Vérification de modèles AltaRica

Aymeric.Vincent@labri.fr

22 octobre 2003

Equipes et outils

OCAS AltaRica par Dassault : modélisation, simulation, arbres de défaillance

AltaRicaDF par ARBoost technologies : simulation, arbres de défaillance, stochastique

altatools au LaBRI (Gérald Point) : simulation, sémantique, vérification, conversions $\leftrightarrow$ Lustre

Timed AltaRica à l'IRCCyN (Claire Pagetti) : langage AltaRica temporisé, et traduction vers Uppaal (thèse en cours)

Mec V au LaBRI (Aymeric Vincent) : vérification (thèse en cours)

AltaRica : des automates à contraintes

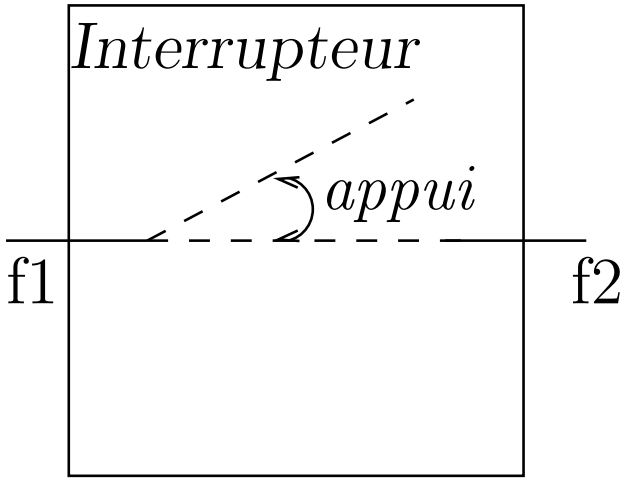
Des variables $\vec{s}$

Des événements E

Des transitions gardées :

$$G(\vec{s}) \xrightarrow{e} \vec{s} := f(\vec{s})$$

Exemple (interrupteur)



Exemple (interrupteur)

```
node Interrupteur
  state ouvert : bool;
  flow f1, f2 : bool;
  event appui;
  trans
    true |- appui -> ouvert := ~ouvert;
  assert
    (~ouvert) => (f1 = f2);
  init
    ouvert := false;
edon
```

Relations

Mec 5 ne manipule que des relations

Un nœud AltaRica est transformé en :

- son ensemble de configurations initiales
- sa relation de transitions

Configurations initiales

Exemple sur le nœud `Switch`

deux configurations initiales :

```
~ouvert   f1   f2
~ouvert  ~f1  ~f2
```

Relation de transitions

Exemple sur le nœud `Switch` :

`~ouvert`	`f1`	`f2`	`appui`	`ouvert`	`~f1`	`~f2`
`~ouvert`	`f1`	`f2`	`appui`	`ouvert`	`f1`	`~f2`
`~ouvert`	`f1`	`f2`	`appui`	`ouvert`	`~f1`	`f2`
`~ouvert`	`f1`	`f2`	`appui`	`ouvert`	`f1`	`f2`
			⋮			

Le langage de spécification

Propriétés élémentaires AltaRica

`A.s = 1`, `x + y = z`, ...

Opérateurs logiques AltaRica

`~p`, `p & q`, `p | q`, `ite(p, e1, e2)`, ...

Le langage de spécification (2)

Quantificateurs du premier ordre

```
[b : bool] p, <b : bool> p
```

Définition de relations par points fixes

```
R(s : [0, 10]) += s = 4 | s <= 9 & R(s + 1);
```

Et par systèmes d'équations de points fixes.

→ Logique très expressive

Pourquoi cette logique ?

Choix dirigé par la technologie : BDD, Toupie

Permet d'exprimer toutes les formules de CTL, LTL, CTL*

Permet d'exprimer la bisimulation

Grande culture "μ-calcul — jeux" à Bordeaux

Outil d'expérimentation pour le μ-calcul

Le langage de spécification (3)

Chaque nœud AltaRica `A` définit :

