UPMC PARISUNIVERSITAS

LIP6

Langages synchrones

Jérôme Hugues (ex. ENST)
Emmanuelle Encrenaz-Tiphène

(emmanuelle.encrenaz@lip6.fr)

M2 SAR/STL LS

V3.3, septembre 2014

UPMC PARISUNIVERSITAS

LIP6

M2 SAR/STL LS

3. Vérification fonctionnelle

a) Génération et parcours du graphe d'états du système

b) Propriétés de sûreté / observateurs

UPMC PARISUNIVERSITAS

Modèles de programme

LIP6

- Un programme Lustre est représentable par une machine de Mealy (interprétée ou non).
- Notion d'état (variables internes du programme) et de transition (effet d'une réaction sur les variables d'état et les sorties du programme).

Automate explicite
- États énumérés
- Transition globale d'un état à un successeur

Automate implicite
- États = produit cartésien de val. de variables internes
- Transition : une collection de fonctions décrivant l'évolution de chaque variable en une réaction

---

UPMC PARISUNIVERSITAS

LIP6

En calculant les successeurs en une transition de chaque état atteint, on obtient l'arbre d'exécution infini du programme.

En regroupant les états identiques (ayant les mêmes valeurs pour toutes les variables qui les composent), on obtient un graphe orienté représentant l'automate explicite du programme, déplié sur toutes les variables internes du programme : c'est une machine de Mealy.

On peut analyser les comportements du programme en étudiant la structure de ce graphe :
les états qui le composent (et les valeurs de certains signaux dans ces états),
les successions d'états le long d'un chemin,
les composantes fortement connexes, …

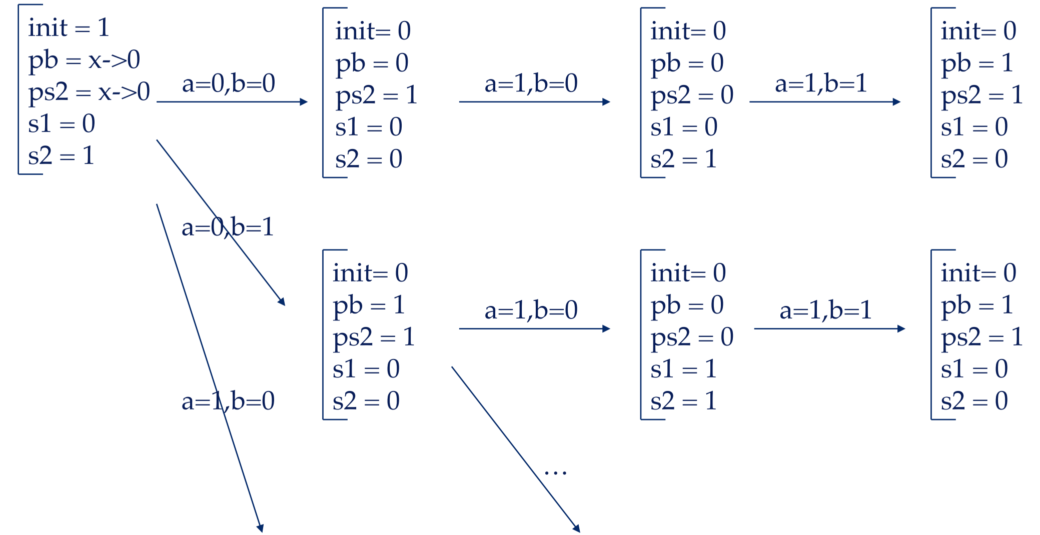
Running Example

- **Le nœud n1 :**

```
node n1 (a,b : bool) return (s1,s2: bool);
let
  s1 = 0 -> a and pre b;
  s2 = 1 -> not pre s2;
tel
```

