots, y_p)$

assume e_A

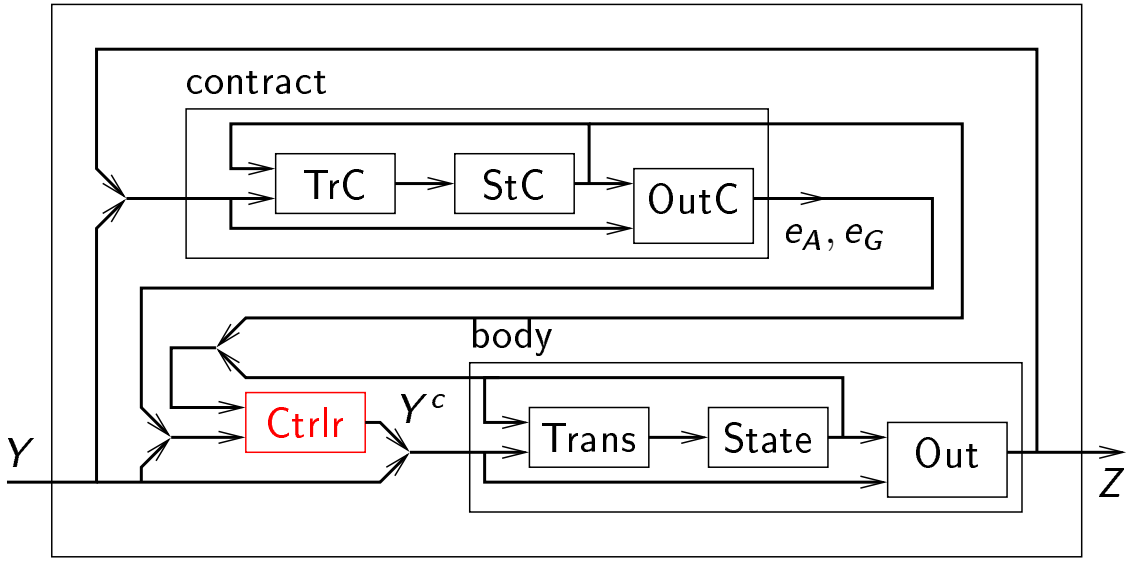
enforce e_G

with $c_1, \ldots, c_q$

$y_1 = f_1(x_1, \ldots, x_n, c_1, \ldots, c_q)$

$\ldots$

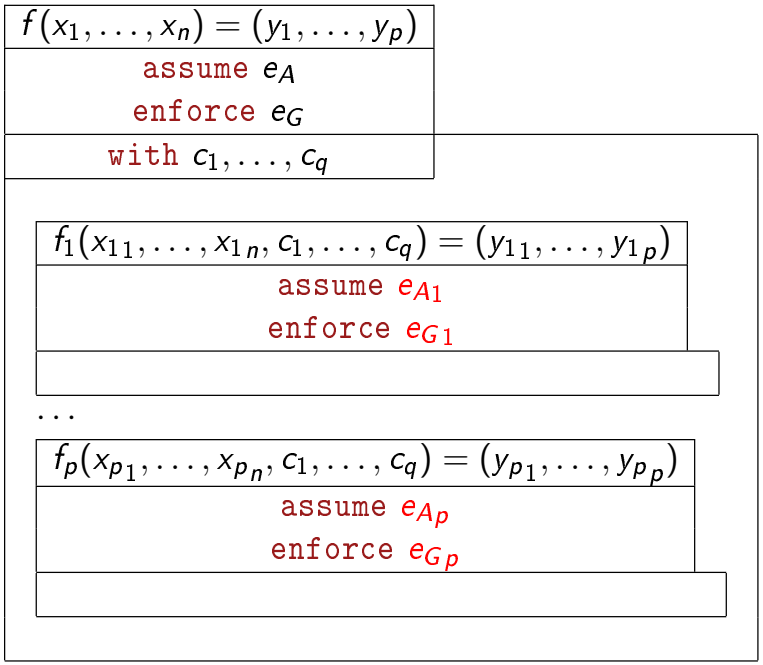
$y_p = f_p(x_1, \ldots, x_n, c_1, \ldots, c_q)$

