Simulation de systèmes réactifs indéterministes

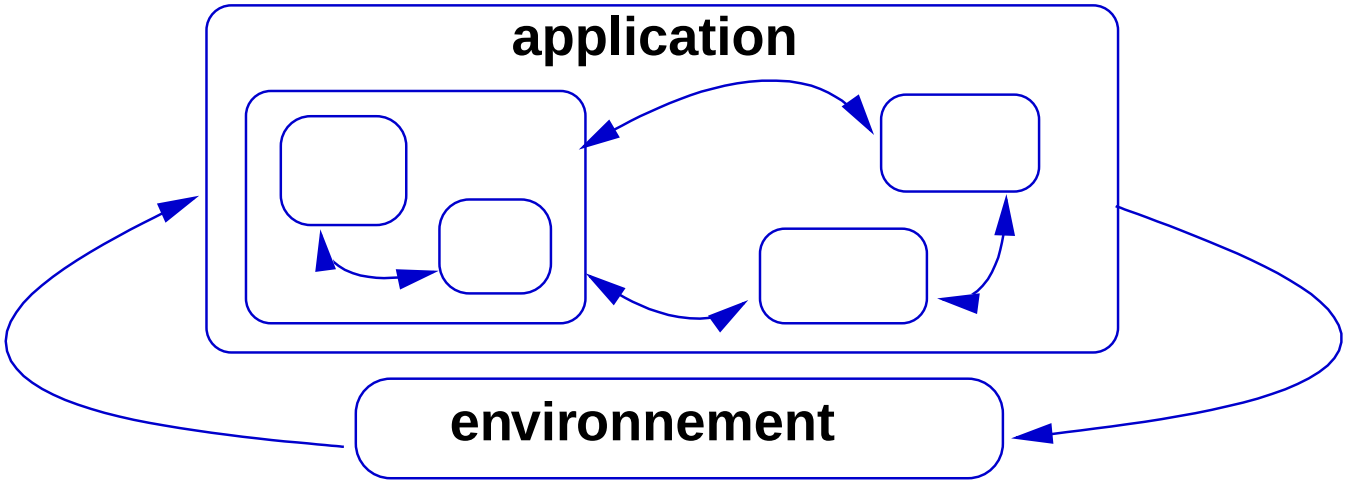
P.Raymond, E.Jahier, Y.Roux

Verimag

Systèmes réactifs indéterministes

But : test/simulation précoce

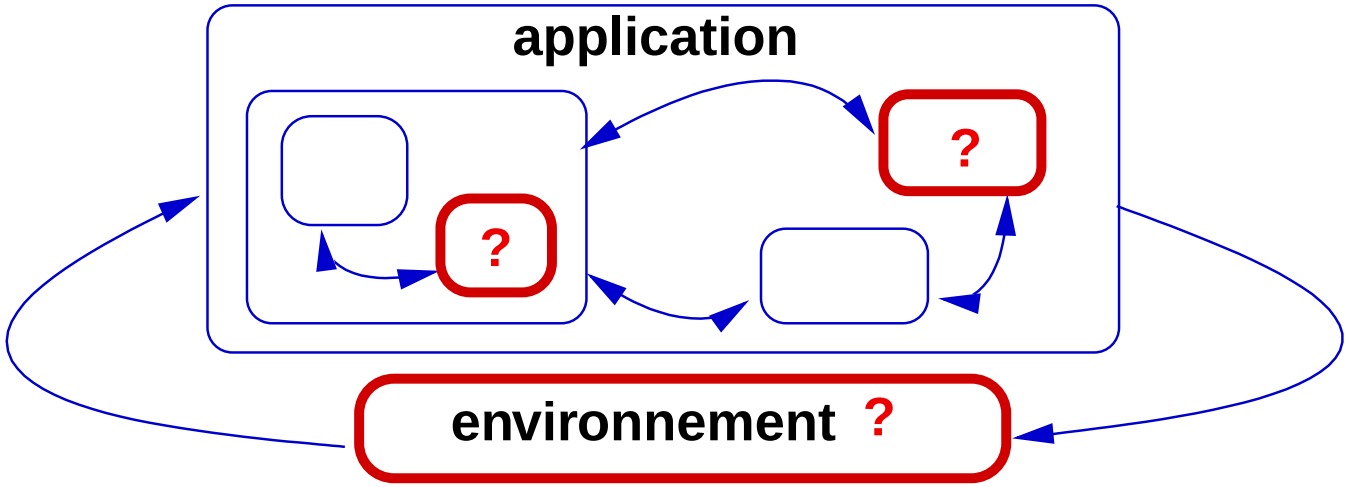
Système réactif “fermé” (application + environnement)



