Larissa, un langage d'aspects pour le développement des systèmes réactifs sûrs

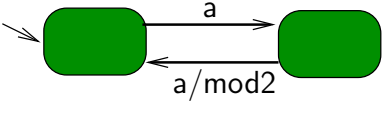
Karine Altisen, Florence Maraninchi, David Stauch
Verimag, Grenoble, France

Introduction

- **Les systèmes réactifs sont des systèmes en interaction constante avec leur environnement**
- **Ils sont programmés dans des langages dédiés**
- **Des préoccupations transversales existent, mais des langages d'aspects existants ne peuvent pas être utilisés**
- **Larissa est un langage d'aspects pour le langage dédié Argos**
- **Cette article : modélisation d'un système réactif typique avec Argos et Larissa**

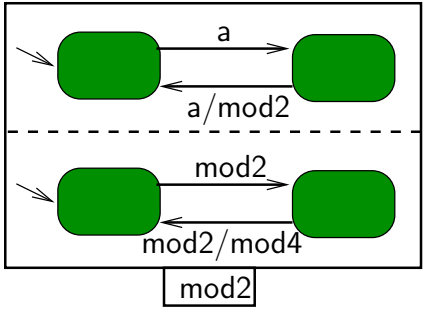
Argos

- **Langage hiérarchique à base d'automates synchrones**
- **Élément de base : automates de Mealy complets et déterministes avec des signaux booléens**
- **Interface : un ensemble d'entrées, un de sorties**



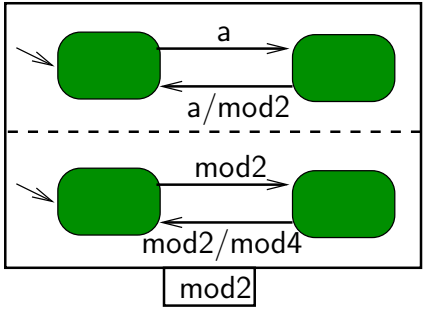
Argos

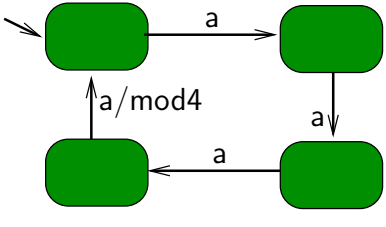
- **Langage hiérarchique à base d'automates synchrones**
- **Élément de base : automates de Mealy complets et déterministes avec des signaux booléens**
- **Interface : un ensemble d'entrées, un de sorties**
- **Opérateurs : produit parallèle, encapsulation**



Argos

- **Langage hiérarchique à base d'automates synchrones**
- **Élément de base : automates de Mealy complets et déterministes avec des signaux booléens**
- **Interface : un ensemble d'entrées, un de sorties**
- **Opérateurs : produit parallèle, encapsulation**
- **Sémantique définie par automate plat**



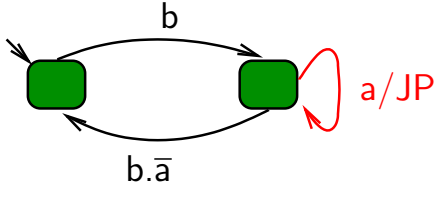


Larissa

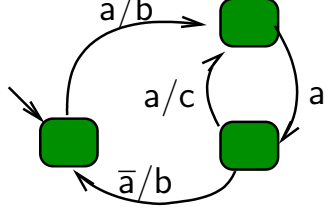
- Un langage d'aspects pour Argos
- Il y a des coupes et des inserts :
  - Les points de jointure sont des transitions
  - Les coupes choisissent des transitions dans des automates
  - Les inserts remplacent ces transitions :
    - par d'autres transitions
    - par insertion d'un automate
- Le tissage d'aspect est un nouvel opérateur :
  - ne peut pas être fait avec les opérateurs existants
  - préserve complétude et déterminisme
  - préserve l'équivalence sémantique entre programmes

Les coupes

- **Coupe : un automate observateur avec une sortie JP**
- **JP émis quand le programme est dans un point de jointure**
- **Implémenté avec produit parallèle**
- **Indépendant de l'implémentation du programme**