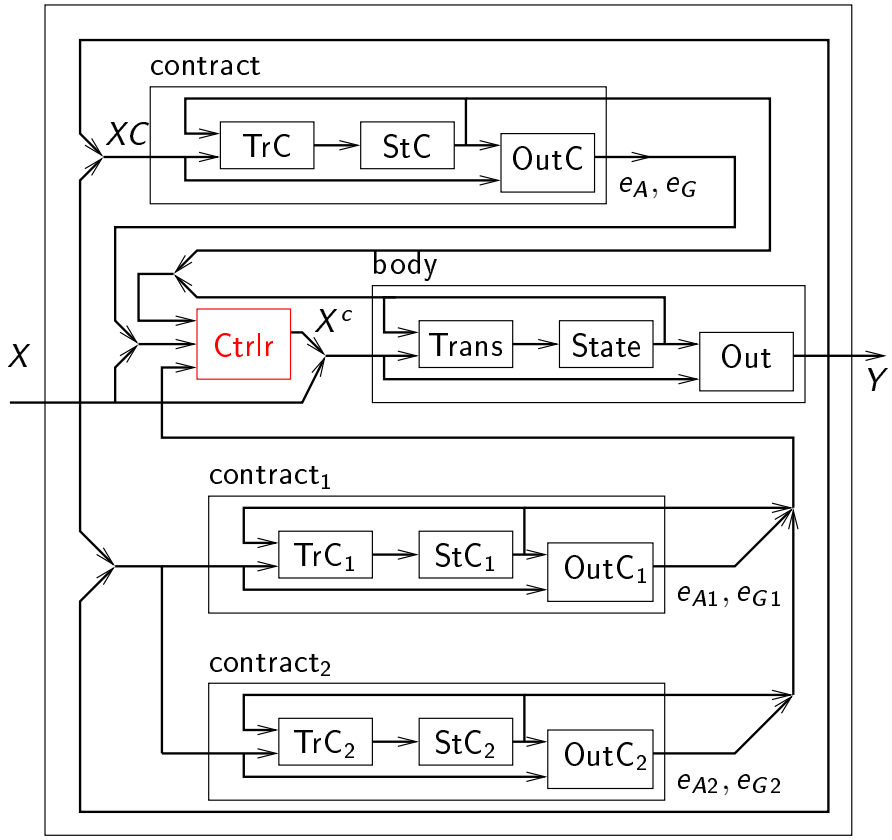


- Associer, à chaque contrat, des variables contrôlables locales
- Synthétiser un contrôleur local pour chaque nœud muni de contrat

Nœuds composites

Utilisation de contrats pour chaque sous-nœud :

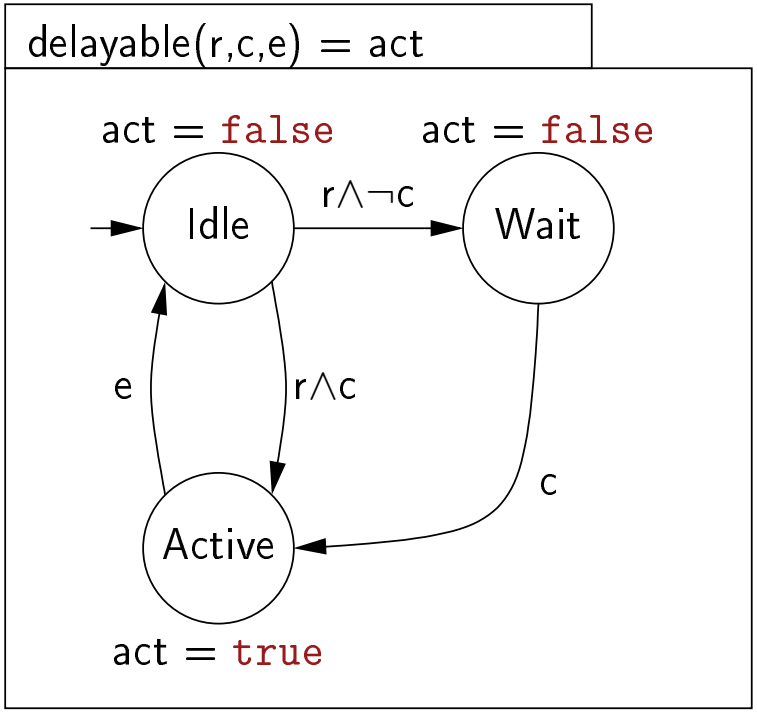




- Objectif de synthèse (propriété à imposer) :
  $\forall\Box\Big((e_{A1} \Rightarrow e_{G1}) \wedge \ldots \wedge (e_{A1} \Rightarrow e_{G1}) \wedge e_A\Big) \Rightarrow \Big(e_G \wedge e_{A1} \wedge \ldots \wedge e_{Ap}\Big)$
- sorties locales y_{ij} vues comme des entrées incontrôlables pour la synthèse de contrôleur

Exemple : tâche retardable

```
node delayable(r,c,e:bool) returns (act:bool)
let
  automaton
    state Idle
      do act = false
      unless (r & c) then Active
           | r then Wait
    state Wait
      do act = false
      unless c then Active
    state Active
      do act = true
      unless e then Idle
  end
tel
```



