Conception et réalisation d’un vérificateur de modèles AltaRica

Aymeric Vincent

LaBRI, Université Bordeaux 1

Contexte

AltaRica

Contexte

AltaRica

- Système hydraulique et électrique de l'A340, CERT-ONERA pour EADS

Contexte

AltaRica

- Système hydraulique et électrique de l'A340, CERT-ONERA pour EADS
- Alimentation électrique de plate-formes off-shore, Total

Contexte

AltaRica

- Système hydraulique et électrique de l'A340, CERT-ONERA pour EADS
- Alimentation électrique de plate-formes off-shore, Total
- Sous-systèmes du Falcon F7X, Dassault Aviation

Contexte

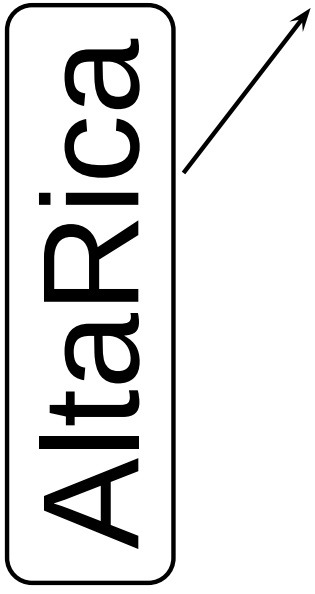
AltaRica

- Système hydraulique et électrique de l'A340, CERT-ONERA pour EADS
- Alimentation électrique de plate-formes off-shore, Total
- Sous-systèmes du Falcon F7X, Dassault Aviation
- Etude d'un brevet sur le contrôle des téléphones portables, LaBRI

Contexte

AltaRica

Contexte



Arbre de défaillances

Formule booléenne

Contexte

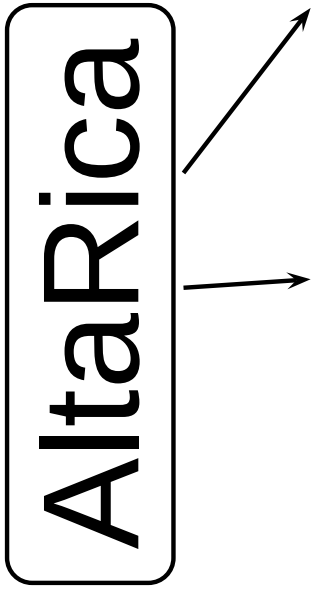
AltaRica

Arbre de défaillances
Formule booléenne

$\rightsquigarrow$

Pr(Evénement redouté)
Implicants premiers

Contexte



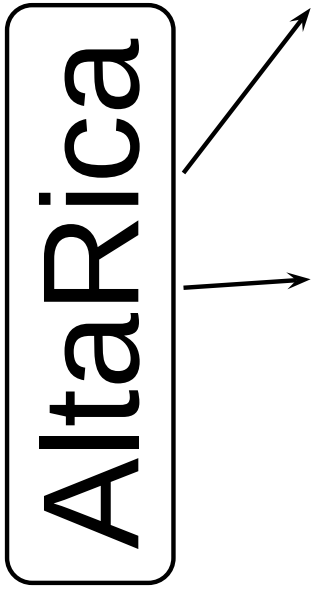
Arbre de défaillances
Formule booléenne

Réseau de Petri
Chaîne de Markov

$\rightsquigarrow$

Pr(Evénement redouté)
Implicants premiers

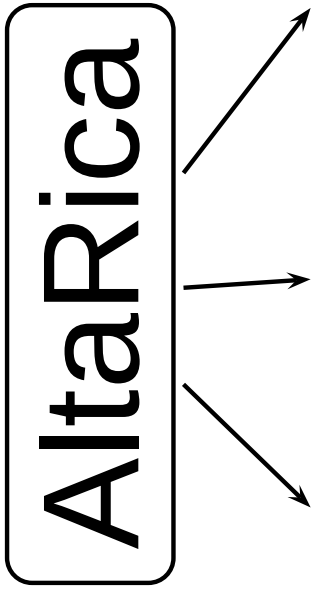
Contexte



Arbre de défaillances
Formule booléenne
$\rightsquigarrow$
Pr(Evénement redouté)
Implicants premiers

Réseau de Petri
Chaîne de Markov
$\rightsquigarrow$
Simulation stochastique
Disponibilité

Contexte

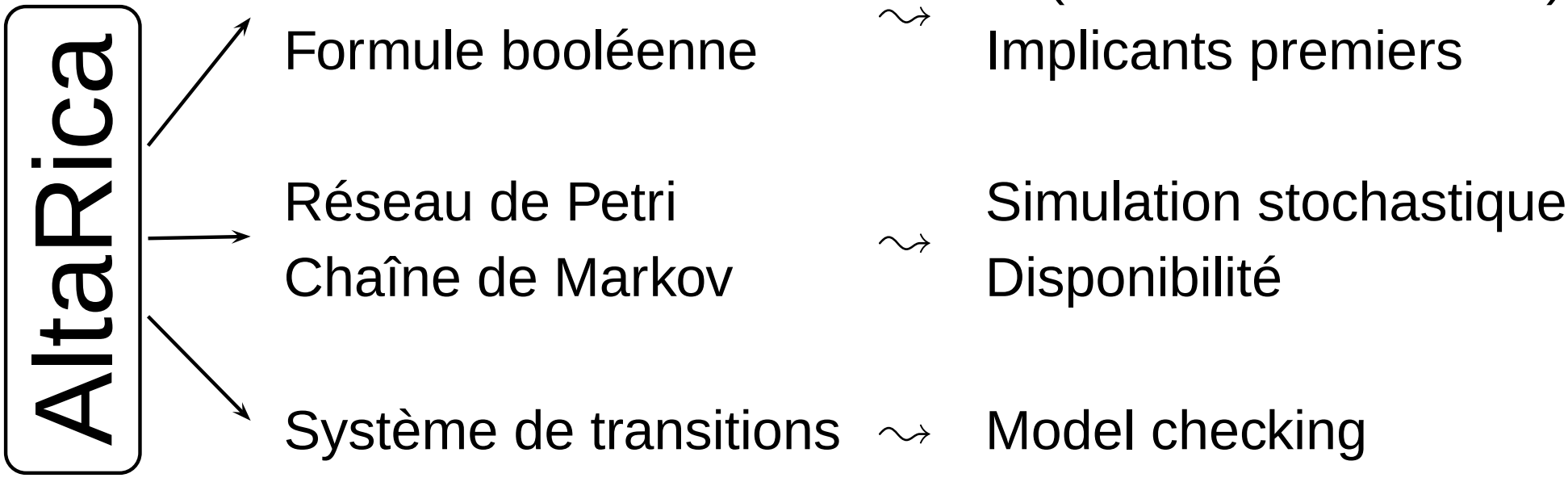


Arbre de défaillances
Formule booléenne
$\rightsquigarrow$
Pr(Evénement redouté)
Implicants premiers

Réseau de Petri
Chaîne de Markov
$\rightsquigarrow$
Simulation stochastique
Disponibilité

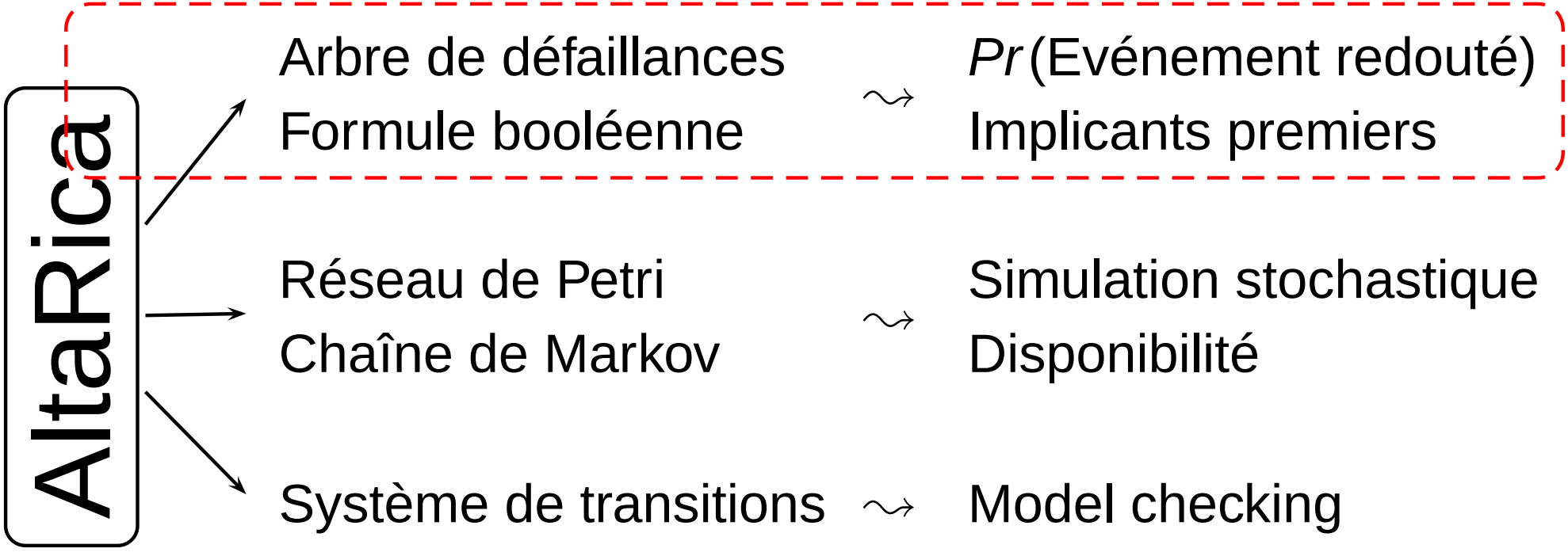
Système de transitions

Contexte


AltaRica
Arbre de défaillances
Formule booléenne
⇝
Pr (Evénement redouté)
Implicants premiers
Réseau de Petri
Chaîne de Markov
⇝
Simulation stochastique
Disponibilité
Système de transitions
⇝
Model checking

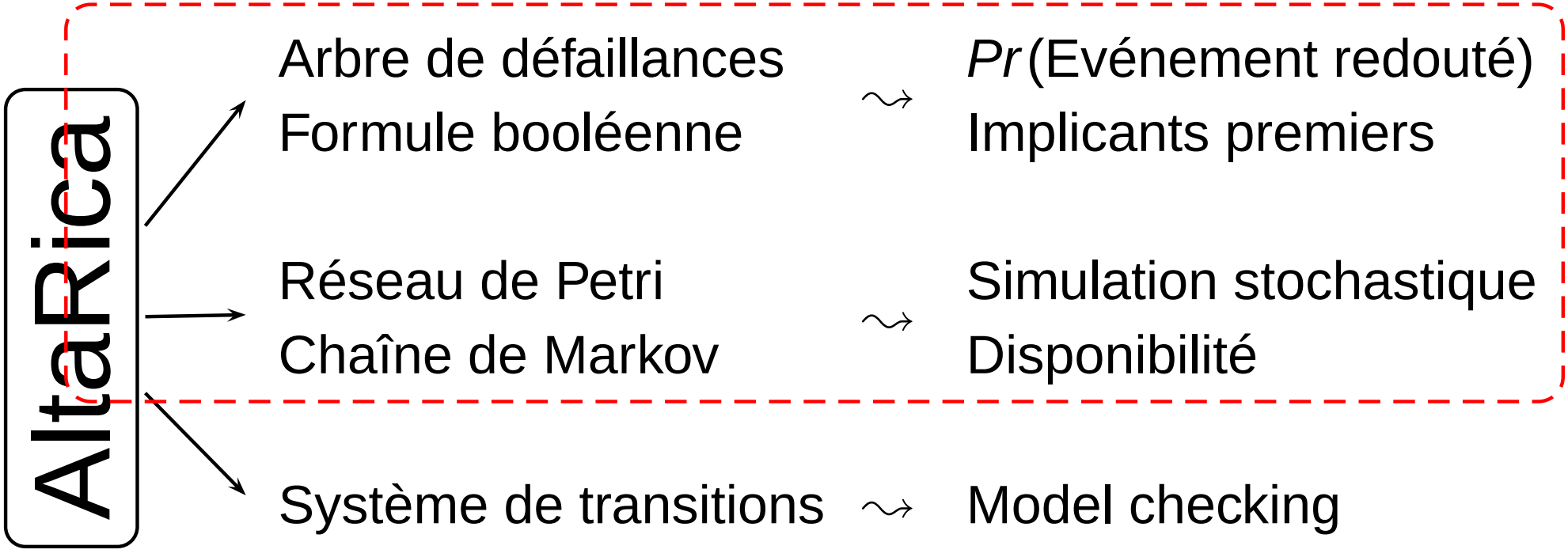
Contexte

Cecilia/OCAS, par GFI Consulting et Dassault (2003)