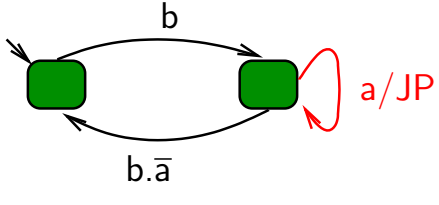
coupe



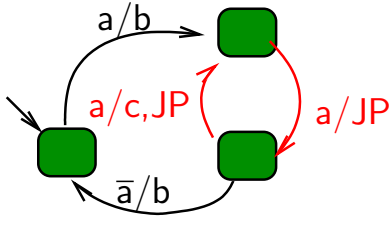
programme de base

Les coupes

- **Coupe : un automate observateur avec une sortie JP**
- **JP émis quand le programme est dans un point de jointure**
- **Implémenté avec produit parallèle**
- **Indépendant de l'implémentation du programme**



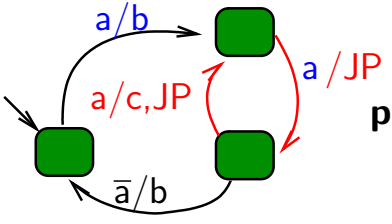
coupe



programme de jointure

Insert toInit

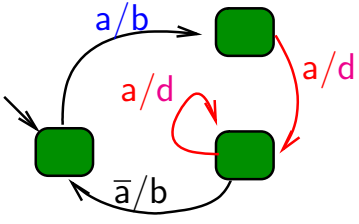
- **Quand on passe un point de jointure, l'insert change l'exécution du programme :**
  - **émet des sorties O**
  - **on va vers un *état but***
  - **l'état but est défini par une trace finie sur les entrées**
- **Exemple d'insert : trace a.a, sorties d**



programme de jointure

Insert toInit

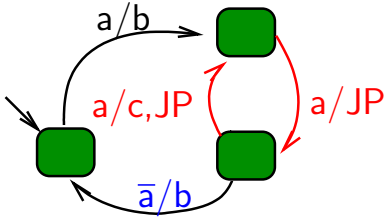
- Quand on passe un point de jointure, l'insert change l'exécution du programme :
  - émet des sorties O
  - on va vers un *état but*
  - l'état but est défini par une trace finie sur les entrées
- Exemple d'insert : trace a.a, sorties d



programme tissé

Insert toTarget

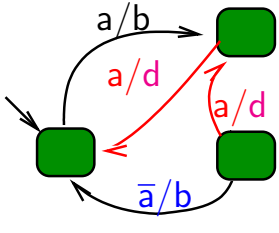
– **Comme toInit, mais execute la trace à partir de l'etat *but* de la transition de jointure**
– **Exemple d'insert : trace $\overline{a}$, sorties d**



programme de jointure

Insert toTarget

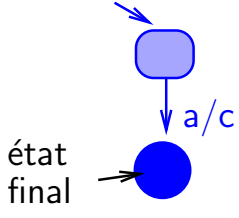
– **Comme toInit, mais execute la trace à partir de l'etat *but* de la transition de jointure**
– **Exemple d'insert : trace $\overline{a}$, sorties d**



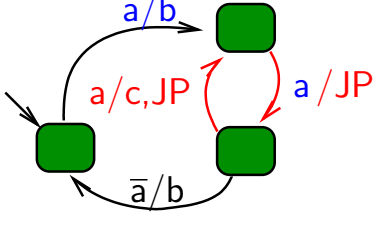
programme tissé

Programme d'insert

- **Remplace transition de jointure par un automate**
- **Transitions d'insert vont vers l'état initial de l'automate inséré, transitions de l'automate inséré qui allait vers l'état final vont vers l'état but défini par la trace**
- **Exemple : insert tolnit, trace a.a, sortie d**



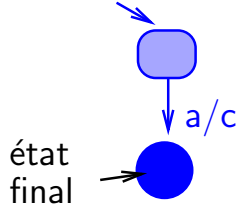
automate inseré



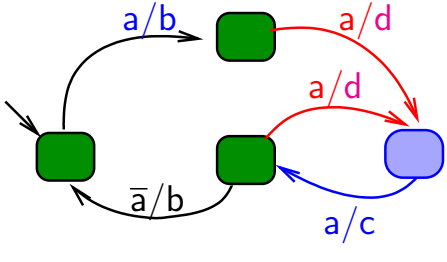
programme de jointure

Programme d'insert

- **Remplace transition de jointure par un automate**
- **Transitions d'insert vont vers l'état initial de l'automate inséré, transitions de l'automate inséré qui allait vers l'état final vont vers l'état but défini par la trace**
- **Exemple : insert tolnit, trace a.a, sortie d**