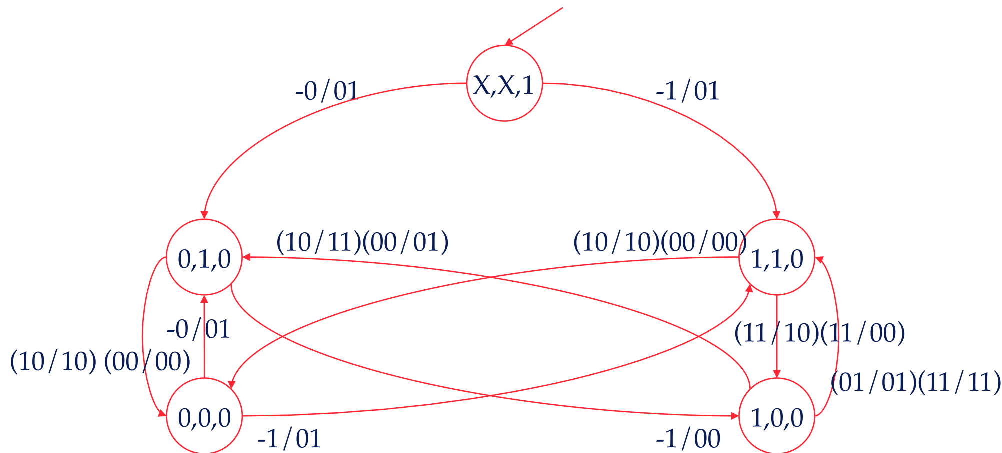
```
/* réaction */
             s1 = if (init) 0 else (a && pb);
             s2 = if (init) 1 else  (! ps2);
             pb = b;
             ps2 = s2;
             init = 0;
```

Parcours d'espace d'états à partir de l'automate implicite



Automate déplié de n1

- **Dépliage selon pb, ps2, init**



b) Propriétés du système

1. **Propriétés de sûreté**
2. **Propriétés de vivacité**
3. **Equité**

M2 SAR/STL LS

Propriétés du système

- **Propriétés : Expression des bonnes caractéristiques du système**
  - **Fonctionnelles** :
    - *Calcul réalisé est celui attendu*
    - *Accès exclusif aux ressources critiques*
    - *Pas de blocage des ressources*
    - *Toute requête finira par être traitée*
  - **Fonctionnelles et temporelles**:
    - *L'alarme se déclenche au plus tard 10 ms après la détection de l'erreur*
    - *Le circuit fonctionne pour une horloge inférieure à 1 Ghz*
  - **Non fonctionnelles**
    - *Le circuit a une consommation de 10mW à une fréquence de 100 Mhz*
    - *La bande passante du réseau est 1 Gbits/s*
    - *Le circuit fonctionne pour une température comprise entre -50°C et 50°C*
- **On se limite aux propriétés fonctionnelles**

Sûreté et Vivacité (1)

- **Propriétés de sûreté :**
  - Le système ***ne peut pas*** effectuer quelque chose de ***mauvais***
    - *« pour une ressource partagée critique, au même instant, il ne peut y avoir deux processus simultanément en section critique »*
  - Vérifiées sur les *états* ou sur des *séquences finies d'états* du système
    - *Analyse des états de l'espace d'états ou reconnaissance de séquences par automate fini (et analyse des états accepteurs de l'automate)*

    →*L'ensemble des états accessibles suffit*
  - Simplement exprimable : prédicats sur les variables d'états, automates observateurs
  - Vérification implantée dans Lesar, Scade (Lustre) / Auto, Esterel-Studio (Esterel)

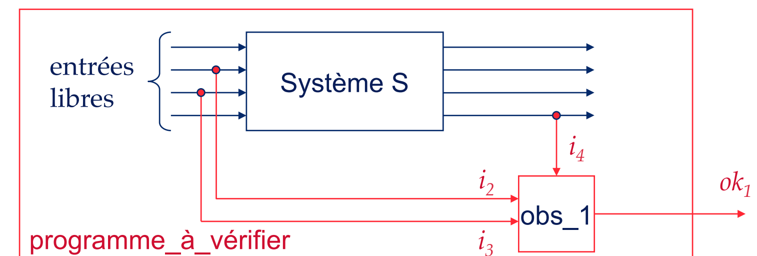
Sûreté et Vivacité (2)