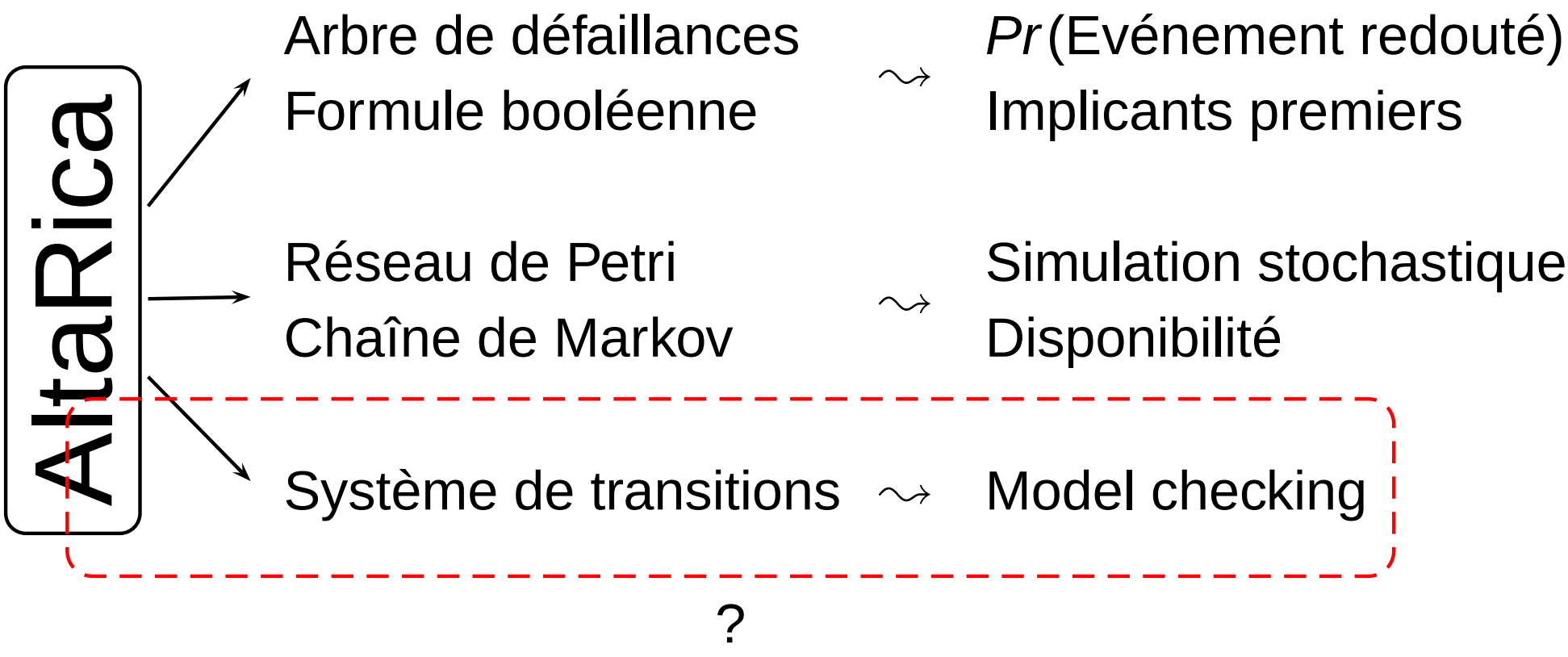


Contexte

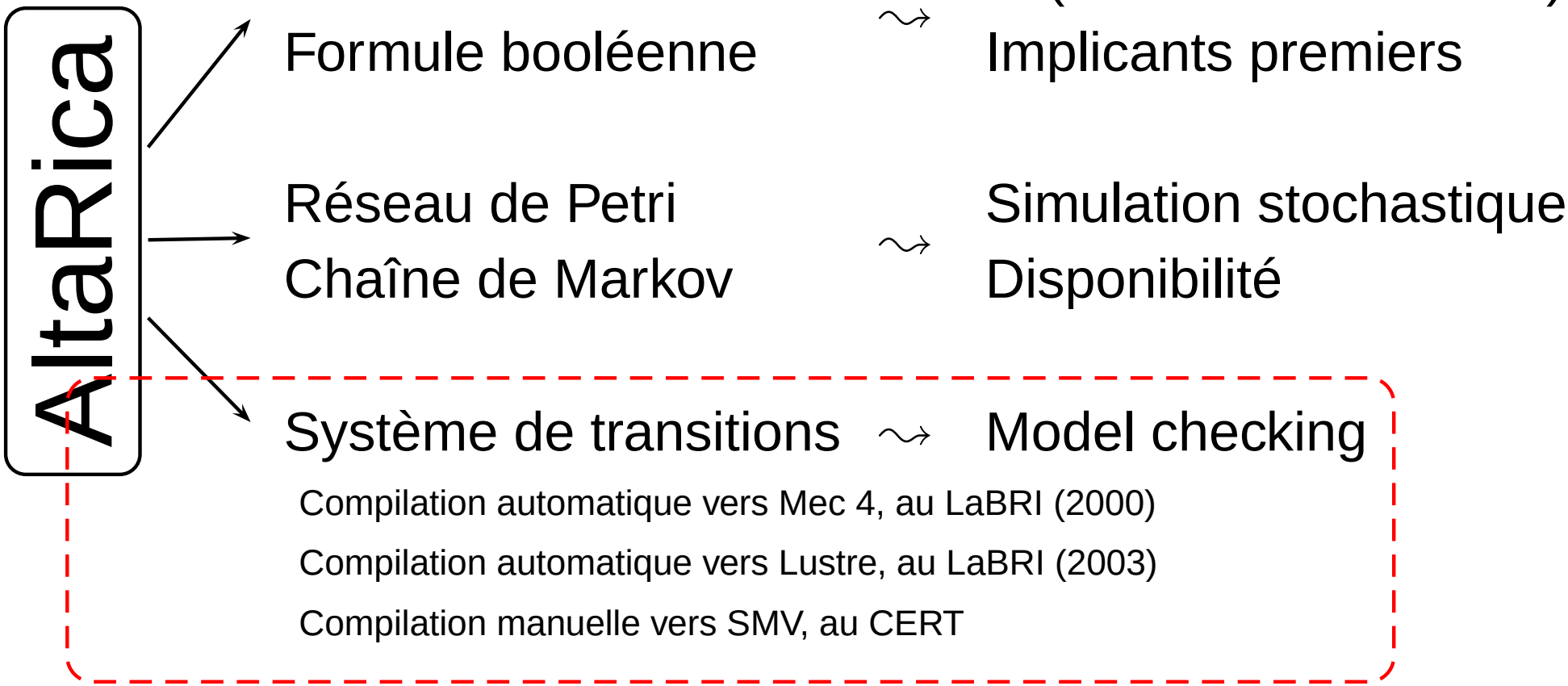
ARDF Toolbox, par ARBoost Technologies (fin 2002)



Contexte


AltaRica
Arbre de défaillances
Formule booléenne
⇝
Pr (Evénement redouté)
Implicants premiers
Réseau de Petri
Chaîne de Markov
⇝
Simulation stochastique
Disponibilité
Système de transitions
⇝
Model checking
?

Contexte


AltaRica
Arbre de défaillances
Formule booléenne
⇝
Pr (Evénement redouté)
Implicants premiers
Réseau de Petri
Chaîne de Markov
⇝
Simulation stochastique
Disponibilité
Système de transitions
⇝
Model checking
Compilation automatique vers Mec 4, au LaBRI (2000)
Compilation automatique vers Lustre, au LaBRI (2003)
Compilation manuelle vers SMV, au CERT

Contexte

Acheck, au LaBRI (2002)

Mec V, au LaBRI (2003)

Récapitulatif

- De nombreux outils “AltaRica” existent, ainsi que des bibliothèques de composants

Récapitulatif

- De nombreux outils “AltaRica” existent, ainsi que des bibliothèques de composants
- Nous souhaitons *vérifier* des modèles AltaRica

Récapitulatif

- De nombreux outils “AltaRica” existent, ainsi que des bibliothèques de composants
- Nous souhaitons *vérifier* des modèles AltaRica
- La *compilation* vers un model-checker n’est pas adaptée

Cahier des charges

- Ecrire un model-checker
- pour des modèles AltaRica
- utilisant une structure de données “symbolique” (BDDs)
- qui soit librement (ré-)utilisable

Objectifs de la thèse

- Ecrire un model-checker
- pour des modèles AltaRica
- utilisant une structure de données “symbolique” (BDDs)
- qui soit librement (ré-)utilisable

Objectifs de la thèse

- Ecrire un model-checker
- pour des modèles AltaRica
- utilisant une structure de données “symbolique” (BDDs)
- qui soit librement (ré-)utilisable
- offrant une logique de spécification expressive
- écrit de manière extensible et modulaire
- qui puisse servir de terrain d’expérimentation

Principe de la vérification

$$\text{Modèle} \overset{?}{\models} \text{propriété}$$

Principe de la vérification

Modèle décrit en AltaRica $\overset{?}{\models}$ propriété

Principe de la vérification

Modèle décrit en AltaRica $\overset{?}{\models}$ propriété écrite en μ-calcul de Park

AltaRica

Le formalisme AltaRica

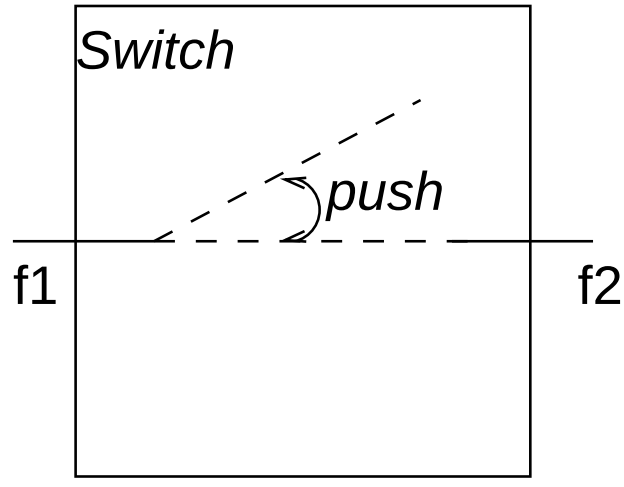
Caractéristiques : facilité de modélisation, concision

- Automates à contraintes
  - Les configurations sont des valuations de variables
  - Les transitions sont gardées par des formules
  - Les transitions mettent à jour les variables
- Modélisation hiérarchique
  - Compositionnelle
  - préserve la bisimulation

Le formalisme AltaRica (2)

- Partage de variables par assertion
  $x \leq y, \qquad x = 2$
- Synchronisation d'événements
  $\langle e, S_1.e, S_2.e \ldots, S_n.e \rangle$
- Priorités entre événements (ordre partiel)
  $e_1 < e_2, \qquad e_1 < \{e_2, e_3\} < e_4$
- Vecteurs de diffusion
  $\langle e?, S_1.e, S_2.e? \ldots, S_n.e \rangle$

Exemple : un interrupteur




```
node Switch
  state open : bool;
  flow f1, f2 : bool;
  event push;
  trans true |- push -> open := ~open;
  assert (~open) => (f1 = f2);
edon
```

Evolution du langage AltaRica

- Souhaits exprimés par les utilisateurs

Evolution du langage AltaRica

- Souhaits exprimés par les utilisateurs
- Modifications locales dans les outils
  - Priorités généralisées
  - Vecteurs de diffusion restreinte

Evolution du langage AltaRica

- Souhaits exprimés par les utilisateurs
- Modifications locales dans les outils
  - Priorités généralisées
  - Vecteurs de diffusion restreinte
- Normalisation du langage AltaRica
  - Visibilité des variables et des événements (privé, parent, public)
  - Attributs génériques (orientation des flux, contrôlabilité, observabilité, . . .)

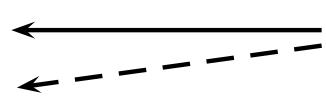
Evolution du langage AltaRica

- Souhaits exprimés par les utilisateurs
- Modifications locales dans les outils
  - Priorités généralisées
  - Vecteurs de diffusion restreinte
- Normalisation du langage AltaRica
  - Visibilité des variables et des événements (privé, parent, public)
  - Attributs génériques (orientation des flux, contrôlabilité, observabilité, . . .)
- Spécialisation en des sous-langages :
  - AltaRica DataFlow
  - AltaRica Automates Finis
  - AltaRica Temps Réel
  - AltaRica “vers Lustre”

Evolution du langage AltaRica

- Souhaits exprimés par les utilisateurs
- Modifications locales dans les outils
  - Priorités généralisées
  - Vecteurs de diffusion restreinte
- Normalisation du langage AltaRica
  - Visibilité des variables et des événements (privé, parent, public)
  - Attributs génériques (orientation des flux, contrôlabilité, observabilité, . . .)
- Spécialisation en des sous-langages :
  - AltaRica DataFlow
  - AltaRica Automates Finis ⟵ Mec V
  - AltaRica Temps Réel
  - AltaRica “vers Lustre”

Evolution du langage AltaRica

- Souhaits exprimés par les utilisateurs
- Modifications locales dans les outils
  - Priorités généralisées
  - Vecteurs de diffusion restreinte
- Normalisation du langage AltaRica
  - Visibilité des variables et des événements (privé, parent, public)
  - Attributs génériques (orientation des flux, contrôlabilité, observabilité, . . .)
- Spécialisation en des sous-langages :
  - AltaRica DataFlow
  - AltaRica Automates Finis  Mec V
  - AltaRica Temps Réel
  - AltaRica “vers Lustre”

Logiques

Logique temporelle : CTL*

- Exprime des propriétés sur des graphes
- Deux sortes de propriétés :
  - d'états

$$\varphi ::= \text{atome} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$$

  - de chemins

$$\psi ::= \neg\psi \mid \psi \vee \psi \mid \psi \wedge \psi$$

Logique temporelle : CTL*

- Exprime des propriétés sur des graphes
- Deux sortes de propriétés :
  - d'états

$$\varphi ::= \text{atome} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathbf{E}\psi \mid \mathbf{A}\psi$$