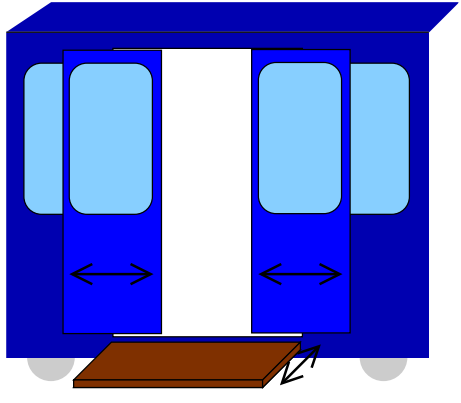
automate inseré



programme tissé

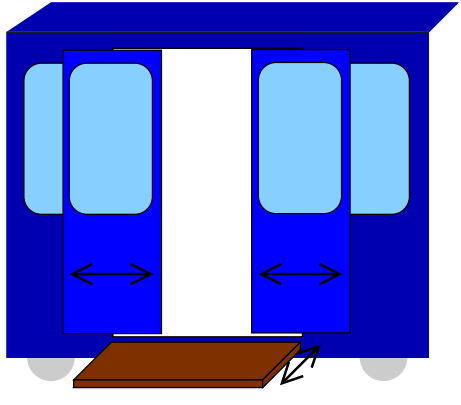
Exemple : contrôleur de porte de tramway

- Implémentation d'un contrôleur de porte de tramway :
  - ouvre et ferme la porte
  - sort et rentre une passerelle



Exemple : contrôleur de porte de tramway

- Implémentation d'un contrôleur de porte de tramway :
  - ouvre et ferme la porte
  - sort et rentre une passerelle



- Implémentation d'une version sans passerelle en Argos
- Ajout de la passerelle avec des aspects Larissa

Propriétés de sûreté

- **contrôleur de porte doit respecter trois propriétés de sûreté :**
  - **la porte n'est jamais ouverte en dehors d'une station**
  - **la passerelle n'est jamais sortie en dehors d'une station**
  - **la passerelle ne bouge pas si la porte n'est pas fermée**

Interface du contrôleur

– Les entrées sont reçues de l'environnement

enStation	Le tram est dans une station
attDep	Le tram veut partir
porteOuv	La porte est ouverte
porteFermé	La porte est fermée
demPorte	Un passager veut quitter le tram
timer	Déclenche un timer

Interface du contrôleur

– Les sorties sont émises vers l'environnement

porteOK	La porte est fermée et prête à partir
ouvrirPorte	ouvre la porte
fermerPorte	ferme la porte
beep	émet un son
setTimer	démarre un timer

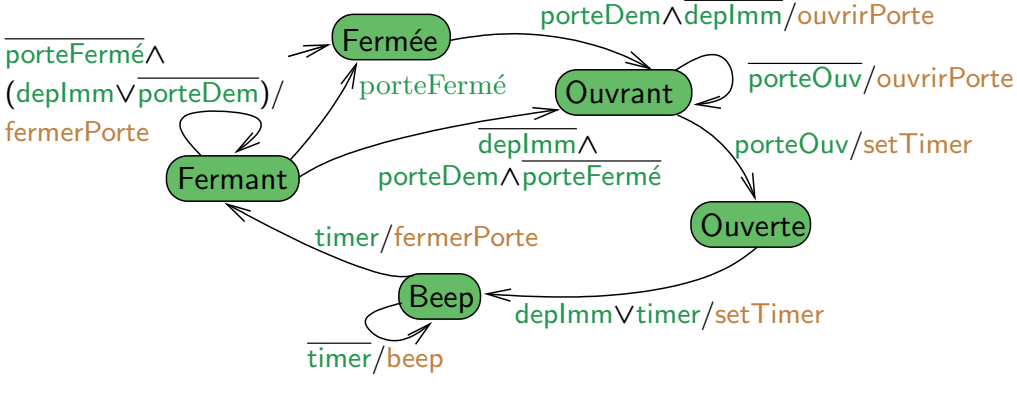
Signaux auxiliaires

- **Entrées additionnelles du contrôleur**
- **Calculées à partir des entrées, par un programme Argos parallèle au contrôleur**

accepterDem	**Le passager peut demander la porte ou la passerelle**
porteDem	**Le passager a demandé l'ouverture de la porte**
depImm	**Le tram veut quitter la station**