- **Propriété de vivacité :**
  - Le système ***finira*** par effectuer quelque chose de ***bon***
    - *« tout émetteur soumettant une requête à l'arbitre de bus finira par obtenir l'accès au bus »*
  - Vérifiées sur des *séquences infinies* d'états (futur non borné)
    - *Analyse des circuits (ou des composantes fortement connexes) de l'espace d'états*

    → *analyse à partir du **Graphe** des états accessibles et non pas de l'Ensemble des états accessibles*
  - Vérification non implantée dans Lustre/esterel (mais possible avec d'autres) : SPIN / VIS / SMV …

Observateur

- **Observateur**
  - Scrutation des entrées / sorties du système (non intrusif)
  - Automate fini reconnaissant la négation de la propriété, avec une sortie ok, indiquant si l'automate est dans l'état accepteur :
    - *tant que formule non invalidée : ok = true*
    - *si rencontre un état invalidant la formule : ok = false*
  - Connexion synchrone avec le système observé
- **Propriété à vérifier :** `"Dans tous les états : ok` $_1$ `= true"`



Remarque

Parcourir l'espace d'états du programme à vérifier (système + observateur) revient à réaliser le *produit synchronisé* des graphes des états du système et de l'observateur.

Parcours des états accessibles d'un automate produit (représentation explicite) :

1. l'état initial du produit est e = ($init_1$,$init_2$), le marquer « à visiter »
2. pour chaque transition franchissable à partir de e=(e_1,e_2), collecter l'état atteint e'=(e'_1,e'_2), *(transition franchissable de e vers e' : e->e' ssi t_1 = (e_1->e'_1) dans M_1 et t_2 = (e_2->e'_2) dans M_2 et les conditions de franchissement de t_1 et t_2 sont compatibles)*

   2.1. si e' est nouveau, le créer et le marquer « à visiter »
   2.2. insérer la transition e->e'

3. démarquer e.
4. choisir un nouvel état e parmi les états « à visiter » et retourner en 2.
5. si plus d'état « à visiter », retourner l'automate produit.
6. Déterminer si un état de l'automate produit contient : ok=false.

Exemple de vérification

On considère le nœud n1 sur lequel on veut vérifier la propriété
P : « s2 change de valeur à chaque cycle »

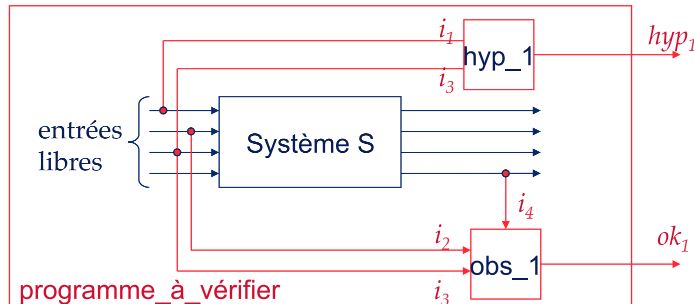
1. Construire O l'observateur représentant la négation de P