  - de chemins

$$\psi ::= \neg\psi \mid \psi \vee \psi \mid \psi \wedge \psi$$

Logique temporelle : CTL*

- Exprime des propriétés sur des graphes
- Deux sortes de propriétés :
  - d'états

$$\varphi ::= \text{atome} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathbf{E}\psi \mid \mathbf{A}\psi$$

  - de chemins

$$\psi ::= \neg\psi \mid \psi \vee \psi \mid \psi \wedge \psi \mid \mathbf{X}\psi \mid \mathbf{F}\psi \mid \mathbf{G}\psi \mid \psi\mathbf{U}\psi$$

Logique temporelle : CTL*

- Exprime des propriétés sur des graphes
- Deux sortes de propriétés :
  - d'états

$$\varphi ::= \text{atome} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathbf{E}\psi \mid \mathbf{A}\psi$$

  - de chemins

$$\psi ::= \neg\psi \mid \psi \vee \psi \mid \psi \wedge \psi \mid \mathbf{X}\psi \mid \mathbf{F}\psi \mid \mathbf{G}\psi \mid \psi\mathbf{U}\psi \mid \varphi$$

Logique temporelle : CTL*

- Exprime des propriétés sur des graphes
- Deux sortes de propriétés :
  - d'états

$$\varphi ::= \text{atome} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathbf{E}\psi \mid \mathbf{A}\psi$$

  - de chemins

$$\psi ::= \neg\psi \mid \psi \vee \psi \mid \psi \wedge \psi \mid \mathbf{X}\psi \mid \mathbf{F}\psi \mid \mathbf{G}\psi \mid \psi\mathbf{U}\psi \mid \varphi$$

- Deux restrictions sont largement employées : LTL et CTL

μ-calcul modal

- Exprime des propriétés sur des graphes
- Uniquement des propriétés d'états :

$$\varphi ::= \text{atome} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$$

μ-calcul modal

- Exprime des propriétés sur des graphes
- Uniquement des propriétés d'états :

$$\varphi ::= \text{atome} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \Diamond\varphi \mid \Box\varphi$$

μ-calcul modal

- Exprime des propriétés sur des graphes
- Uniquement des propriétés d'états :

$$\varphi ::= \text{atome} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \Diamond\varphi \mid \Box\varphi \mid X \mid \mu X.\varphi \mid \nu X.\varphi$$

μ-calcul modal

- Exprime des propriétés sur des graphes
- Uniquement des propriétés d'états :

$$\varphi ::= \text{atome} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \Diamond\varphi \mid \Box\varphi \mid X \mid \mu X.\varphi \mid \nu X.\varphi$$

- permet d'exprimer toutes les propriétés de CTL*

μ-calcul modal

- Exprime des propriétés sur des graphes
- Uniquement des propriétés d'états :

$$\varphi ::= \text{atome} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \Diamond\varphi \mid \Box\varphi \mid X \mid \mu X.\varphi \mid \nu X.\varphi$$

- permet d'exprimer toutes les propriétés de CTL*
- CTL se traduit immédiatement dans le μ-calcul

Sémantique des modalités

Pour un état s d'un système de transitions $\mathcal{A}$,

- $s \in [\![\Diamond\varphi]\!]_{\mathcal{A}}$ ssi
  il existe un successeur s' de s tel que $s' \in [\![\varphi]\!]_{\mathcal{A}}$
- $s \in [\![\Box\varphi]\!]_{\mathcal{A}}$ ssi
  pour tout successeur s' de s, on a $s' \in [\![\varphi]\!]_{\mathcal{A}}$

Sémantique des modalités

Pour un état s d'un système de transitions $\mathcal{A}$, dont la relation de transition est T,

- $s \in [\![\Diamond\varphi]\!]_{\mathcal{A}}$ ssi
  il existe un successeur s' de s tel que $s' \in [\![\varphi]\!]_{\mathcal{A}}$
- $s \in [\![\Box\varphi]\!]_{\mathcal{A}}$ ssi
  pour tout successeur s' de s, on a $s' \in [\![\varphi]\!]_{\mathcal{A}}$

Sémantique des modalités

Pour un état s d'un système de transitions $\mathcal{A}$, dont la relation de transition est T,

- $s \in [\![\Diamond\varphi]\!]_{\mathcal{A}}$ ssi $\exists s' \, (T(s,s') \wedge s' \in [\![\varphi]\!]_{\mathcal{A}})$
  il existe un successeur s' de s tel que $s' \in [\![\varphi]\!]_{\mathcal{A}}$

- $s \in [\![\Box\varphi]\!]_{\mathcal{A}}$ ssi
  pour tout successeur s' de s, on a $s' \in [\![\varphi]\!]_{\mathcal{A}}$

Sémantique des modalités

Pour un état s d'un système de transitions $\mathcal{A}$, dont la relation de transition est T,

- $s \in [\![\Diamond\varphi]\!]_{\mathcal{A}}$ ssi $\exists s' \, (T(s,s') \wedge s' \in [\![\varphi]\!]_{\mathcal{A}})$
  il existe un successeur s' de s tel que $s' \in [\![\varphi]\!]_{\mathcal{A}}$
- $s \in [\![\Box\varphi]\!]_{\mathcal{A}}$ ssi $\forall s' \, (T(s,s') \Rightarrow s' \in [\![\varphi]\!]_{\mathcal{A}})$
  pour tout successeur s' de s, on a $s' \in [\![\varphi]\!]_{\mathcal{A}}$

Sémantique des modalités

Pour un état s d'un système de transitions $\mathcal{A}$, dont la relation de transition est T,

- $s \in [\![\Diamond\varphi]\!]_{\mathcal{A}}$ ssi $\exists s' \, (T(s, s') \wedge s' \in [\![\varphi]\!]_{\mathcal{A}})$

- $s \in [\![\Box\varphi]\!]_{\mathcal{A}}$ ssi $\forall s' \, (T(s, s') \Rightarrow s' \in [\![\varphi]\!]_{\mathcal{A}})$

Sémantique des modalités

Pour un état s d'un système de transitions $\mathcal{A}$, dont la relation de transition est T,

- $s \in [\![\Diamond\varphi]\!]_{\mathcal{A}}$ ssi $\exists s' \, (T(s, s') \wedge s' \in [\![\varphi]\!]_{\mathcal{A}})$
- $s \in [\![\Box\varphi]\!]_{\mathcal{A}}$ ssi $\forall s' \, (T(s, s') \Rightarrow s' \in [\![\varphi]\!]_{\mathcal{A}})$
- Il est donc naturel d'introduire dans le μ-calcul :

Sémantique des modalités

Pour un état s d'un système de transitions $\mathcal{A}$, dont la relation de transition est T,

- $s \in [\![\Diamond\varphi]\!]_{\mathcal{A}}$ ssi $\exists s' \, (T(s, s') \wedge s' \in [\![\varphi]\!]_{\mathcal{A}})$
- $s \in [\![\Box\varphi]\!]_{\mathcal{A}}$ ssi $\forall s' \, (T(s, s') \Rightarrow s' \in [\![\varphi]\!]_{\mathcal{A}})$
- Il est donc naturel d'introduire dans le μ-calcul :
  - Les quantificateurs du premier ordre

Sémantique des modalités

Pour un état s d'un système de transitions $\mathcal{A}$, dont la relation de transition est T,

- $s \in [\![\Diamond\varphi]\!]_{\mathcal{A}}$ ssi $\exists s' \, (T(s, s') \wedge s' \in [\![\varphi]\!]_{\mathcal{A}})$

- $s \in [\![\Box\varphi]\!]_{\mathcal{A}}$ ssi $\forall s' \, (T(s, s') \Rightarrow s' \in [\![\varphi]\!]_{\mathcal{A}})$

- Il est donc naturel d'introduire dans le μ-calcul :
  - Les quantificateurs du premier ordre
  - Les relations

Sémantique des modalités

Pour un état s d'un système de transitions $\mathcal{A}$, dont la relation de transition est T,

- $s \in [\![\Diamond\varphi]\!]_{\mathcal{A}}$ ssi $\exists s' \, (T(s, s') \wedge s' \in [\![\varphi]\!]_{\mathcal{A}})$

- $s \in [\![\Box\varphi]\!]_{\mathcal{A}}$ ssi $\forall s' \, (T(s, s') \Rightarrow s' \in [\![\varphi]\!]_{\mathcal{A}})$

- Il est donc naturel d'introduire dans le μ-calcul :
  - Les quantificateurs du premier ordre
  - Les relations

  L'outil Toupie (Antoine Rauzy, 1993) l'implémente

μ-calcul de Park

En notant $x_0\ x_1\ \ldots$ les variables du premier ordre, et
$X_0\ X_1\ \ldots$ les variables de relations,

- Formules bien formées :

$$\varphi ::= \top \mid \bot \mid x_i = x_j \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid$$
$$\exists x_i\, \varphi \mid \forall x_i\, \varphi \mid \psi(x_{i0}, \ldots, x_{in})$$

- Termes relationnels :

$$\psi ::= X_i \mid \mu X_i.\psi \mid \nu X_i.\psi \mid$$
$$\lambda y_1 \ldots y_n.\varphi$$

Exemple : la clôture transitive d'une relation R

$$\mu T.\lambda xy.\,[R(x,y) \vee \exists z\,(R(x,z) \wedge T(z,y))]$$

Langage de spécification de Mec V

Langage de spécification de Mec V

- Opérateurs logiques usuels : `~p`, `p & q`, `p | q`, ...
- Opérations arithmétiques usuelles : `a + b`, `a = b`, ...
- Quantificateurs du premier ordre : `<x>p`, `[x]p`
- Définition de relations : `R(x, y) := p;`
- Définition de relations par points fixes :
  - `R(x, y) += p;`
  - `R(x, y) -= p;`

Exemple : la clôture transitive d'une relation `R`

```
T(x, y) += R(x, y) | <z>(R(x, z) & T(z, y));
```

Intégration avec AltaRica

- Mêmes types de base
  - booléen
  - intervalle fini d’entiers
  - énumération

Intégration avec AltaRica

- Mêmes types de base
  - booléen
  - intervalle fini d'entiers
  - énumération

Pour chaque nœud AltaRica A, sont ajoutés :

Intégration avec AltaRica

- Mêmes types de base
  - booléen
  - intervalle fini d'entiers
  - énumération

  Pour chaque nœud AltaRica A, sont ajoutés :
- Deux types
  - le type de ses configurations A!c
  - le type de ses vecteurs d'événements A!ev

Intégration avec AltaRica

- Mêmes types de base
  - booléen
  - intervalle fini d'entiers
  - énumération

  Pour chaque nœud AltaRica `A`, sont ajoutés :

- Deux types
  - le type de ses configurations `A!c`
  - le type de ses vecteurs d'événements `A!ev`
- Deux relations
  - sa relation de transition `A!t` ($\subseteq$ `A!c`$\times$`A!ev`$\times$`A!c`)
  - son ensemble d'états initiaux `A!init` ($\subseteq$ `A!c`)

Exemple

Calcul des configurations accessibles d'un nœud `A` :

```
Acc(s) += A!init(s) |
          <s'><e>(A!t(s', e, s) & Acc(s'));
```

définit la relation unaire (l'ensemble) `Acc` comme l'ensemble minimum des configurations qui sont initiales, ou qui ont un prédecesseur qui est lui-même accessible

Points fixes imbriqués

- Mec V permet de résoudre des *systèmes* d'équations
- Cela lui donne le même pouvoir d'expression que le μ-calcul
- Permet d'exprimer les propriétés d'équité, par exemple :