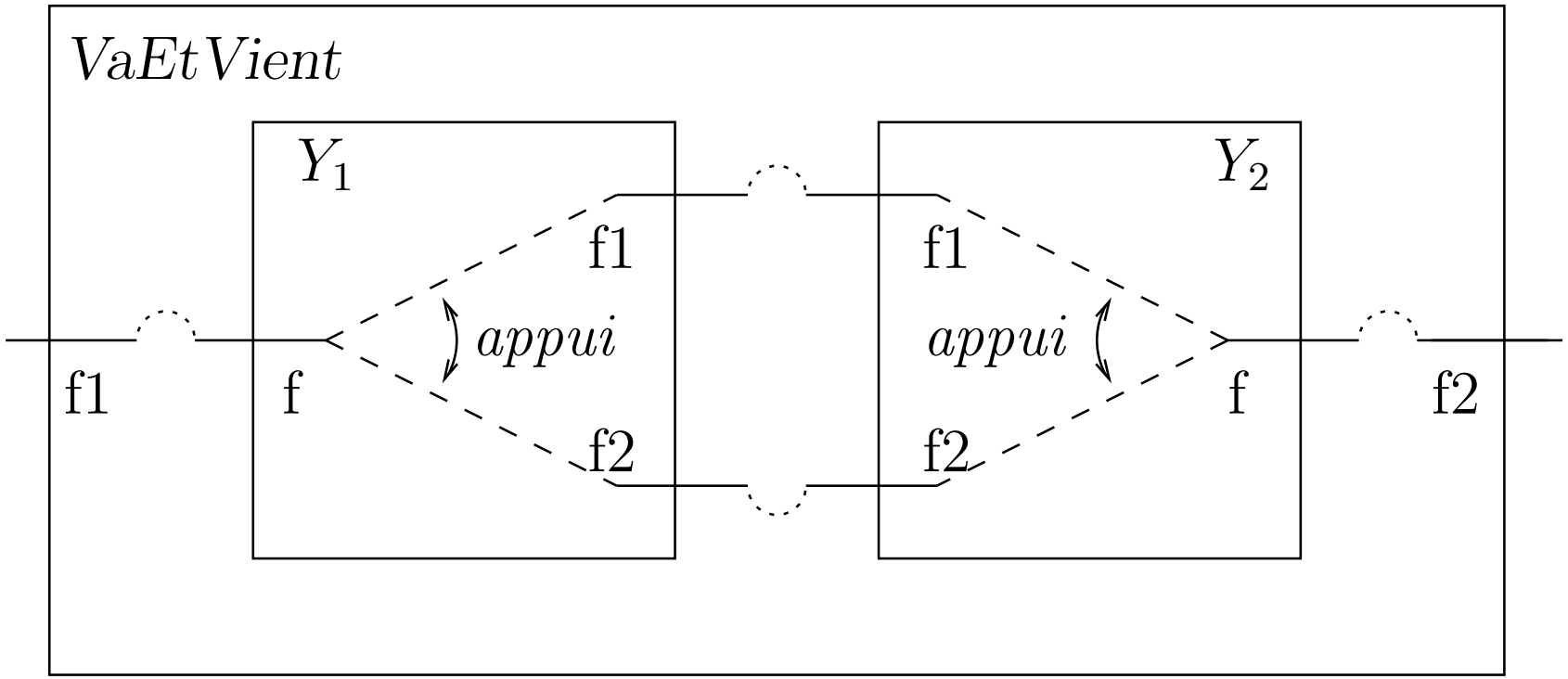
- Deux types :
  - `A!c` des configurations de `A`
  - `A!ev` des vecteurs d'événements de `A`
- Deux prédicats :
  - `A!init` des états initiaux de `A` (type `A!c`)
  - `A!t` des transitions de `A` (type `A!c` × `A!ev` × `A!c`)

Le langage de spécification (4)

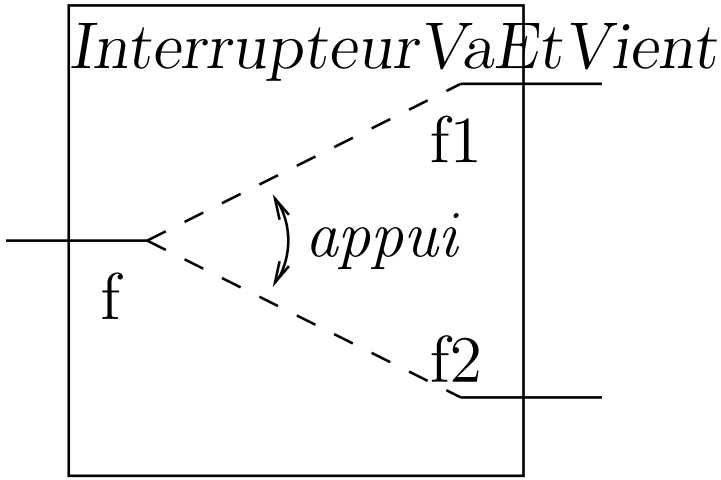
Accesseurs des variables de type “AltaRica” :

- `A!c` : accès à la hiérarchie du nœud `A` par notation pointée
- `A!ev` : accès aux événements du vecteur par notation pointée, et test d’égalité avec une constante