Exemple (suite)

Ensemble de n tâches retardables exclusives

$$\texttt{ntasks}(r_1, \ldots, r_n, e_1, \ldots, e_n) = (a_1, \ldots, a_n)$$

$$ca_1 = a_1 \wedge (a_2 \vee \ldots \vee a_n)$$
$$\ldots$$
$$ca_{n-1} = a_{n-1} \wedge a_n$$

$$\texttt{assume true}$$
$$\texttt{enforce } \neg(ca_1 \vee \ldots \vee ca_{n-1})$$

$$\texttt{with } c_1, \ldots, c_n$$

$$a_1 = \texttt{inlined delayable}(r_1, c_1, e_1)$$
$$\ldots$$
$$a_n = \texttt{inlined delayable}(r_n, c_n, e_n)$$

Exemple : composition

$$\texttt{main}(r_1, \ldots, r_{2n}, e_1, \ldots, e_{2n}) = (a_1, \ldots, a_{2n})$$

$$ca_1 = a_1 \wedge (a_2 \vee \ldots \vee a_{2n})$$
$$\ldots$$
$$ca_{2n-1} = a_{2n-1} \wedge a_{2n}$$

$$\texttt{assume true}$$
$$\texttt{enforce } \neg(ca_1 \vee \ldots \vee ca_{2n-1})$$
$$\texttt{with } \emptyset$$

$$(a_1, \ldots, a_n) = \texttt{ntasks}(r_1, \ldots, r_n, e_1, \ldots, e_n)$$
$$(a_{n+1}, \ldots, a_{2n}) = \texttt{ntasks}(r_{n+1}, \ldots, r_{2n}, e_{n+1}, \ldots, e_{2n})$$

$\longrightarrow$ le nœud `ntasks` n'est pas assez contrôlable pour assurer le contrat du nœud `main`

Example (correction, version naïve)

Contract refinement for composition of several `ntasks` components :

$$\texttt{ntasks}(\texttt{c}, \texttt{r}_1, \ldots, \texttt{r}_n, \texttt{e}_1, \ldots, \texttt{e}_n) = (\texttt{a}_1, \ldots, \texttt{a}_n)$$

$$\texttt{ca}_1 = \texttt{a}_1 \wedge (\texttt{a}_2 \vee \ldots \vee \texttt{a}_n)$$
$$\ldots$$
$$\texttt{ca}_{n-1} = \texttt{a}_{n-1} \wedge \texttt{a}_n$$
$$\texttt{one} = \texttt{a}_1 \vee \ldots \vee \texttt{a}_n$$

$$\texttt{assume true}$$
$$\texttt{enforce } \neg(\texttt{ca}_1 \vee \ldots \vee \texttt{ca}_{n-1}) \wedge (\texttt{c} \vee \neg\texttt{one})$$

$$\texttt{with } \texttt{c}_1, \ldots, \texttt{c}_n$$

$$\texttt{a}_1 = \texttt{inlined delayable}(\texttt{r}_1, \texttt{c}_1, \texttt{e}_1)$$
$$\ldots$$
$$\texttt{a}_n = \texttt{inlined delayable}(\texttt{r}_n, \texttt{c}_n, \texttt{e}_n)$$

Exemple : composition, 2^{e} essai

$$\texttt{main}(r_1, \ldots, r_{2n}, e_1, \ldots, e_{2n}) = (a_1, \ldots, a_{2n})$$

$$ca_1 = a_1 \wedge (a_2 \vee \ldots \vee a_{2n})$$
$$\ldots$$
$$ca_{2n-1} = a_{2n-1} \wedge a_{2n}$$

$$\texttt{assume true}$$
$$\texttt{enforce } \neg(ca_1 \vee \ldots \vee ca_{2n-1})$$

$$\texttt{with } c_1, c_2$$

$$(a_1, \ldots, a_n) = \texttt{ntasks}(c_1, r_1, \ldots, r_n, e_1, \ldots, e_n)$$
$$(a_{n+1}, \ldots, a_{2n}) = \texttt{ntasks}(c_2, r_{n+1}, \ldots, r_{2n}, e_{n+1}, \ldots, e_{2n})$$

$\longrightarrow$ la synthèse réussit, mais les contrôleurs du nœud `ntasks` ne peuvent pas autoriser une seule des n tâches à aller dans l'état actif !