Toute exécution passera infiniment souvent par Repetes

Points fixes imbriqués

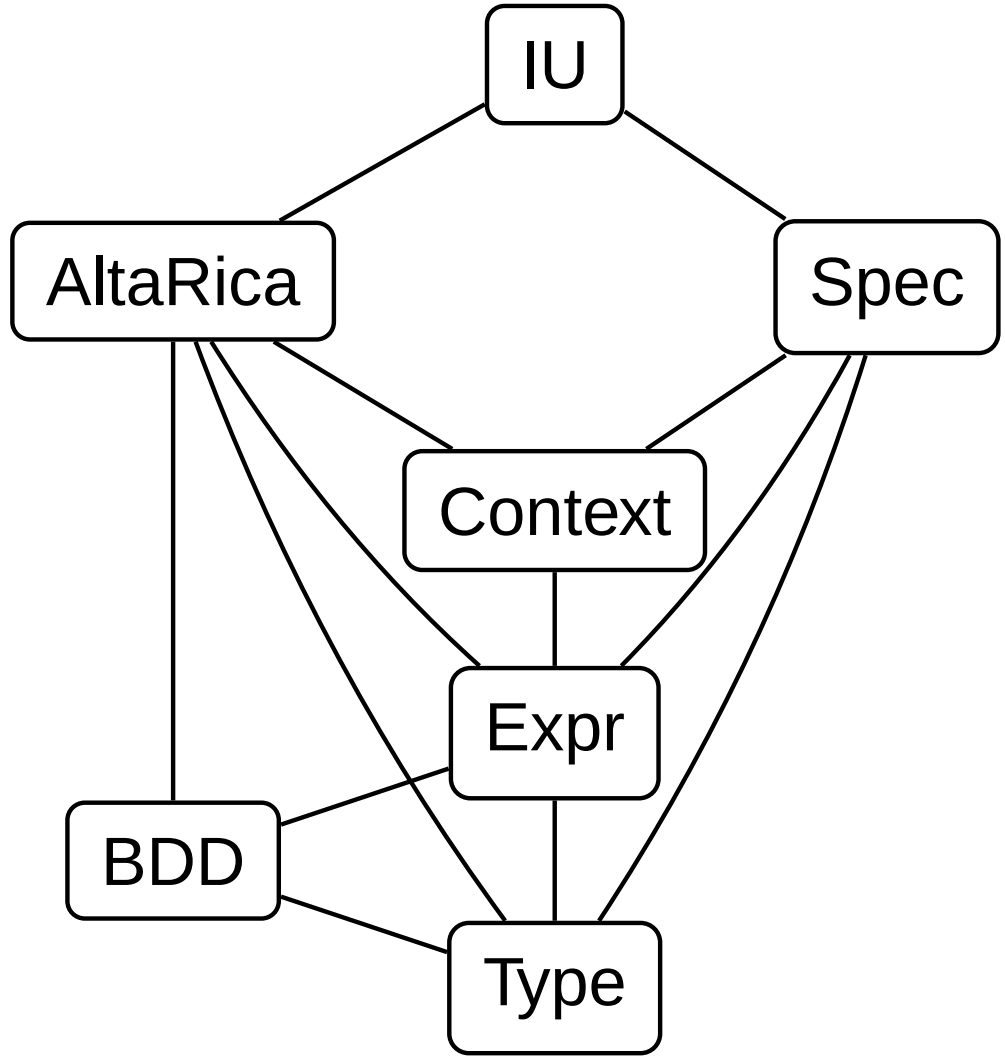
- Mec V permet de résoudre des *systèmes* d'équations
- Cela lui donne le même pouvoir d'expression que le μ-calcul
- Permet d'exprimer les propriétés d'équité, par exemple :

Toute exécution passera infiniment souvent par Repetes

```
begin
        PredInf(u) -1=   Pred(u);
 local    Pred(u) +2=   <v>(T(u, v) & Repetes(v) & PredInf(v)) |
                        <v>(T(u, v) & Pred(v));
end
```

Implémentation de Mec V

Architecture de Mec V



Diagrammes de Décisions Binaires

Un BDD représente une fonction booléenne de manière :

- compacte
- canonique

Les opérations suivantes sont efficaces sur les BDDs :

- Le complémentaire, l'union, l'intersection
- La projection, la cylindrification
- La substitution
- Le test d'égalité est immédiat

Ils sont un choix naturel pour implémenter Mec V

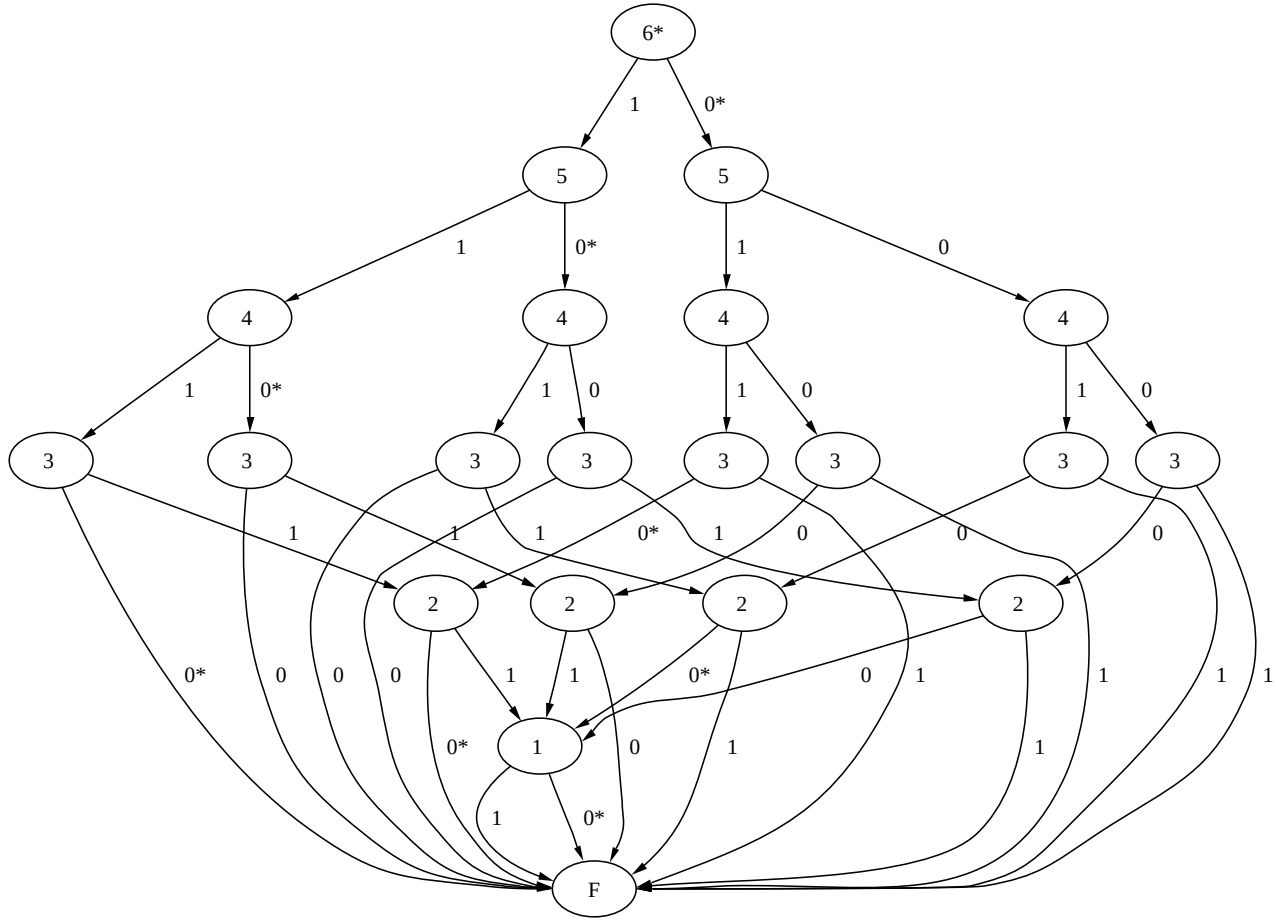
Sensibilité à l'ordre des variables (1)

La taille d'un BDD dépend *fortement* de l'ordre des variables

Codage de la relation à 6 variables $x_1x_2x_3 = y_1y_2y_3$ avec les ordres

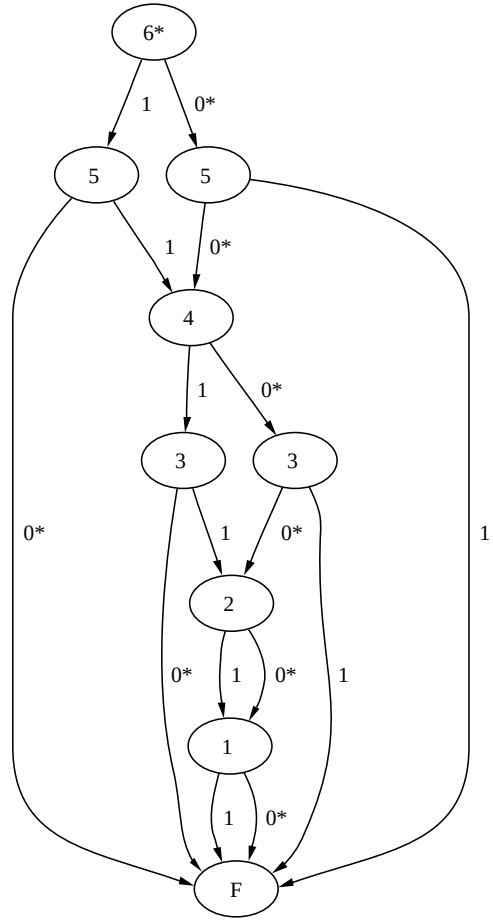
- $x_1 \leq x_2 \leq x_3 \leq y_1 \leq y_2 \leq y_3$
- $x_1 \leq y_1 \leq x_2 \leq y_2 \leq x_3 \leq y_3$

Sensibilité à l'ordre des variables (2)



$$x_1 \leq x_2 \leq x_3 \leq y_1 \leq y_2 \leq y_3$$

Sensibilité à l'ordre des variables (3)



$$x_1 \leq y_1 \leq x_2 \leq y_2 \leq x_3 \leq y_3$$

Choix d'implémentation pour les BDDs

- Les BDDs utilisent l'ordre d'entrelacement entre composantes d'une relation
- Les BDDs contiennent des arcs négatifs
  - Permet de calculer la négation en temps constant
  - Peut réduire la taille d'un BDD d'un facteur 2
- Les variables sur domaine fini sont représentées par des *vecteurs* de BDDs
  - Notre module peut traiter les nombres négatifs
- Les nœuds sont récupérés par un ramasse-miettes qui fonctionne par "marquage et nettoyage"
- Les variables toujours égales sont fusionnées

Calcul de la sémantique d'un nœud

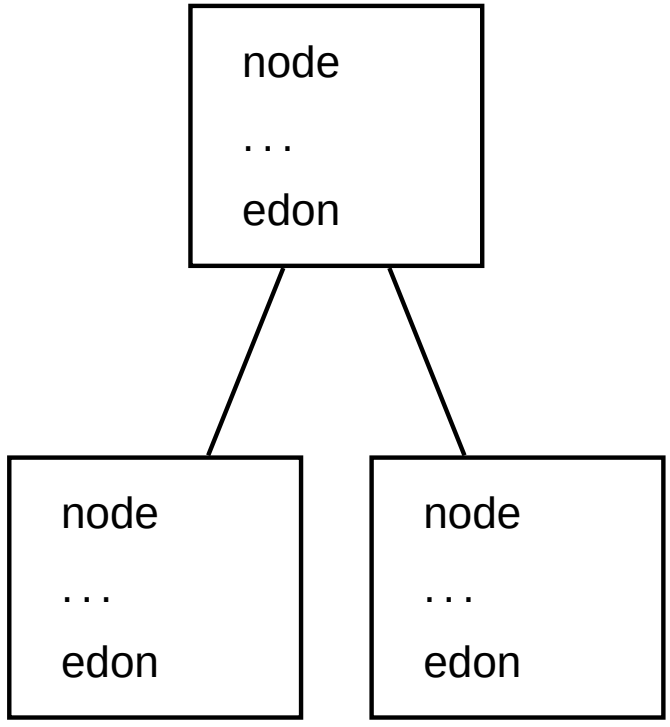
- La sémantique d'un nœud est un système de transitions

Calcul de la sémantique d'un nœud

- La sémantique d'un nœud est un système de transitions
- Les outils existants utilisent la "mise à plat"

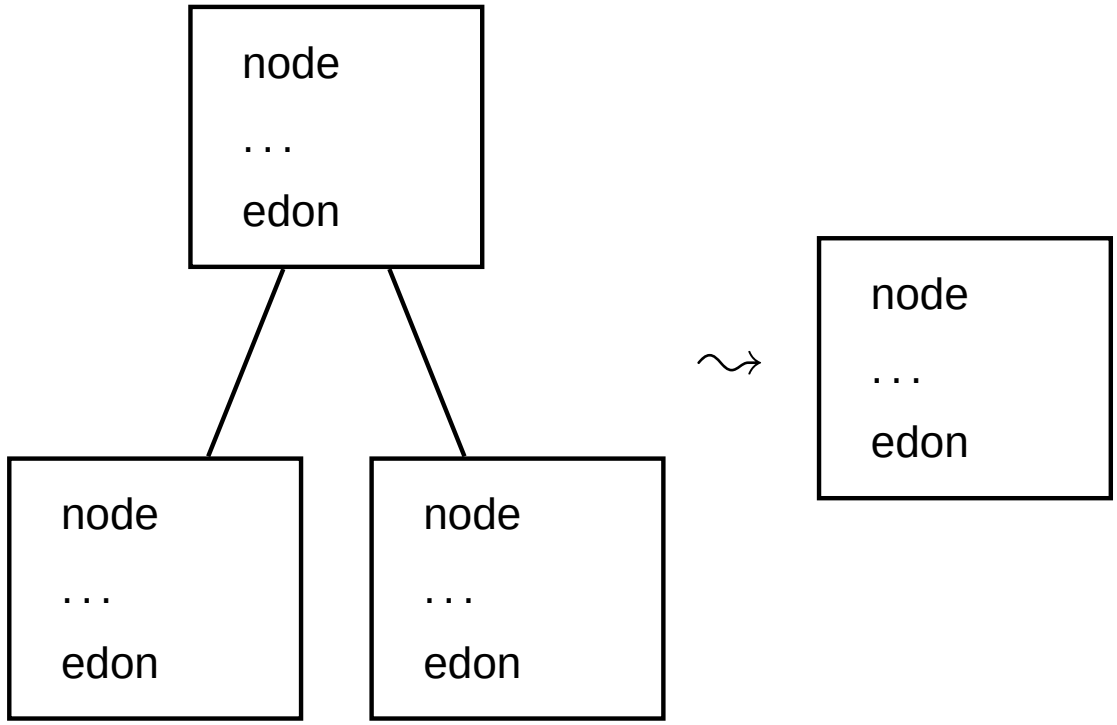
Calcul de la sémantique d'un nœud

- La sémantique d'un nœud est un système de transitions
- Les outils existants utilisent la "mise à plat"