Systèmes réactifs indéterministes

But : test/simulation précoce

Système réactif “fermé” (application + environnement)



- Environnement : par nature indéterministe
- Composants non totalement réalisés : connus par des spécifications ⇔ systèmes indéterministes

Modèle pour la simulation

Approche synchrone

Les modules indéterministes sont destinés à se substituer à des modules synchrones :

- **interface bien identifiée (E/S)**
- **fonctionnement cyclique dans mémoire interne bornée (M) :**
  - ⋆ **le module reçoit E_i, et, suivant son état interne M_i,**
  - ⋆ **produit S_i,**
  - ⋆ **change sont état interne M_{i+1}**

“Seule” différence : $(S_i, (E_i, M_i))$ n’est pas une fonction, mais une *relation* quelconque.

Expression du comportement

- **Aussi riche que possible (dans le cadre synchrone) : doit pouvoir faire au moins du Esterel/Lustre.**
- **Style impératif et déclaratif :**
  - ⋆ **déclaratif utile pour exprimer les relations entre variables (en particulier numériques),**
  - ⋆ **impératif (séquentiel) pour exprimer la structure de contrôle du module.**

Au final : modèle d'automates interprétés dont les transitions sont étiquetées par des relations (Lucky)

Les variables

- les entrées (non contrôlables),
- les sorties, les variables internes (contrôlables),
- les valeurs précédentes de celles-ci (cf. `pre` Lustre).

Les relations

- *formules* logiques sur ces variables, avec les opérateurs arithmétiques et logique usuels,
- on note •X pour “pre X”,

Exemple : `if A then (X >= •X) else (X <= •X)`

L'automate

Un système de transition $(Q, q_0, T \subseteq Q \times \Phi \times Q)$,

où Φ est l'ensemble des formules considérées :

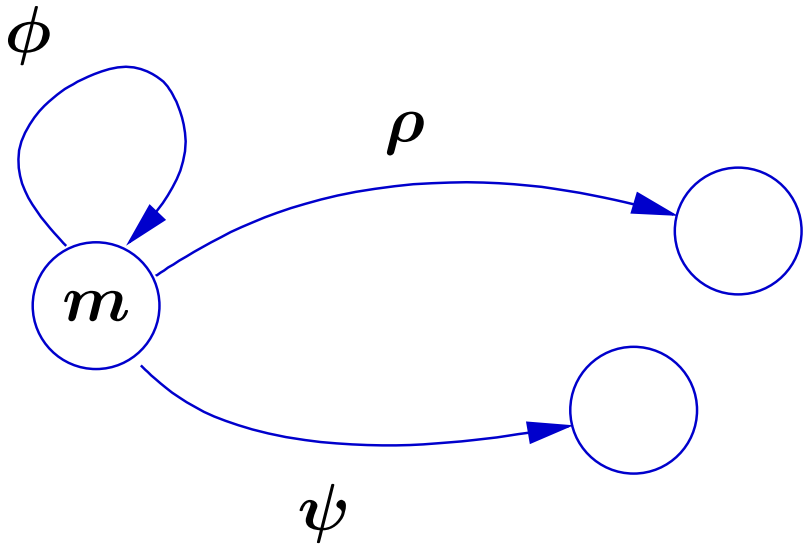
$\Phi = M \times E \times S \times L \longrightarrow$ B (avec $M = E \times S \times L$)

La mémoire initiale m_0 est donnée.