Exemple (version correcte)

$$\texttt{ntasks}(c, r_1, \ldots, r_n, e_1, \ldots, e_n) = (a_1, \ldots, a_n)$$

$$\begin{aligned} ca_1 &= a_1 \wedge (a_2 \vee \ldots \vee a_n) \\ &\ldots \\ ca_{n-1} &= a_{n-1} \wedge a_n \\ one &= a_1 \vee \ldots \vee a_n \\ pone &= \texttt{false fby } one \end{aligned}$$

$$\texttt{enforce } \neg(ca_1 \vee \ldots \vee ca_{n-1}) \wedge (c \vee \neg(one \wedge \neg pone))$$

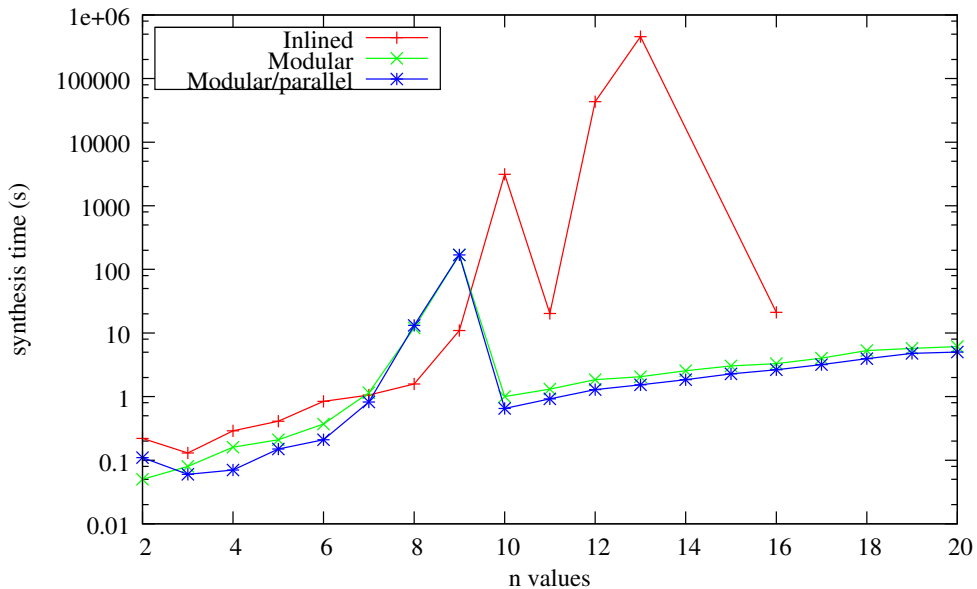
$$\texttt{with } c_1, \ldots, c_n$$

$$\begin{aligned} a_1 &= \texttt{inlined delayable}(r_1, c_1, e_1) \\ &\ldots \\ a_n &= \texttt{inlined delayable}(r_n, c_n, e_n) \end{aligned}$$

Performances

Comparaison des temps de synthèse pour :

- $3n$ tâches exclusives inlinées ($2n$ «rejetables» et n «retardables»)
- $3n$ tâches, factorisées en 3 instantiations
- $3n$ tâches, factorisées, synthèses effectuées en parallèle (sur un multi-cœur)



Cycle de conception (réel) avec Heptagon/BZR

Programmeur

chaîne d'outils BZR

Voici un programme non déterministe. Est-il possible de le contraindre afin qu'il vérifie la propriété P ?

Ce n'est pas possible.

Voici un programme non déterministe (modifié). Est-il possible de le contraindre afin qu'il vérifie la propriété P ?

Ce n'est pas possible.

Voici un programme non déterministe (modifié). Est-il possible de le

Conclusion

- Langage de programmation mixte impératif/déclaratif
- Propriétés et contrats assurés à la compilation
- Utilisation de la synthèse de contrôleurs discrets en programmation

Conclusion

- Langage de programmation mixte impératif/déclaratif
- Propriétés et contrats assurés à la compilation
- Utilisation de la synthèse de contrôleurs discrets en programmation

Utilisations connues

- Contrôle discret de tâches temps-réelles [IFAC 2011]
- Contrôle de reconfigurations dans le modèle Fractal [EMSOFT 2011]
- Coordination de boucles d'administration dans des systèmes autonomiques [GCM 2010, Gueye et al, ICAS 2012]

Travaux en cours et perspectives

- Méthodes de programmation avec la synthèse de contrôleurs : travaux croisés entre méthodes issues de l'informatique et de l'automatique
- Verrous méthodologiques : diagnostics (comment caractériser, détecter et résoudre un échec de la SdC ?), debugging (controller in the loop)
- Utilisation à l'exécution du contrôleur synthétisé : déterminisation, contrôlabilité, équité, lisibilité/compréhensibilité du point de vue du programmeur
- Intégration de nouveaux algorithmes de synthèse : synthèse optimale, synthèse utilisant des variables numériques