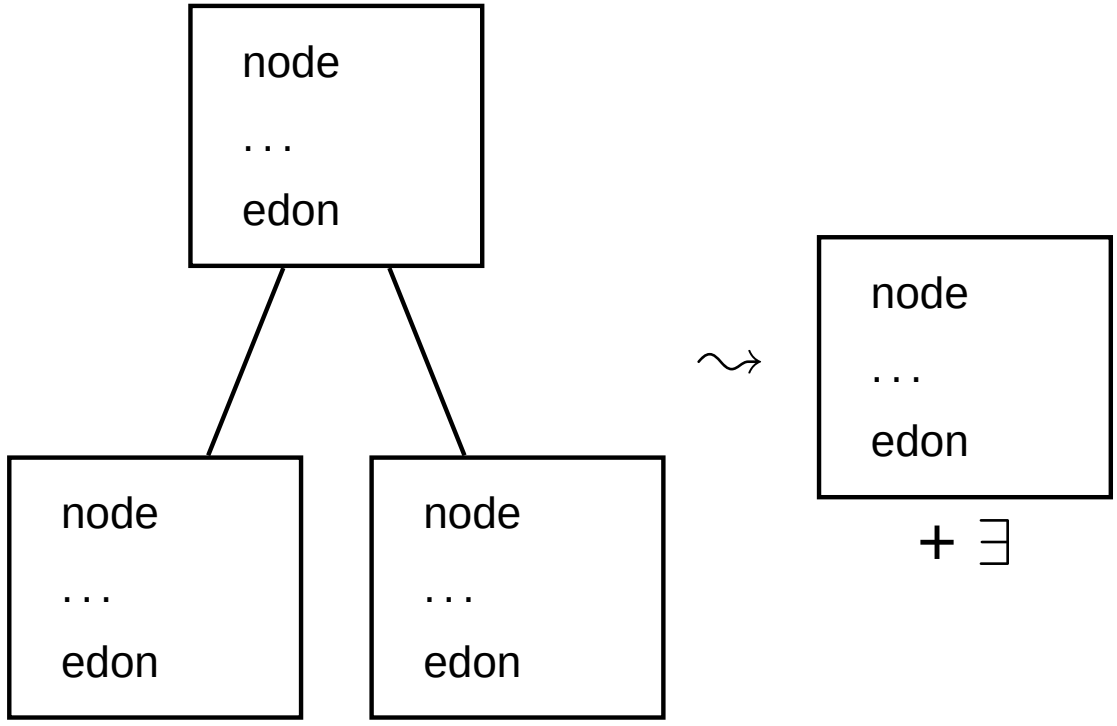
Calcul de la sémantique d'un nœud

- La sémantique d'un nœud est un système de transitions
- Les outils existants utilisent la "mise à plat"



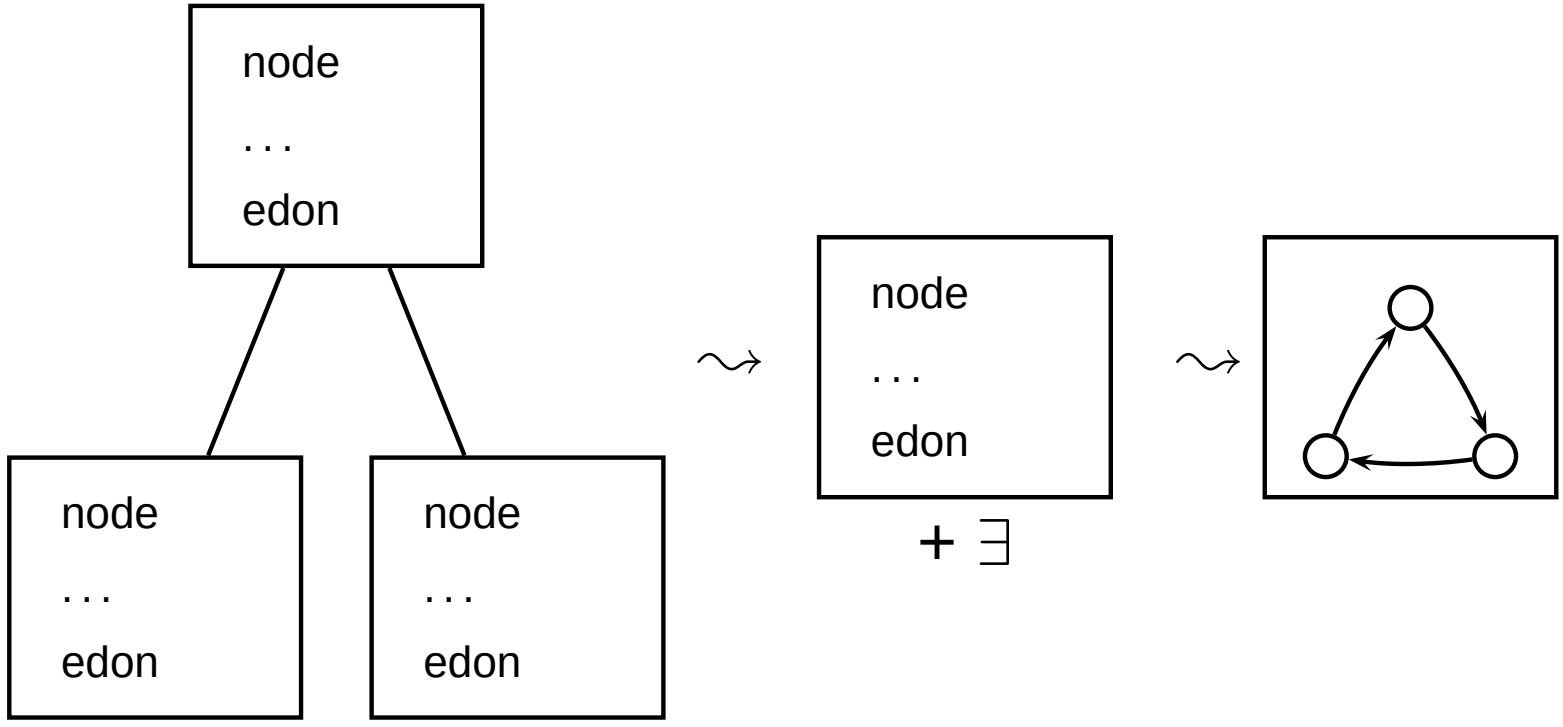
Calcul de la sémantique d'un nœud

- La sémantique d'un nœud est un système de transitions
- Les outils existants utilisent la "mise à plat"



Calcul de la sémantique d'un nœud

- La sémantique d'un nœud est un système de transitions
- Les outils existants utilisent la "mise à plat"



Calcul de la sémantique d'un nœud

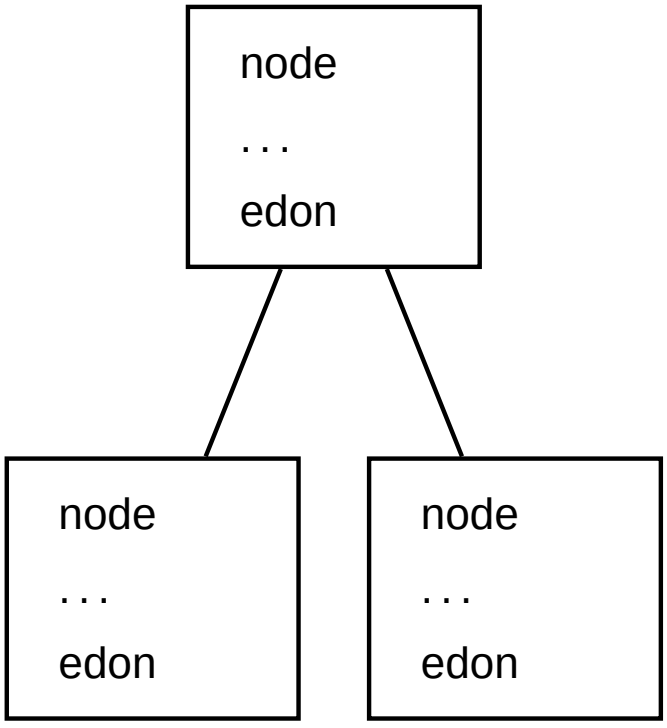
- La sémantique d'un nœud est un système de transitions
- Les outils existants utilisent la "mise à plat"

Calcul de la sémantique d'un nœud

- La sémantique d'un nœud est un système de transitions
- Les outils existants utilisent la "mise à plat"
- Mec V calcule la sémantique de tous les nœuds

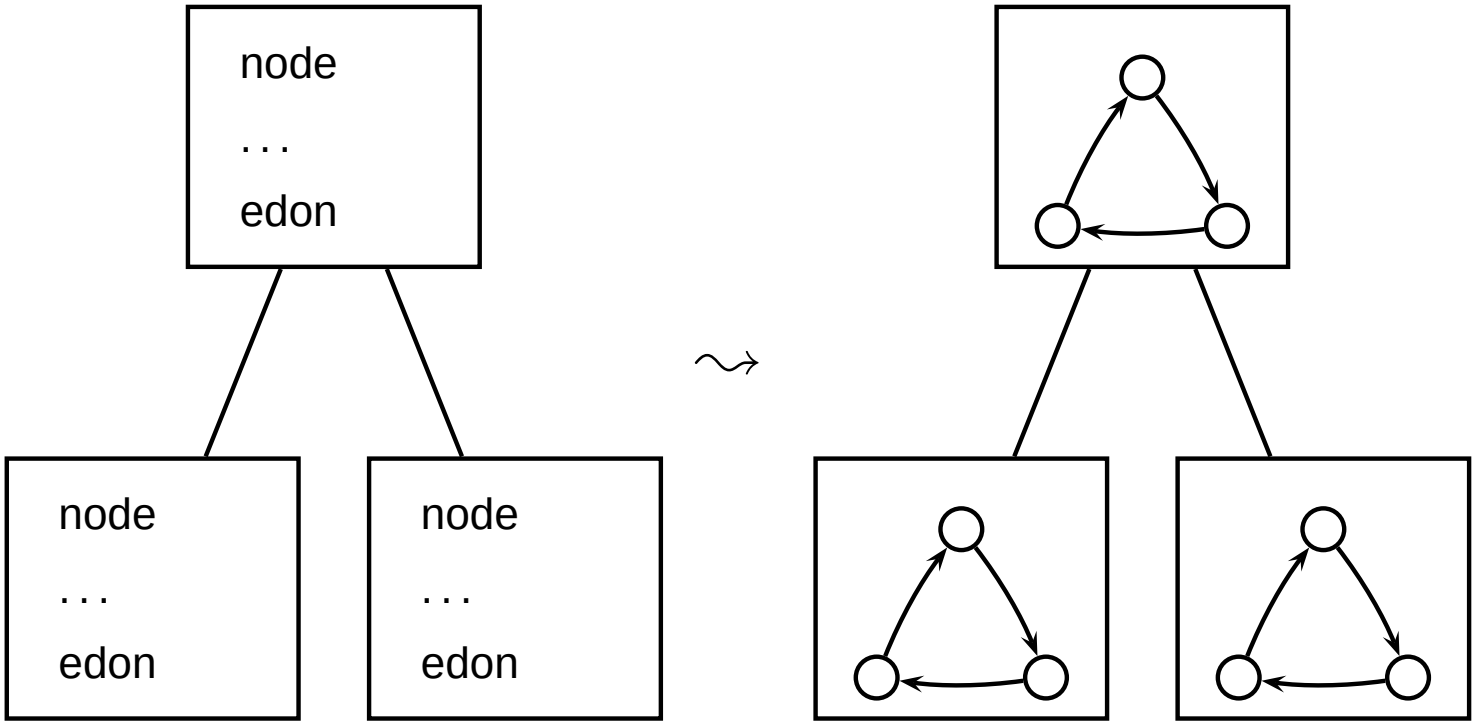
Calcul de la sémantique d'un nœud

- La sémantique d'un nœud est un système de transitions
- Les outils existants utilisent la "mise à plat"
- Mec V calcule la sémantique de tous les nœuds