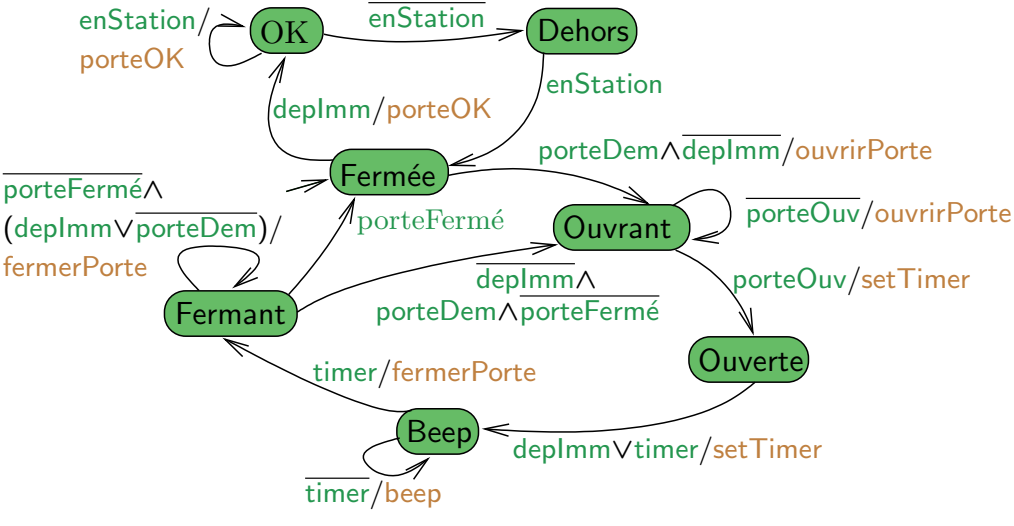
Implémentation en Argos

– On implémente d'abord un contrôleur sans passerelle



Implémentation en Argos

– On implémente d'abord un contrôleur sans passerelle



Ajout de la passerelle

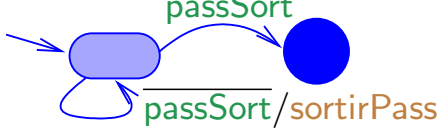
- **Ajout de la passerelle dans le contrôleur avec des aspects**
- **Deux aspects :**
  - **un aspect sort la passerelle quand elle a été demandée**
  - **un aspect rentre la passerelle avant que le tram ne quitte la station**

Interface de la passerelle

- **Les entrées de la passerelle :**
  - **passSort** La passerelle est sortie
  - **passEnt** La passerelle est rentrée
  - **demPass** Un passager veut utiliser la passerelle
- **Les sorties de la passerelle :**
  - **sortirPass** Sort la passerelle
  - **rentrerPass** Rentre la passerelle
- **Un nouveau signal auxiliaire :**
  - **passDem** Le passager a demandé la passerelle

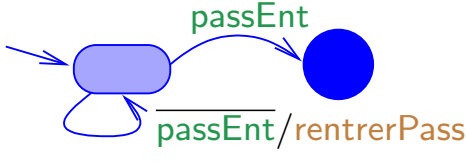
Aspect de l'ouverture

- Sort la passerelle avant l'ouverture de la porte, si elle a été demandée.
- Coupe choisit toutes les transitions où ouvrirPorte∧passDem∧porteFermé∧$\overline{\text{passSort}}$ est vrai.
- Coupe respecte les propriétés de sûreté :
  - on peut émettre ouvrirPorte (le tram est en station)
  - la porte est fermée
- Insert toTarget, insère un automate qui sort la passerelle :