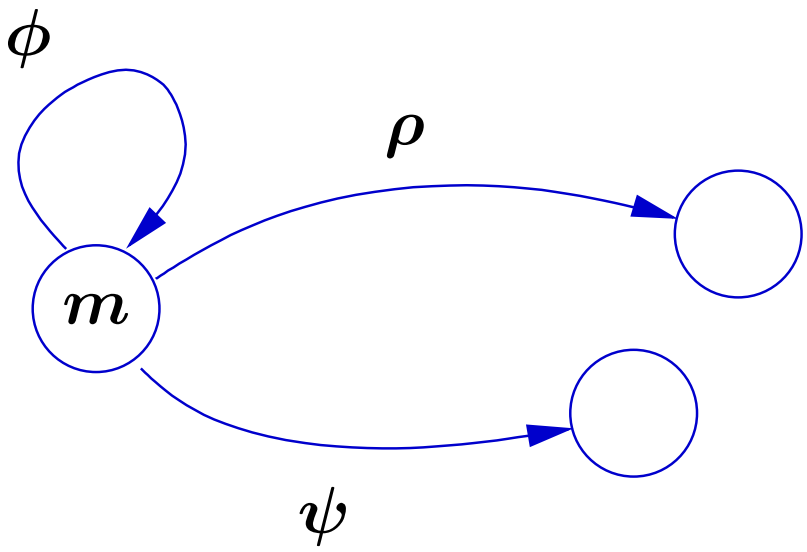
Exécution

- Un état à l'exécution = (q, m), (initialement (q_0, m_0))
  (point de contrôle, état précédent des variables)
- Un pas de "calcul" pour une valuation e des entrées =
  - ⋆ choisir $q \xrightarrow{\phi} q'$ telle que $\exists s, l \mid \phi(m, e, s, l)$
  - ⋆ Émettre s et continuer dans l'état (q', m') où $m' = (e, s, l)$