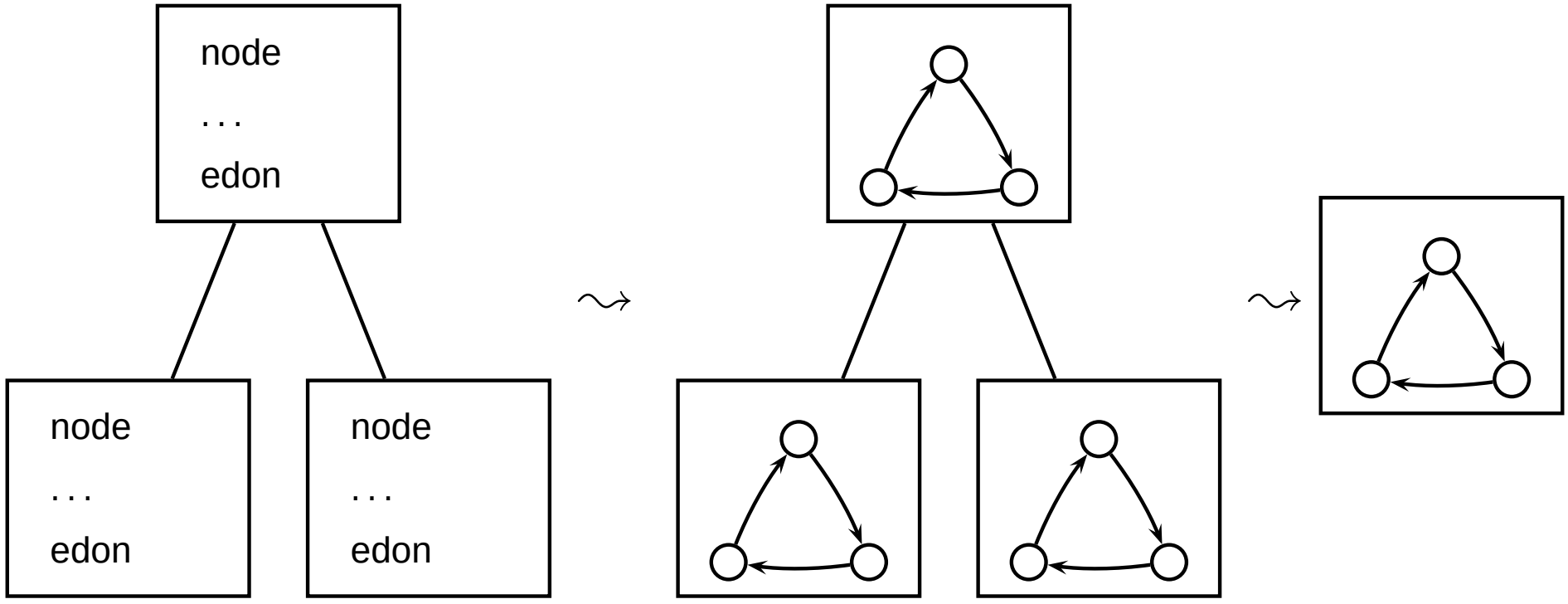
Calcul de la sémantique d'un nœud

- La sémantique d'un nœud est un système de transitions
- Les outils existants utilisent la "mise à plat"
- Mec V calcule la sémantique de tous les nœuds



Calcul de la sémantique d'un nœud

- La sémantique d'un nœud est un système de transitions
- Les outils existants utilisent la "mise à plat"
- Mec V calcule la sémantique de tous les nœuds



Implémentation de Mec V

- Interpréteur de commandes interactif
- Inférence de types simple

Implémentation de Mec V

- Interpréteur de commandes interactif
- Inférence de types simple
- Ecrit en langage C ANSI
  - Conception modulaire
  - Utilisation d'exceptions
  - Portable
- Mis dans le domaine public
- Code source utilisé en enseignement

Implémentation de Mec V

- Interpréteur de commandes interactif
- Inférence de types simple
- Ecrit en langage C ANSI
  - Conception modulaire
  - Utilisation d'exceptions
  - Portable
- Mis dans le domaine public
- Code source utilisé en enseignement

- A. Griffault, A. Vincent, *Vérification de modèles AltaRica*, communication à MAJECSTIC 2003

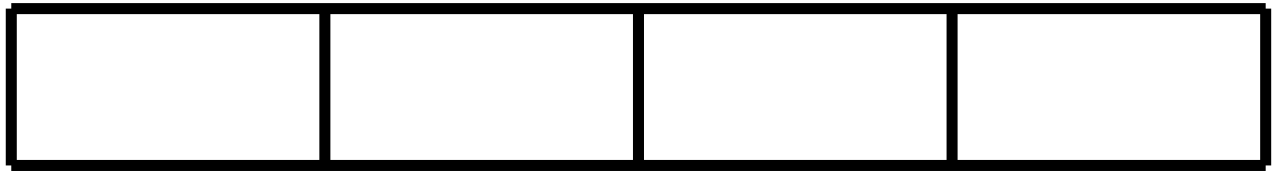
Exemple d’utilisation de Mec V

FIFO par buffer circulaire

```
node
      flow   vide, plein : bool;
             top : DONNEE;
      event  put, get;
```

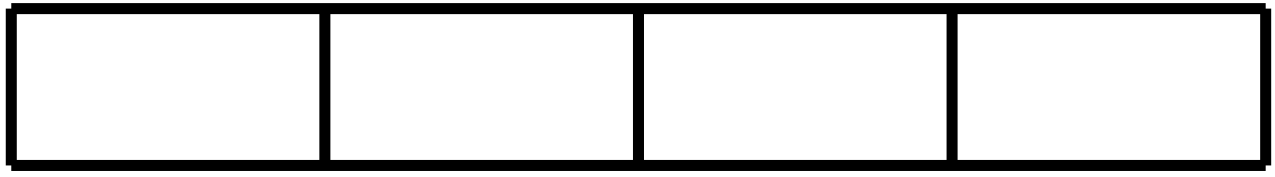
FIFO par buffer circulaire

```
node
      flow    vide, plein : bool;
              top : DONNEE;
      event   put, get;
      sub     B0, B1, B2, B3 : Buffer;
```



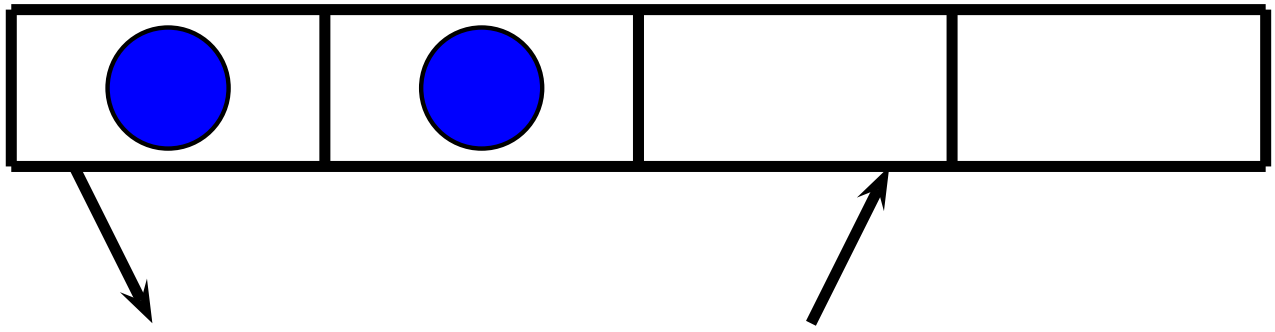
FIFO par buffer circulaire

```
node
      flow    vide, plein : bool;
              top : DONNEE;
      event   put, get;
      sub     B0, B1, B2, B3 : Buffer;
      state   depot, retrait : [0, 3];
```



FIFO par buffer circulaire

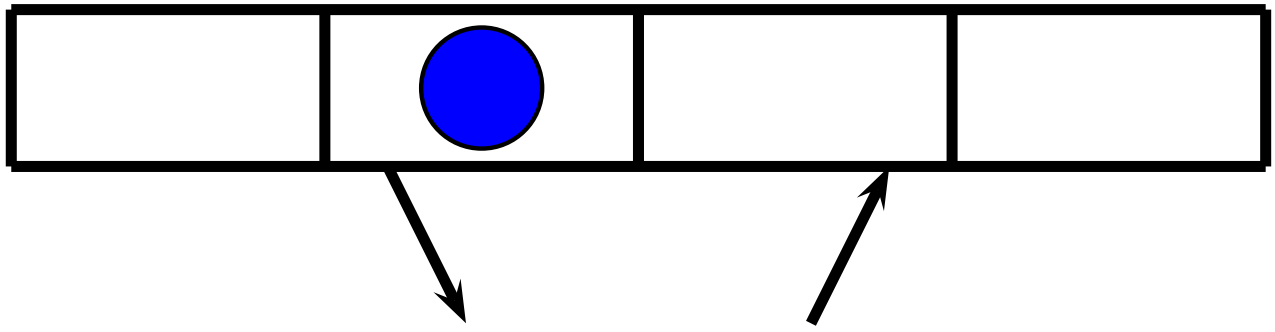
```
node
        flow    vide, plein : bool;
                top : DONNEE;
        event   put, get;
        sub     B0, B1, B2, B3 : Buffer;
        state   depot, retrait : [0, 3];
```



get,

FIFO par buffer circulaire

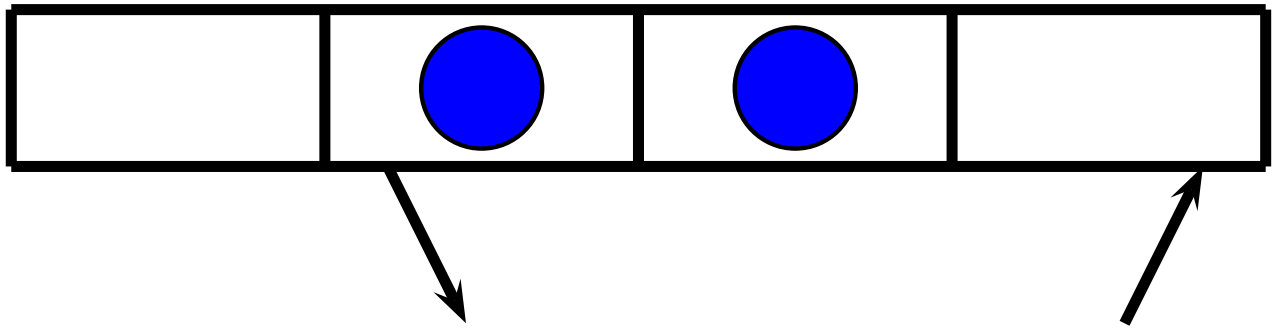
```
node
        flow    vide, plein : bool;
                top : DONNEE;
        event   put, get;
        sub     B0, B1, B2, B3 : Buffer;
        state   depot, retrait : [0, 3];
```



get, put,

FIFO par buffer circulaire

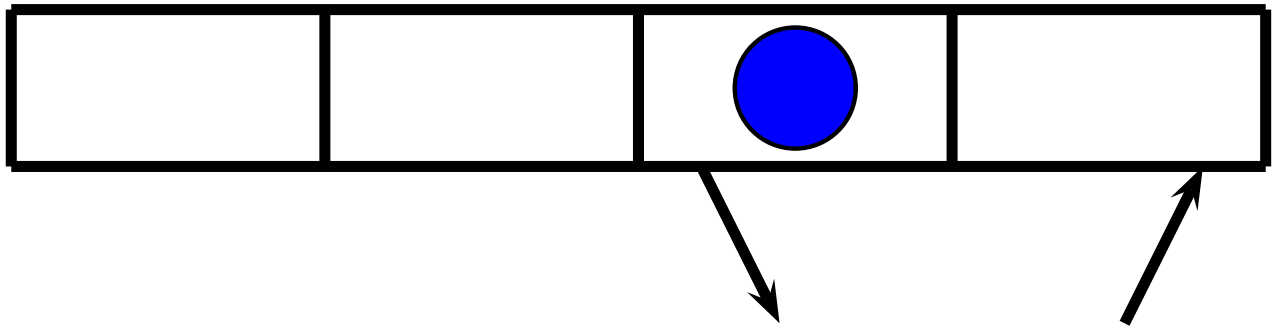
```
node
        flow    vide, plein : bool;
                top : DONNEE;
        event   put, get;
        sub     B0, B1, B2, B3 : Buffer;
        state   depot, retrait : [0, 3];
```



get, put, get,

FIFO par buffer circulaire

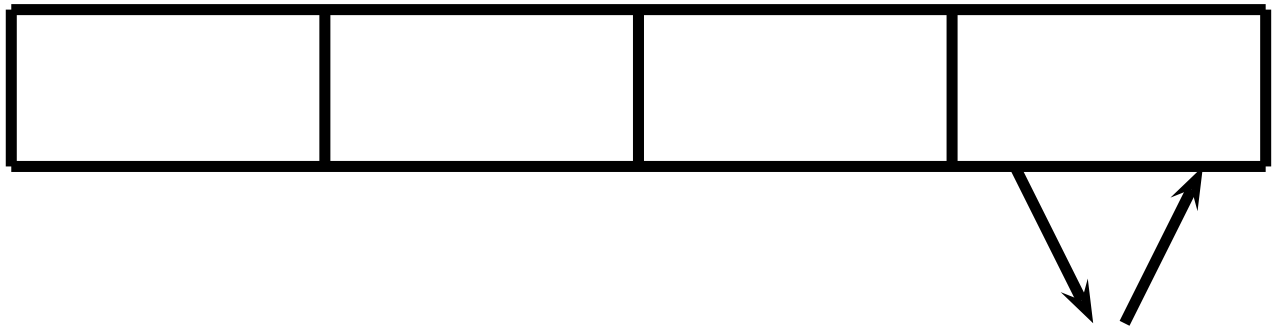
```
node
        flow    vide, plein : bool;
                top : DONNEE;
        event   put, get;
        sub     B0, B1, B2, B3 : Buffer;
        state   depot, retrait : [0, 3];
```