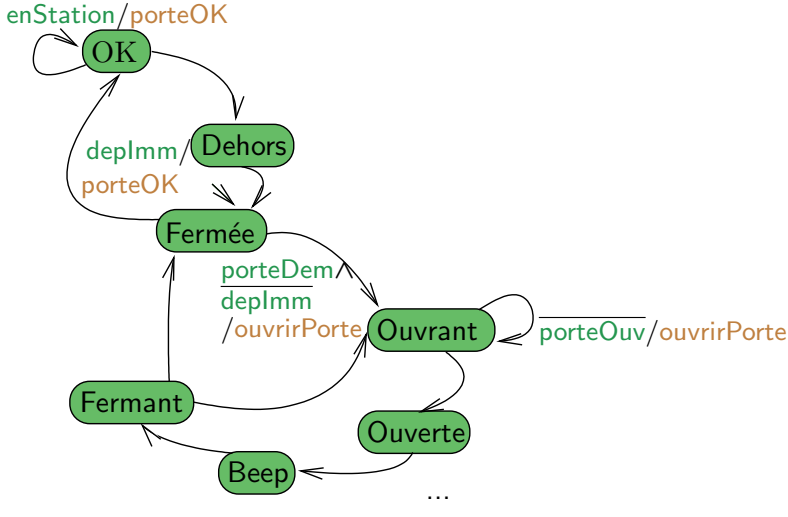


Aspect de la fermeture

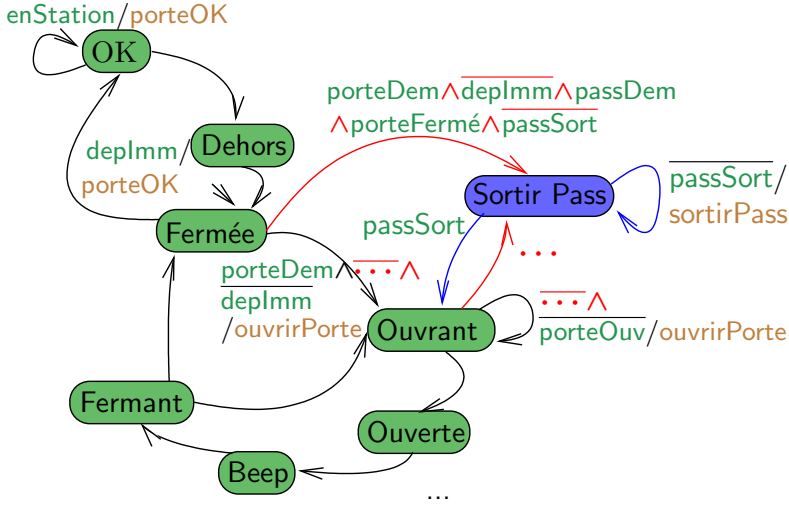
- Rentre la passerelle avant que le tram émette porteOK
- Coupe choisit toutes les transitions où porteOK$\wedge\overline{\text{passEnt}}$ est vrai.
- Coupe respecte les propriétés de sûreté :
  - rentre la passerelle avant d'émettre porteOK
  - on peut émettre porteOK(la porte est fermée)
- Insert toTarget, insère un automate qui rentre la passerelle :



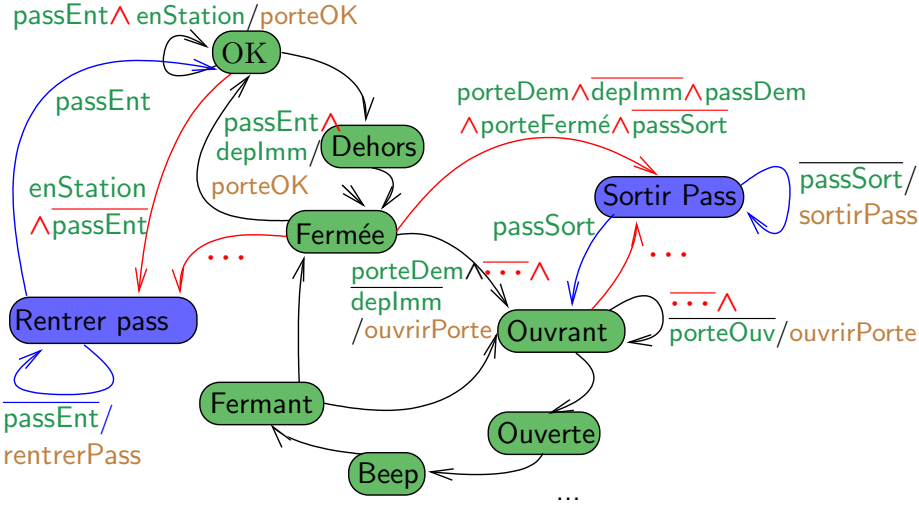
Application au contrôleur



Application au contrôleur



Application au contrôleur



Vérification formelle

- On veut vérifier formellement les propriétés de sûreté
- Facilement possible parce que le programme tissé est un programme Argos
- Modélisation de l'environnement physique du contrôleur (porte, passerelle, tram)
- On vérifie que le contrôleur avec les aspects respecte les propriétés de sûreté dans l'environnement
- Vérification effectué avec un outil de model-checking

Conclusion

- **Premier exemple d'un système réactif critique**
- **Larissa est bien adapté à la programmation incrémentale**
- **Le contrôleur est un exemple d'un aspect fonctionnel**
- **Autres exemples traités sont aussi des aspects fonctionnels**

Conclusion

- Premier exemple d'un système réactif critique
- Larissa est bien adapté à la programmation incrémentale
- Le contrôleur est un exemple d'un aspect fonctionnel
- Autres exemples traités sont aussi des aspects fonctionnels
- Question : est-ce qu'il y a des préoccupations non fonctionnelles transverses dans les systèmes réactifs ?
- Idée : modélisation des systèmes sur puce :
  - Modélisation du comportement fonctionnel en Argos, ajout des informations temporelles avec Larissa