Exemple : le va-et-vient



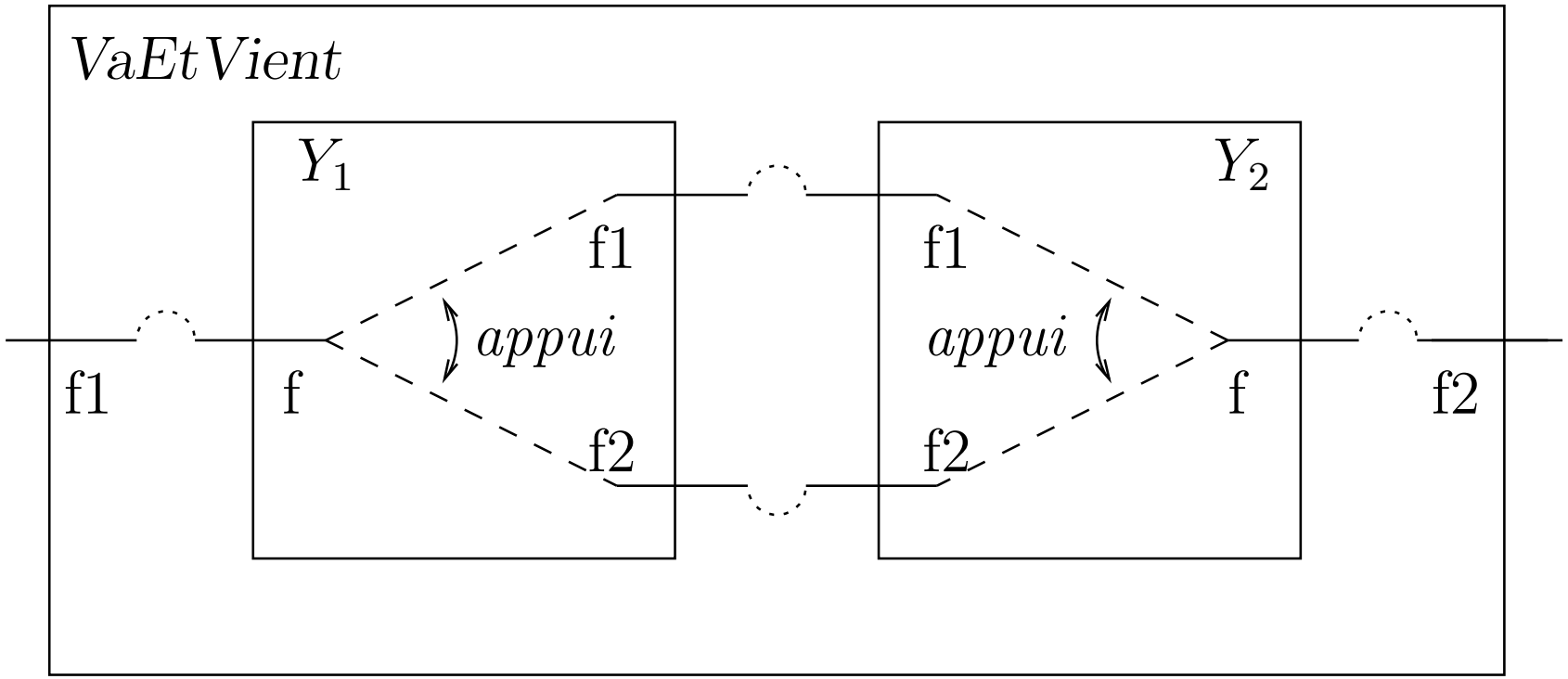
Exemple (Interrupteur de va-et-vient)



Exemple (Interrupteur de va-et-vient)

```
node InterrupteurVaEtVient
  state choix : [1, 2];
  flow f, f1, f2 : bool;
  event appui;
  trans
    choix = 1 |- appui -> choix := 2;
    choix = 2 |- appui -> choix := 1;
  assert
    (choix = 1) => (f = f1);
    (choix = 2) => (f = f2);
edon
```

Exemple : le va-et-vient



Exemple : le va-et-vient

```
node VaEtVient
  flow f1, f2 : bool;
  sub I1, I2 : InterrupteurVaEtVient;
  event appui
  trans true |- appui -> ;
  sync
    <appui, I1.appui>;
    <appui, I2.appui>;
  assert
    f1 = I1.f;
    I1.f1 = I2.f1 & I1.f2 = I2.f2;
    f2 = I2.f;
edon
```

L'interrupteur et le va-et-vient sont-ils bisimilaires ?

Equivalences de base

```
eqEv(a : Interrupteur!ev, b : VaEtVient!ev) :=
    (a. = appui & b. = appui) |
    (a. = "" & b. = "");

eqC(a : Interrupteur!c, b : VaEtVient!c) :=
  (a.f1 = b.f1) & (a.f2 = b.f2);
```

Bisimulation

```
bisim(a, a') -= eqC(a,a') &
  ([e][s] (Interrupteur!t(a,e,s) =>
    <e'><s'> (VaEtVient!t(a',e',s') &
      eqEv(e,e') & bisim(s,s')))) &
  ([e'][s'] (VaEtVient!t(a',e',s') =>
    <e><s> (Interrupteur!t(a,e,s) &
      eqEv(e,e') & bisim(s,s'))));
```

Résultat

```
estBisim(x : bool) :=
  x = (([a]<a'> bisim(a,a')) &
       ([a']<a> bisim(a,a')));
```

```
[mec5] :display estBisim
(true)
[mec5]
```

Donc les deux systèmes sont bisimilaires.

Intérêt :

	états	transitions
Switch + Lampe + Générateur	12	76
SwitchSystem + Lampe + Générateur	32	320