2. Construire le produit synchronisé des graphes d'états de n1 et O

3. Déterminez si la propriété P est vérifiée par le nœud n1

Hypothèses sur l'environnement

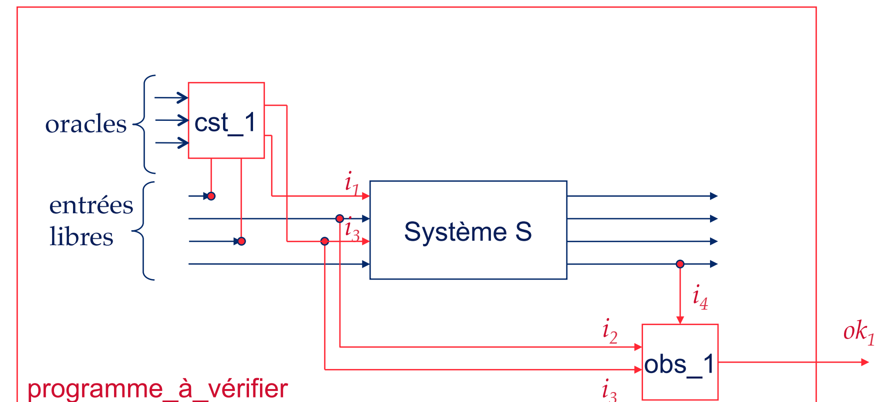
- **En général, une propriété ne doit pas être vérifiée pour un système placé dans un environnement *quelconque*,**
- **il faut préciser les hypothèses respectées par l'environnement**
  - `assert(expr)` en Lustre
  - `expr` est un invariant (automate fini) sur les entrées du système



`"Dans tous les états : (hyp1 n'a jamais été faux => (ok1 = true))"`

Autre approche possible

- **L'hypothèse sur l'environnement est une *contrainte* limitant la variabilité des entrées libres**
- **Propriété :** `"Dans tous les états : ok1 = true"`



oracle : une variable libre supplémentaire permettant de coder le non déterminisme dans les contraintes

UPMC PARIS UNIVERSITAS

Exercice

- **Déterminez si les propriétés suivantes relatives au nœud n1 peuvent être décrites sous la forme d'observateur. Dans l'affirmative, donnez l'observateur associé à la propriété.**
- **Déterminez si la propriété est vraie sur le graphe des états accessibles.**
  - on n'a jamais simultanément s1 et s2
  - s2 ne peut pas être vrai pendant trois cycles consécutifs
  - lorsque a passe de vrai à faux, alors s1 passe à 1 au plus deux cycles plus tard.
  - s1 passera inévitablement à 1 dans moins de trois cycles
  - s1 passera inévitablement à 1 (dans un futur non borné)

UPMC PARIS UNIVERSITAS

c) Equivalences de système

M2 SAR/STL LS

UPMC PARIS UNIVERSITAS

Equivalence observationnelle

- Soient $M_1 = <I_1,O_1,S_1,t_1,g_1,s_{01}>$ et $M_2 = <I_2,O_2,S_2,f_2,g_2,s_{02}>$ deux machines de Mealy, avec $I_1 = I_2 = I$ et $O_1 = O_2 = O$.

- **Sont-elles distinguables du point de vue de leurs entrées / sorties ?**

  Existe-t-il une séquence d'entrées qui, appliquée simultanément sur M_1 et M_2, permet de distinguer les sorties des deux machines ?

Bisimulation :

Soit $\mathcal{R}$ une relation de $S_1 \times S_2$. $\mathcal{R}$ est une **bisimulation** ssi :

- $\forall (q_1,q_2) \in \mathcal{R}, \forall a \in 2^I,$
- $\forall q'_1 \in S_1$ *tq* q_1 -a-> q'_1, $\exists \exists q'_2 \in S_2$ *tq* q_2 -a-> q'_2 *et* $(q'_1,q'_2) \in \mathcal{R}$ *et*
- $\forall q'_2 \in S_2$ *tq* q_2 -a-> q'_2, $\exists \exists q'_1 \in S_1$ *tq* q_1 -a-> q'_1 *et* $(q'_1,q'_2) \in \mathcal{R}$

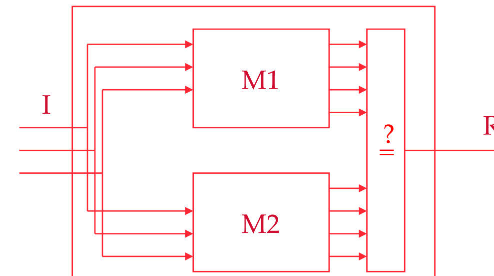
cette définition s'applique à tous systèmes de transitions étiquetés sur transitions, pas nécessairement complets ni déterministes.