Exemple

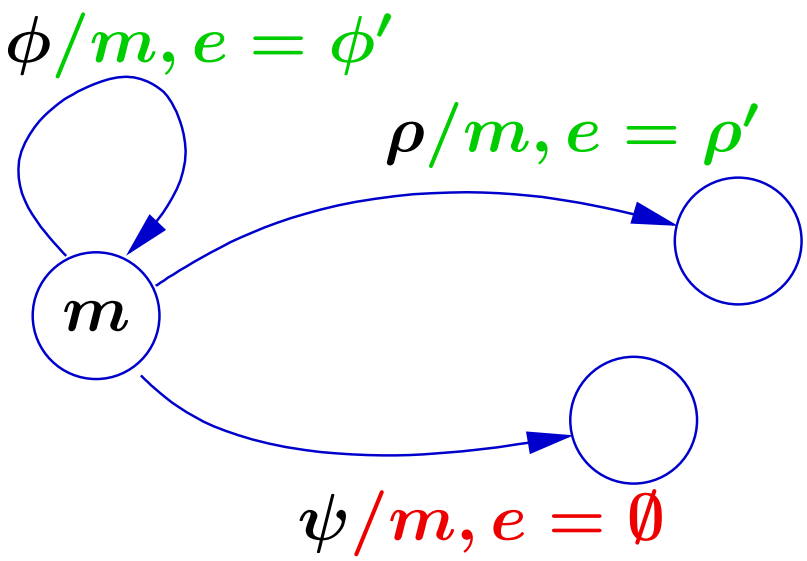


Exemple



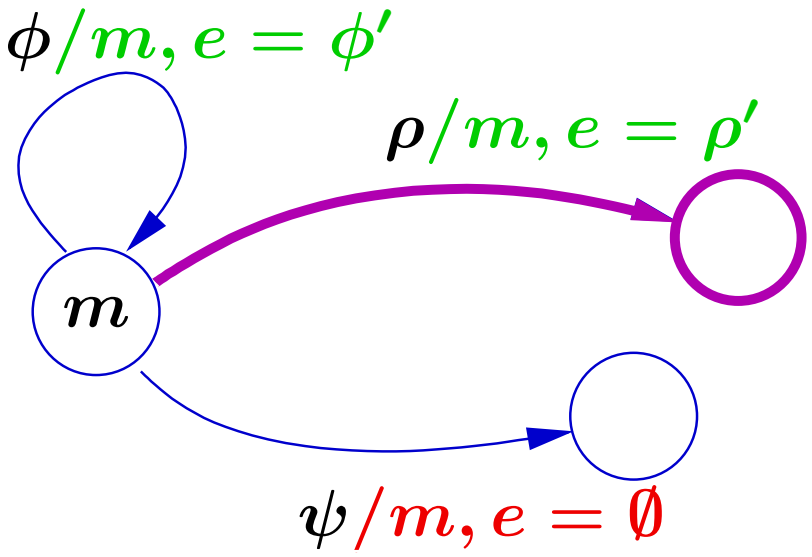
entrée e

Exemple



entrée e

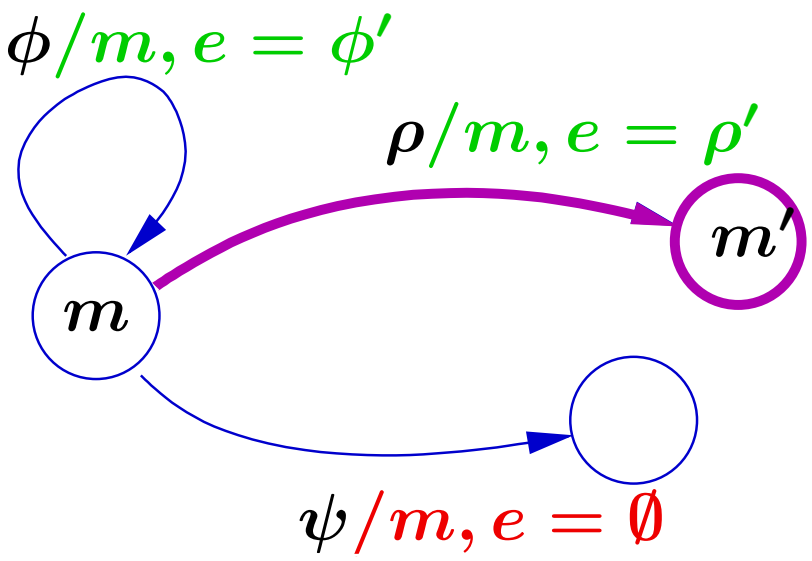
Exemple



entrée e

choix de ρ
et de (s, l) tels que
$\rho'(s, l)$

Exemple



entrée e

choix de ρ
et de (s, l) **tels que**
$\rho'(s, l)$

$$m' = (e, s, l)$$

Remarque :

- **Le système peut se bloquer,**

Maîtrise de l'indéterminisme

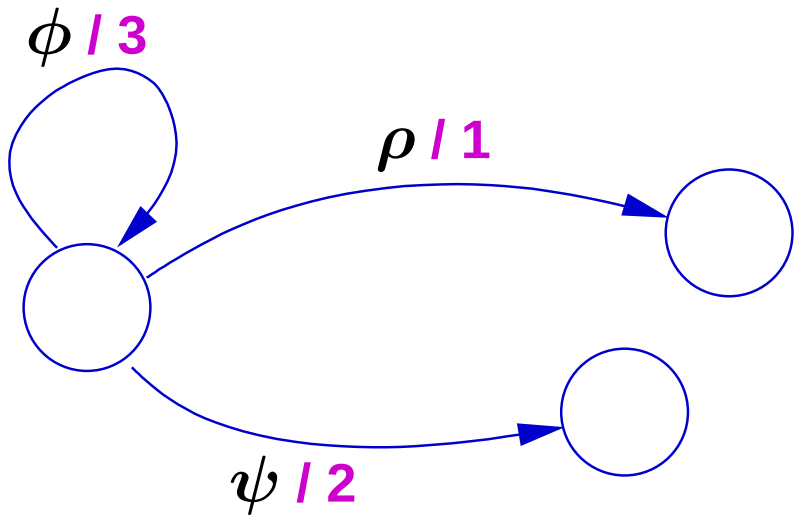
Deux niveaux :

- **choix de la transition (et de la relation ϕ) :**
  on peut facilement adjoindre des infos probabilistes ;
- **choix d'une solution dans ϕ, difficile à maîtriser :**
  on se contente de critères simples (pseudo-équité).