get, put, get, get,

FIFO par buffer circulaire

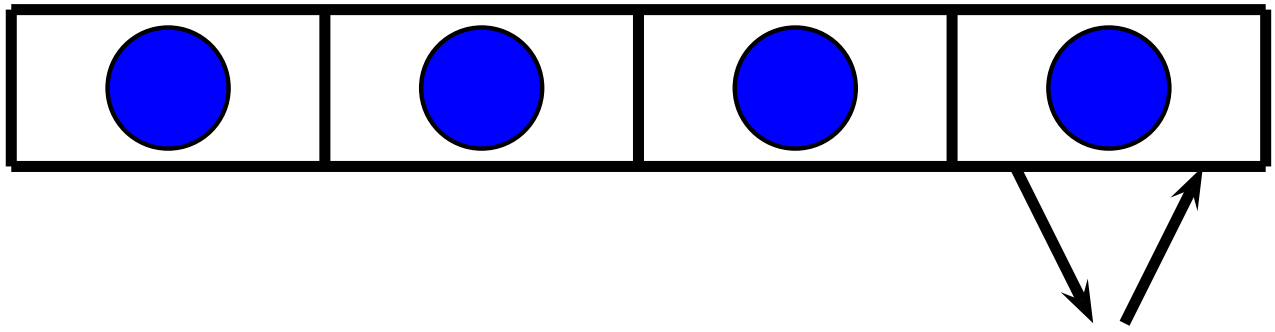
```
node
        flow    vide, plein : bool;
                top : DONNEE;
        event   put, get;
        sub     B0, B1, B2, B3 : Buffer;
        state   depot, retrait : [0, 3];
```



get, put, get, get

FIFO par buffer circulaire

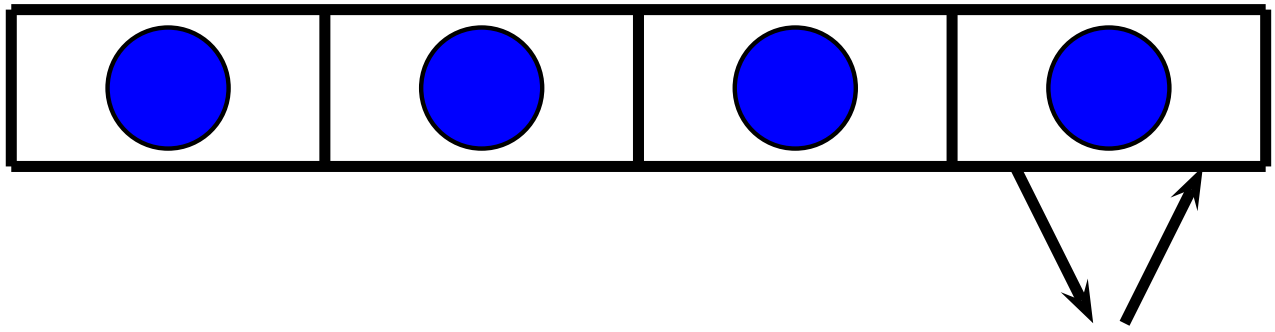
```
node
        flow    vide, plein : bool;
                top : DONNEE;
        event   put, get;
        sub     B0, B1, B2, B3 : Buffer;
        state   depot, retrait : [0, 3];
```



get, put, get, get

FIFO par buffer circulaire

```
node
        flow    vide, plein : bool;
                top : DONNEE;
        event   put, get;
        sub     B0, B1, B2, B3 : Buffer;
        state   depot, retrait : [0, 3];
                svide, splein : bool;
```



get, put, get, get

FIFO par buffer circulaire

```
node
      flow   vide, plein : bool;
             top : DONNEE;
      event  put, get;
      sub    B0, B1, B2, B3 : Buffer;
      state  depot, retrait : [0, 3];
             svide, splein : bool;
      trans  ~plein |- put ->
       depot  := if (depot+1<4) then depot+1 else 0,
       svide  := false,
       splein := if (depot+1<4) then (depot+1)=retrait else 0=retrait;
```

FIFO par buffer circulaire

```
node
      flow   vide, plein : bool;
             top : DONNEE;
      event  put, get;
      sub    B0, B1, B2, B3 : Buffer;
      state  depot, retrait : [0, 3];
             svide, splein : bool;
      trans  ~plein |- put ->
       depot  := if (depot+1<4) then depot+1 else 0,
       svide  := false,
       splein := if (depot+1<4) then (depot+1)=retrait else 0=retrait;
      ...
```

FIFO avec sentinelle

```
node
        flow    vide, plein : bool;
                top : DONNEE;
        event   put, get;
```

FIFO avec sentinelle

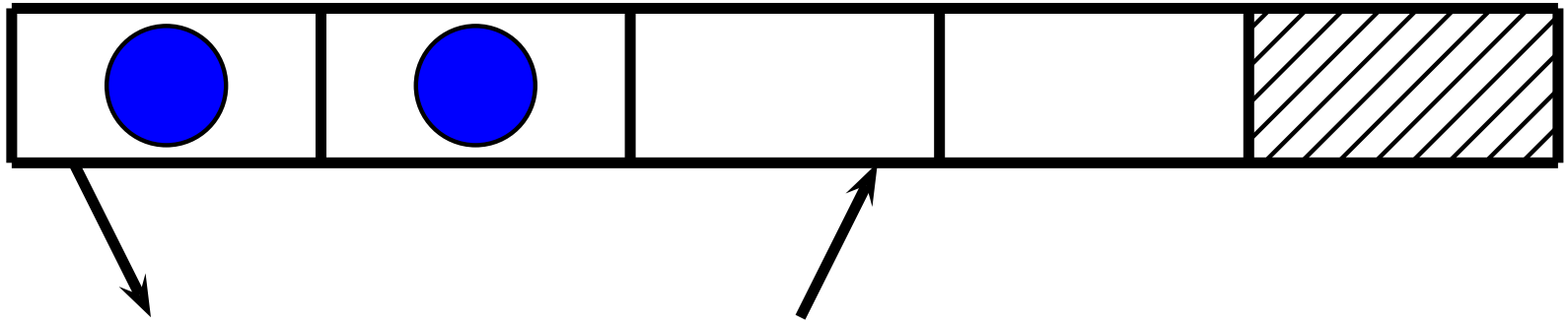
```
node
      flow   vide, plein : bool;
             top : DONNEE;
      event  put, get;
      sub    B0, B1, B2, B3, B4 : Buffer;
```

FIFO avec sentinelle

```
node
      flow    vide, plein : bool;
              top : DONNEE;
      event   put, get;
      sub     B0, B1, B2, B3, B4 : Buffer;
      state   depot, retrait : [0, 4];
```

FIFO avec sentinelle

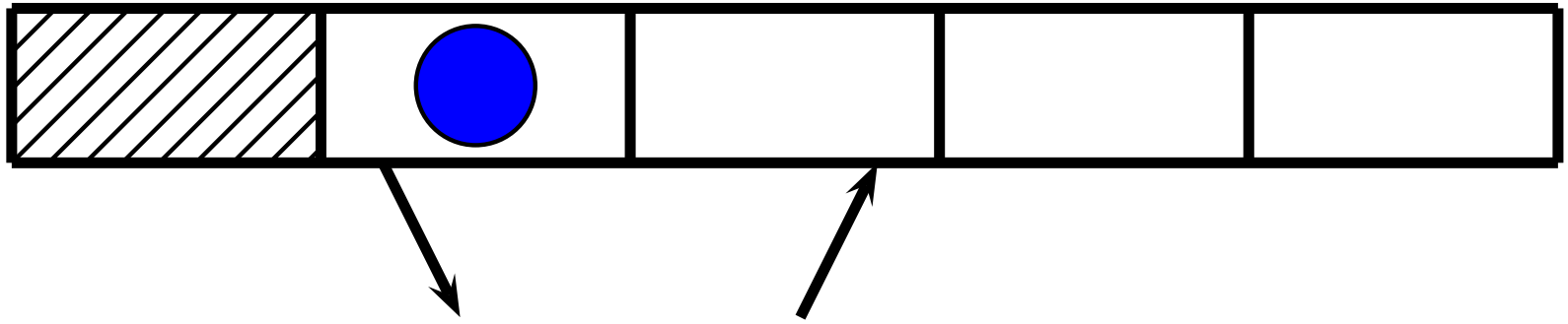
```
node
      flow    vide, plein : bool;
              top : DONNEE;
      event   put, get;
      sub     B0, B1, B2, B3, B4 : Buffer;
      state   depot, retrait : [0, 4];
```



get,

FIFO avec sentinelle

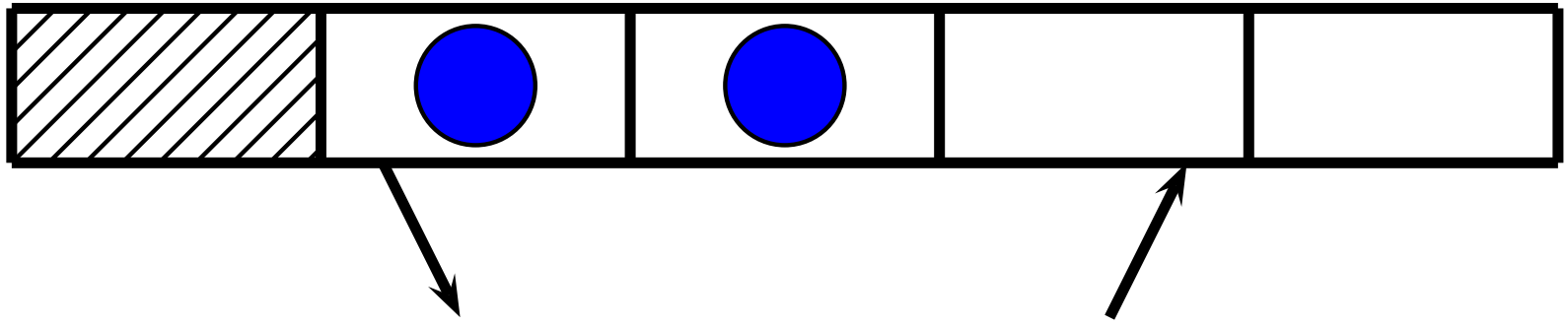
```
node
        flow    vide, plein : bool;
                top : DONNEE;
        event   put, get;
        sub     B0, B1, B2, B3, B4 : Buffer;
        state   depot, retrait : [0, 4];
```



get, put,

FIFO avec sentinelle

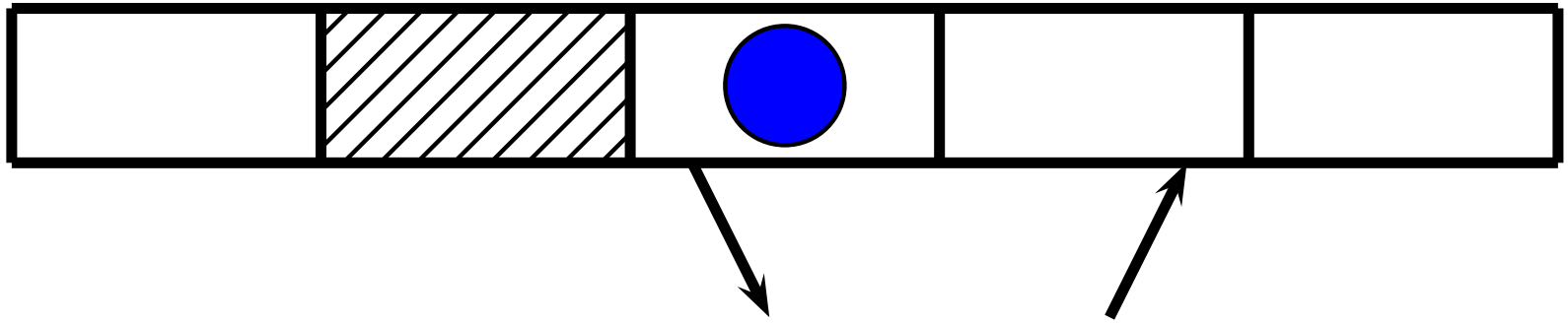
```
node
        flow    vide, plein : bool;
                top : DONNEE;
        event   put, get;
        sub     B0, B1, B2, B3, B4 : Buffer;
        state   depot, retrait : [0, 4];
```



get, put, get,

FIFO avec sentinelle

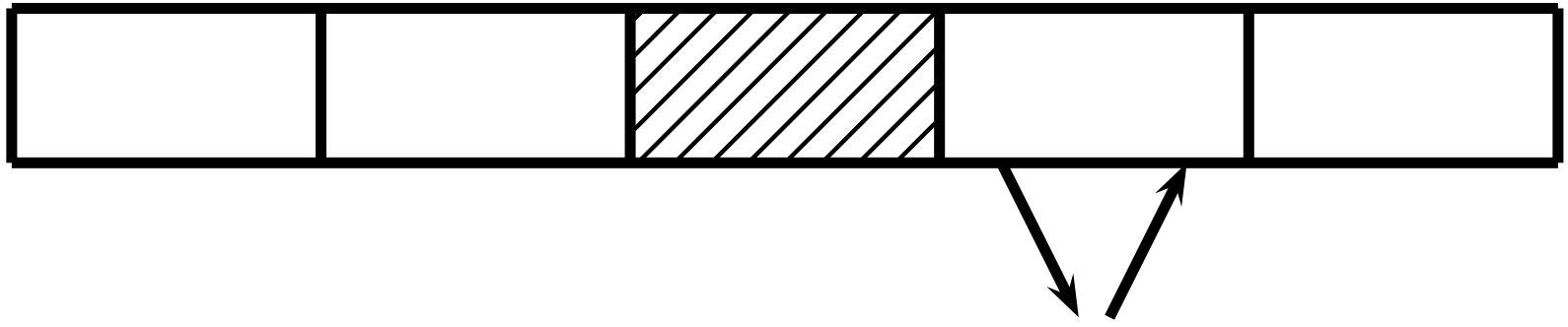
```
node
        flow    vide, plein : bool;
                top : DONNEE;
        event   put, get;
        sub     B0, B1, B2, B3, B4 : Buffer;
        state   depot, retrait : [0, 4];
```