$\mathcal{R}$ est une relation d'équivalence.

UPMC PARIS UNIVERSITAS

Equivalence observationnelle

- Mise en œuvre : on parcourt M = $M_1 \| M_2$ la machine produit définie telle que :



M = <I,R,S,f,g,s0>, avec :

$I = I_1 = I_2$

$R = \{0,1\}$

$S \subseteq S_1 \times S_2$

$f_1((s_1,s_2),i) = (f_1(s_1,i) , f_2(s_2,i))$

$g((s_1,s_2),i) = (g_1(s_1,i) = g_2(s_2,i)) = R$

$s_0 = (s_{01}, s_{02})$

- M_1 et M_2 sont équivalents sur les sorties O (à partir de (s_{01},s_{02})) ssi tous les états de M ont pour sortie R = 1.

- Remarque : M = $M_1 \| M_2$ est un produit synchronisé. Les contraintes de synchronisation sont les couples de configurations d'entrée identiques.

Exercice

- **Soient deux nœuds Lustre n1 et n2 définis tels que**

```
node n1 (a,b : bool) return (s1,s2: bool); comme précédemment.

node n2 (a,b : bool) return (s1,s2: bool);
let
  s1 = 0 -> not(a xor pre b));
  s2 = 1 -> not pre s2;
tel
```

- **Construire le programme Lustre permettant de vérifier l'équivalence de ces deux nœuds et donner la propriété à vérifier.**

- **Ces deux systèmes sont-ils équivalents ?**
  - Construire l'automate de n1, de n2
  - Parcourir les deux automates en parallèle et vérifier pour chaque état que n1.s1 = n2.s1 et n1.s2 = n2.s2

Conclusions

M2 SAR/STL LS

Vérification fonctionnelle

- **Model-checking domaine très vaste**
- **Principe simple mais problème d'explosion combinatoire**
  - 80-90 : algorithmes énumératifs
  - 90-00 : algorithmes symboliques à base de BDD et dérivés
    - *Domaine applicatif principal : matériel (programmes booléens)*
  - depuis 2000 :
    - *Model-checking borné : problème SAT puis SMT (Sat Modulo Theory)*
    - *Vérification compositionnelle : équivalence de composants*
    - *Réduction de la taille du système : abstractions*
    - *Convergence avec Interprétation Abstraite*
    - *Domaine applicatif : matériel et logiciel*
  - Autres dimensions (depuis 90) :
    - *Temporisé, probabiliste, hybride, syst. infinis (paramétrés)*
- **Algorithmes et structures de données très sophistiqués**

Autres types de vérification

1. **Model-checking fonctionnel pour d'autres types d'analyse**
   1. Debugging
   2. Mesure de Robustesse
2. **Model-checking temporisé, probabiliste**
3. **Model-checking de systèmes hybrides**
   1. Consommation
   2. Systèmes analogiques / numériques, mécaniques …
4. **Dimensionnement (buffers / paramètres temporels)**
5. **Analyse statique : détection deadlocks (dans les réseaux sur puce)**

Bibliographie

- **Monographies**
  - *Vérification de logiciels, techniques et outils du model-checking. Ph. Schnoebelen. Vuibert, 1999*
  - Formal Methods for Hardware verification, 6th international school on Formal Methods for the Design of Computer, Communication and Software Systems, Bernardo & Cimatti (ed), LNCS 3965, Springer, 2006
  - Handbook on Model Checking, Springer, 2018

- **Revues**
  - Formal Methods in system Design
  - Software Tools and Technology Transfert

- **Actes des conférences**
  - CAV, TACAS, FMCAD, CONCUR, FORMATS, …