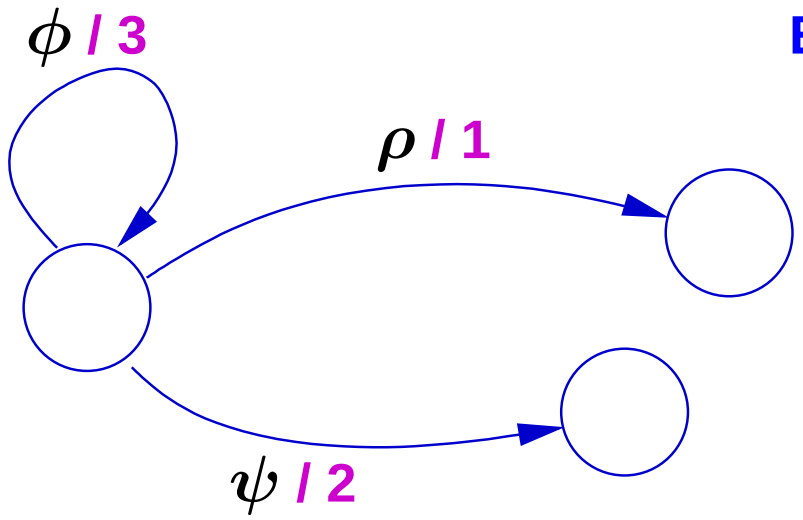
Poids

L'automate est une "machine" à produire des contraintes, le choix de contrainte est influencé par un poids relatif.

Poids relatifs

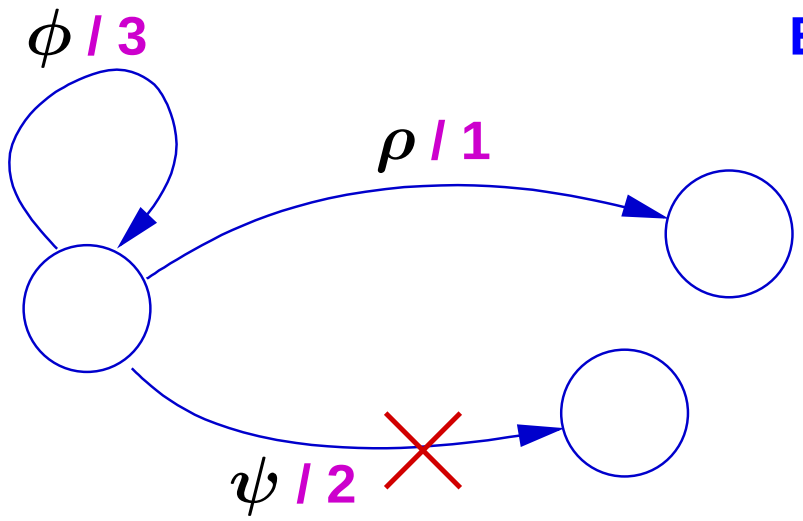


Poids relatifs



Exécution pour (m, e)