get, put, get, get,

FIFO avec sentinelle

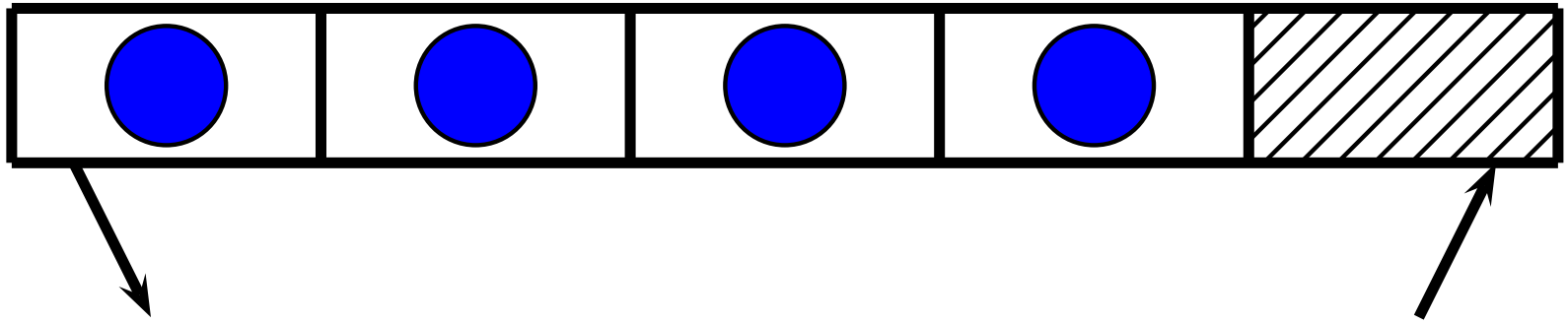
```
node
        flow    vide, plein : bool;
                top : DONNEE;
        event   put, get;
        sub     B0, B1, B2, B3, B4 : Buffer;
        state   depot, retrait : [0, 4];
```



get, put, get, get

FIFO avec sentinelle

```
node
        flow    vide, plein : bool;
                top : DONNEE;
        event   put, get;
        sub     B0, B1, B2, B3, B4 : Buffer;
        state   depot, retrait : [0, 4];
```



get, put, get, get

FIFO avec sentinelle

```
node
        flow    vide, plein : bool;
                top : DONNEE;
        event   put, get;
        sub     B0, B1, B2, B3, B4 : Buffer;
        state   depot, retrait : [0, 4];
        trans
            ~plein |- put -> depot := if (depot<4) then depot+1 else 0;
```

FIFO avec sentinelle

```
node
      flow   vide, plein : bool;
             top : DONNEE;
      event  put, get;
      sub    B0, B1, B2, B3, B4 : Buffer;
      state  depot, retrait : [0, 4];
      trans
          ~plein |- put -> depot := if (depot<4) then depot+1 else 0;
          ...
```

FIFO avec sentinelle

```
node
        flow    vide, plein : bool;
                top : DONNEE;
        event   put, get;
        sub     B0, B1, B2, B3, B4 : Buffer;
        state   depot, retrait : [0, 4];
        trans
            ~plein |- put -> depot := if (depot<4) then depot+1 else 0;
            ...
        assert
            vide = (depot = retrait);
            plein = if depot<4 then depot+1=retrait else 0=retrait;
            ...
```

Equivalence d’interface

- entre configurations

```
eqC(s:FifoC!c, s’:FifoS!c) :=
    (s.plein = s’.plein) &
    (s.vide = s’.vide) &
    (s.top = s’.top);
```

Equivalence d’interface

- entre configurations

```
eqC(s:FifoC!c, s’:FifoS!c) :=
    (s.plein = s’.plein) &
    (s.vide = s’.vide) &
    (s.top = s’.top);
```

- entre événements

```
eqEv(e:FifoC!ev, e’:FifoS!ev) :=
    (e. = put & e’. = put) |
    (e. = get & e’. = get) |
    (e. = "" & e’. = "");
```

Equivalence d'interface

- entre configurations

```
eqC(s:FifoC!c, s':FifoS!c) :=
    (s.plein = s'.plein) &
    (s.vide = s'.vide) &
    (s.top = s'.top);
```

- entre événements

```
eqEv(e:FifoC!ev, e':FifoS!ev) :=
    (e. = put & e'. = put) |
    (e. = get & e'. = get) |
    (e. = "" & e'. = "");
```

```
reachC(t) += FifoC!init(t) | <s><e>(FifoC!t(s, e, t) & reachC(s));
transC(s, e, t) := reachC(s) & FifoC!t(s, e, t);

reachS(t) += FifoS!init(t) | <s><e>(FifoS!t(s, e, t) & reachS(s));
transS(s, e, t) := reachS(s) & FifoS!t(s, e, t);
```

Calcul de la relation de bisimulation

[mec]

Calcul de la relation de bisimulation

```
[mec] bisim(s, s’) -= eqC(s, s’) & reachC(s) & reachS(s’) &
.
```

Calcul de la relation de bisimulation

```
[mec] bisim(s, s’) -= eqC(s, s’) & reachC(s) & reachS(s’) &
.  [e][t](transC(s, e, t) =>
.
```

Calcul de la relation de bisimulation

```
[mec] bisim(s, s’) -= eqC(s, s’) & reachC(s) & reachS(s’) &
.  [e][t](transC(s, e, t) =>
.    <e’><t’>(transS(s’, e’, t’) & eqEv(e, e’) & bisim(t, t’))) &
.
```

Calcul de la relation de bisimulation

```
[mec] bisim(s, s') -= eqC(s, s') & reachC(s) & reachS(s') &
.  [e][t](transC(s, e, t) =>
.    <e'><t'>(transS(s', e', t') & eqEv(e, e') & bisim(t, t'))) &
.  [e'][t'](transS(s', e', t') =>
.    <e><t>(transC(s, e, t) & eqEv(e, e') & bisim(t, t')));
```

Calcul de la relation de bisimulation

```
[mec] bisim(s, s') -= eqC(s, s') & reachC(s) & reachS(s') &
.  [e][t](transC(s, e, t) =>
.    <e'><t'>(transS(s', e', t') & eqEv(e, e') & bisim(t, t'))) &
.  [e'][t'](transS(s', e', t') =>
.    <e><t>(transC(s, e, t) & eqEv(e, e') & bisim(t, t')));
       bisim: (FifoC!c, FifoS!c) -> bool

[mec]
```

Calcul de la relation de bisimulation

```
[mec] bisim(s, s') -= eqC(s, s') & reachC(s) & reachS(s') &
.  [e][t](transC(s, e, t) =>
.    <e'><t'>(transS(s', e', t') & eqEv(e, e') & bisim(t, t'))) &
.  [e'][t'](transS(s', e', t') =>
.    <e><t>(transC(s, e, t) & eqEv(e, e') & bisim(t, t')));
       bisim: (FifoC!c, FifoS!c) -> bool

[mec] estBisim(x) := x = (
.    [i](FifoC!init(i) => <i'>(FifoS!init(i') & bisim(i, i'))) &
.    [i'](FifoS!init(i') => <i>(FifoC!init(i) & bisim(i, i'))));
```

Calcul de la relation de bisimulation

```
[mec] bisim(s, s') -= eqC(s, s') & reachC(s) & reachS(s') &
.  [e][t](transC(s, e, t) =>
.    <e'><t'>(transS(s', e', t') & eqEv(e, e') & bisim(t, t'))) &
.  [e'][t'](transS(s', e', t') =>
.    <e><t>(transC(s, e, t) & eqEv(e, e') & bisim(t, t')));
       bisim: (FifoC!c, FifoS!c) -> bool

[mec] estBisim(x) := x = (
.    [i](FifoC!init(i) => <i'>(FifoS!init(i') & bisim(i, i'))) &
.    [i'](FifoS!init(i') => <i>(FifoC!init(i) & bisim(i, i'))));
       estBisim: (bool) -> bool

[mec]
```

Calcul de la relation de bisimulation

```
[mec] bisim(s, s') -= eqC(s, s') & reachC(s) & reachS(s') &
.  [e][t](transC(s, e, t) =>
.    <e'><t'>(transS(s', e', t') & eqEv(e, e') & bisim(t, t'))) &
.  [e'][t'](transS(s', e', t') =>
.    <e><t>(transC(s, e, t) & eqEv(e, e') & bisim(t, t')));
       bisim: (FifoC!c, FifoS!c) -> bool

[mec] estBisim(x) := x = (
.    [i](FifoC!init(i) => <i'>(FifoS!init(i') & bisim(i, i'))) &
.    [i'](FifoS!init(i') => <i>(FifoC!init(i) & bisim(i, i'))));
       estBisim: (bool) -> bool

[mec] :display estBisim
```

Calcul de la relation de bisimulation

```
[mec] bisim(s, s') -= eqC(s, s') & reachC(s) & reachS(s') &
.  [e][t](transC(s, e, t) =>
.    <e'><t'>(transS(s', e', t') & eqEv(e, e') & bisim(t, t'))) &
.  [e'][t'](transS(s', e', t') =>
.    <e><t>(transC(s, e, t) & eqEv(e, e') & bisim(t, t')));
       bisim: (FifoC!c, FifoS!c) -> bool

[mec] estBisim(x) := x = (
.    [i](FifoC!init(i) => <i'>(FifoS!init(i') & bisim(i, i'))) &
.    [i'](FifoS!init(i') => <i>(FifoC!init(i) & bisim(i, i'))));
       estBisim: (bool) -> bool

[mec] :display estBisim
(true)

[mec]
```

Calcul de la relation de bisimulation

```
[mec] bisim(s, s') -= eqC(s, s') & reachC(s) & reachS(s') &
.  [e][t](transC(s, e, t) =>
.    <e'><t'>(transS(s', e', t') & eqEv(e, e') & bisim(t, t'))) &
.  [e'][t'](transS(s', e', t') =>
.    <e><t>(transC(s, e, t) & eqEv(e, e') & bisim(t, t')));
       bisim: (FifoC!c, FifoS!c) -> bool

[mec] estBisim(x) := x = (
.    [i](FifoC!init(i) => <i'>(FifoS!init(i') & bisim(i, i'))) &
.    [i'](FifoS!init(i') => <i>(FifoC!init(i) & bisim(i, i'))));
       estBisim: (bool) -> bool

[mec] :display estBisim
(true)

[mec] :rel-cardinal bisim
```

Calcul de la relation de bisimulation

```
[mec] bisim(s, s') -= eqC(s, s') & reachC(s) & reachS(s') &
.  [e][t](transC(s, e, t) =>
.    <e'><t'>(transS(s', e', t') & eqEv(e, e') & bisim(t, t'))) &
.  [e'][t'](transS(s', e', t') =>
.    <e><t>(transC(s, e, t) & eqEv(e, e') & bisim(t, t')));
       bisim: (FifoC!c, FifoS!c) -> bool

[mec] estBisim(x) := x = (
.    [i](FifoC!init(i) => <i'>(FifoS!init(i') & bisim(i, i'))) &
.    [i'](FifoS!init(i') => <i>(FifoC!init(i) & bisim(i, i'))));
       estBisim: (bool) -> bool

[mec] :display estBisim
(true)

[mec] :rel-cardinal bisim
cardinal of bisim: 620

[mec]
```

Temps de calcul

Machine de test :

- Pentium IV à 2,8 GHz
- 1 Go de RAM
- NetBSD 1.6ZF

Taille	Config FifoC	Trans FifoC	Config FifoS	Trans FifoS	bisim	Temps	Mémoire
3	45	129	60	172	180	7s	36 Mo
4	124	364	155	455	620	34s	87 Mo
5	315	935	378	1122	1890	138s	181 Mo
6	762	2274	889	2653	5334	456s	400 Mo

Synthèse de contrôleurs

Synthèse de contrôleurs

- Synthétiser un contrôleur revient à calculer une stratégie gagnante dans un jeu à deux joueurs

Synthèse de contrôleurs

- Synthétiser un contrôleur revient à calculer une stratégie gagnante dans un jeu à deux joueurs
- Mec V permet de calculer de telles stratégies gagnantes

Synthèse de contrôleurs

- Synthétiser un contrôleur revient à calculer une stratégie gagnante dans un jeu à deux joueurs
- Mec V permet de calculer de telles stratégies gagnantes
- Stage de DEA : étude de l'algorithme de calcul de stratégies gagnantes dans un jeu de parité de Vöge et Jurdziński (CAV 2000)

Synthèse de contrôleurs

- Synthétiser un contrôleur revient à calculer une stratégie gagnante dans un jeu à deux joueurs
- Mec V permet de calculer de telles stratégies gagnantes
- Stage de DEA : étude de l'algorithme de calcul de stratégies gagnantes dans un jeu de parité de Vöge et Jurdziński (CAV 2000)
- A. Vincent, *Synthèse de contrôleurs et stratégies gagnantes dans les jeux de parité*, MSR 2001
- A. Arnold, A. Vincent et I. Walukiewicz, *Games for synthesis of controllers with partial observation*, TCS 303(1):7–34, 2003

Conclusion

Conclusion

- Le model-checker a été écrit
- Son pouvoir d'expression ouvre la possibilité de :
  - traiter des propriétés d'équité, de bisimulation
  - calculer des stratégies gagnantes dans un jeu
- Les améliorations seront basées sur les retours des utilisateurs