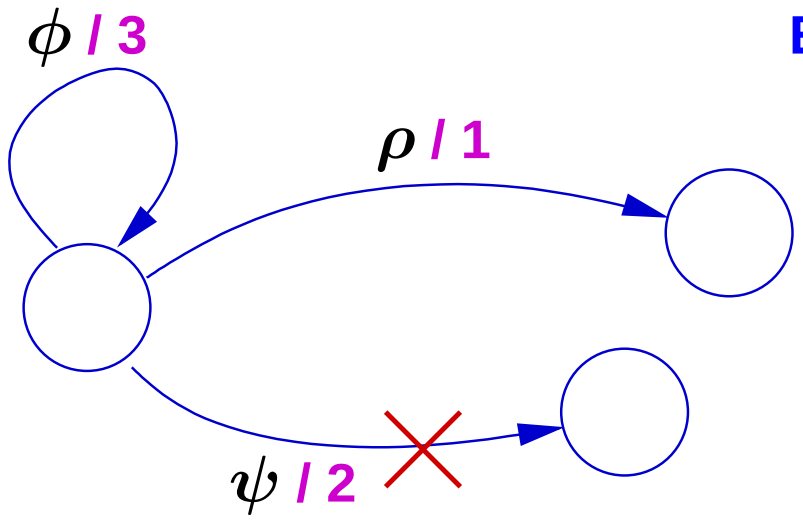
Poids relatifs



Exécution pour (m, e)

ψ impossible,

Poids relatifs



Exécution pour (m, e)

ψ impossible,
proba de ϕ = 3/4
proba de ρ = 1/4

Poids dynamiques

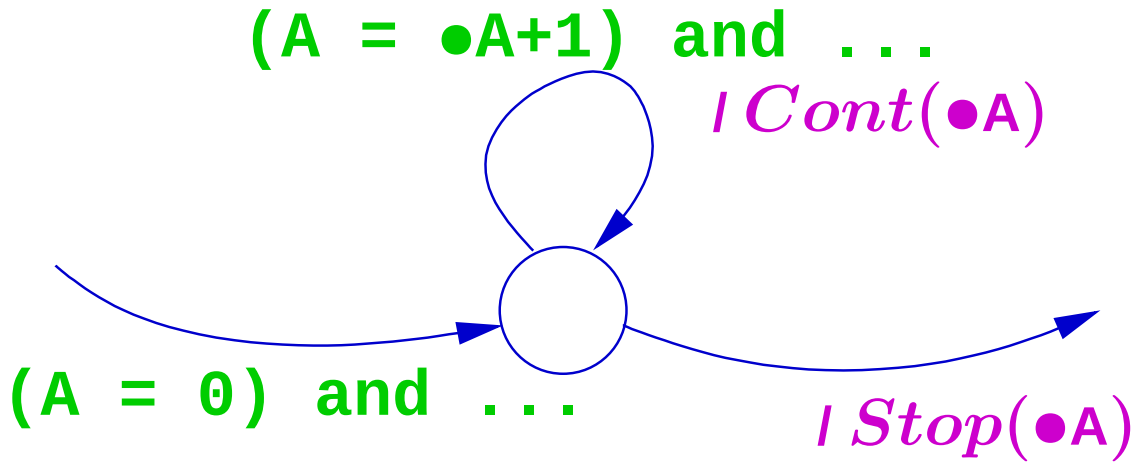
Le poids peut dépendre des variables :

- **d'entrée (l'environnement influence les probas),**
- **de mémoire (le passé influence les probas),**
- **mais pas des contrôlables**

Exemple de poids dynamiques

Simulation des processus vivants, qui “s’usent” avec le temps :

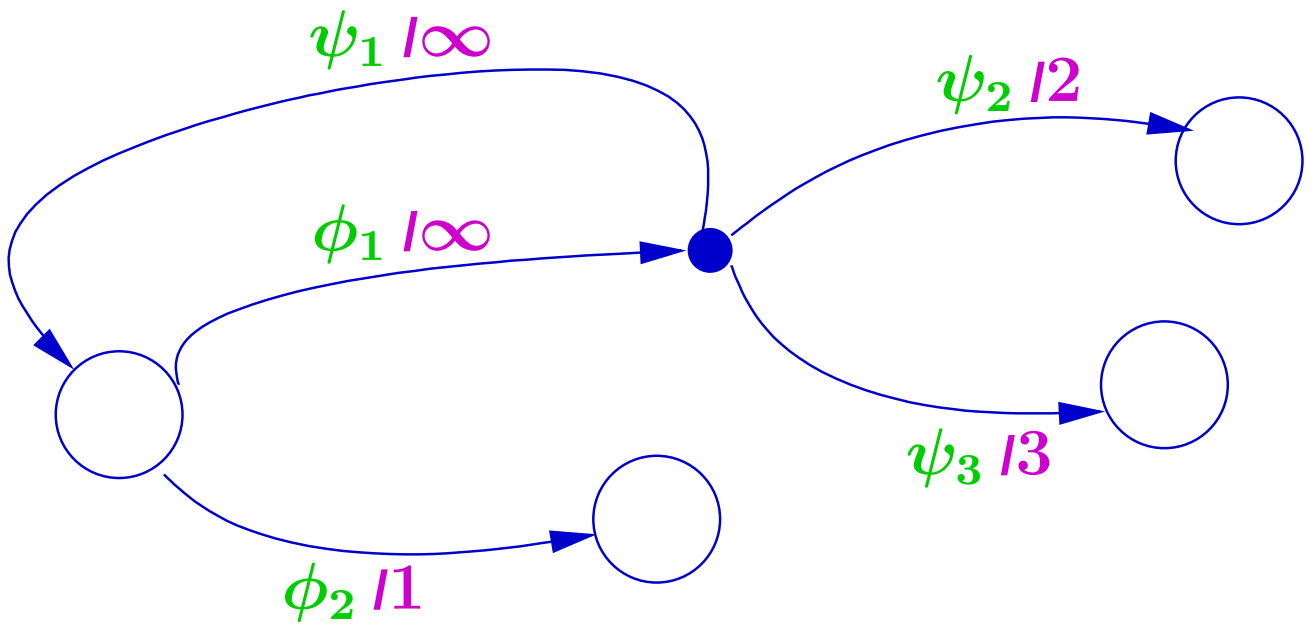
- **utilisation d’une variable interne A(ge),**
- **le rapport des poids $Cont$(inuer) et $Stop$(per) décroît avec l’Age**



Autres caractéristiques

- **Poids infini (expression des priorités)**
- **Transitions arborescentes (factorisation)**

Permet d'exprimer des priorités/probabilités complexes :



Parallélisme

- **Imbriqué (hiérarchie à la SynchCharts/Argos) trop complexe : c'est du domaine "langage de haut niveau" doit être "compilé".**
- **Choix : un parallélisme au top-level seulement, i.e. un système complet est un ensemble d'automates fournissant des contraintes. Le "produit" de ces automates peut être fait à la volée, au cours de la simulation.**

Modèle vs langage

Les automates à contraintes sont un modèle d'exécution, défini avec le soucis :

- **d'être général,**
- **d'avoir une sémantique opérationnelle "simple" (soucis efficacité de la simulation),**

mais pas du tout d'être pratique/facile à utiliser par un humain : c'est un code "machine", qui doit (idéalement) être la cible de langages de plus haut niveau.

Langage de haut niveau

Adaptation des langages synchrones ?

- **Les langages impératifs à structure de contrôle sont bien adaptés (SyncCharts, Esterel),**
- **Lustre beaucoup moins : trop “fonctionnel”, pas de structure de contrôle.**

Un langage a été défini : Lutin, proche d’Esterel/SyncCharts, mais où la structure de contrôle est à base d’opérations régulières.

Lutin

Principes :

- **Un ensemble de variables (entrées/sorties/locales) bien défini,**
- **un moyen de faire référence au passé des variables (`pre`),**
- **des formules de base (relations) sur ces variables,**
- **une structure de contrôle simple à base :**
  - ⋆ **opérateur séquence : "`;`"**
  - ⋆ **opérateur boucle : "`loop ... end`"**
  - ⋆ **opérateur choix : "`{ ... | ... | ... }`"**

En bref : expressions régulières dont les atomes sont des relations.

Autres facilités/constructions

- **modularité : “bouts” de comportements, éventuellement paramétrés :**

```
trace croit(X:real) = loop (X >= pre X) end ;
```

- **choix à poids :** `{ X weight 2 | Y weight 2 | Z }`
- **choix prioritaires :** `{ X |> Y |> Z }`
- **Boucles contraintes :**

```
loop [200,300] X end
loop  500:40 X end
```

Utilisation

Test

Un environnement (indéterministe) + un programme sous test.

Simulation

Un réseau de modules, déterministes ou non, connectés dans un réseau de Khan (sémantique synchrone stricte).