Stage MIM1

LSV, ENS Cachan, du 05/06/2001 au 21/07/2001

Laure Danthony

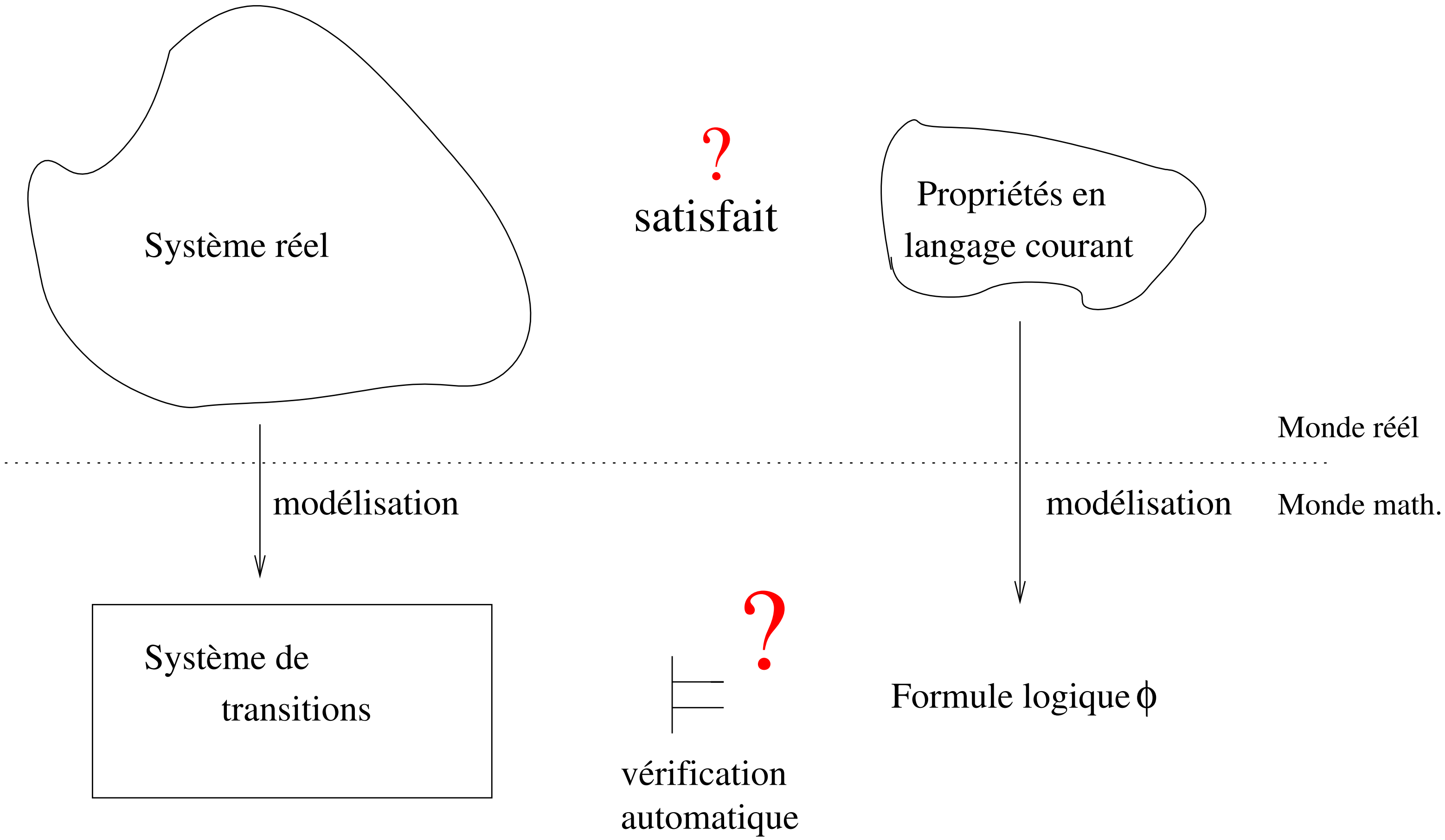
Model Checking symbolique et abstraction

PLAN :

- Définition et problématiques du Model Checking
- SMV, un Model Checker
- Abstraction en Model Checking
- Résultats expérimentaux
- Discours de la méthode

Définitions et problématiques du Model Checking

Problématique de la vérification



Le Model Checking, pour quoi faire ?

- Vérification de logiciels, de programmes
- Combine méthodes exhaustives et automatisation de la vérification
- Outils assez performants dans la pratique

Logiques temporelles

- logique temporelle = cadre formel pour des formules qui tiennent compte du temps
- notations moins lourdes
- restriction de la portée des propriétés
- bon compromis expressivité des formules / complexité de la vérification

Logique temporelle CTL*

- Utilisation des combinateurs $\vee$, $\wedge$, $\neg$, $\Rightarrow$ de la logique classique
- Les **combinateurs temporels X** (next), **U** (until), **F** (forward), ...
- Les **combinateurs de chemin** : **A** (all) et **E** (exists)

Par exemple : $\neg[(\texttt{pierre}\wedge\texttt{rouler})\Rightarrow\texttt{F amasser\_mousse}]$ est un proverbe bien connu !

En pratique, on utilise les restrictions CTL et PLTL de CTL.*

SMV, un Model Checker

Qu'est-ce-qu'un Model Checker ?

- Un outil permettant de vérifier des propriétés logiques d'un système
- quels systèmes ? algos répartis (SPIN), réseaux de Pétri (Design/CPN), automates à variables partagées (SMV), ...
- quelles logiques ? PLTL, CTL, ...
- quelles propriétés ? atteignabilité, sûreté, équité, ...

Toute une gamme d'outils universitaires est disponible

Caractéristiques de SMV

- Cadence SMV, outil universitaire, auteur K. McMillan, Bekerley
- Etats caractérisés par des valeurs de variables (en nombre borné)
- Codage du/des automate(s) par initialisation et transitions des variables
- Vérification de formules PLTL (ou CTL)
- Dans ce cadre ... efficace !

SMV à l'action (1)

- Vérification d'un ascenseur simple, dont les variables sont codées suivant la syntaxe :

```
init(cabine) := 0;

case{
      dir = haut & cabine < MAX : next(cabine) := cabine + 1;
      dir = bas  & cabine > 0   : next(cabine) := cabine - 1;
      1                         : next(cabine) := cabine;
}
```

- Des propriétés du type :

```
ok1[i] : assert G ( (cabine=i) -> (!l_appel[i]) );

ok2[i] : assert G ( l_appel[i] -> ( (G l_appel[i]) | (l_appel[i] U cabine=i)));

str_ok[i] : assert G ( l_appel[i] -> (l_appel[i] U cabine=i) );
```

SMV à l'action (2)

L'interface VW :

Une partie du résultat :

```
Model checking results
======================
ok1[0].............................................true

See file "ex1.warn" for warnings.

user time..........................................0.04 s
system time........................................0 s
```

Abstraction en Model Checking

L'abstraction, pour quoi faire ?

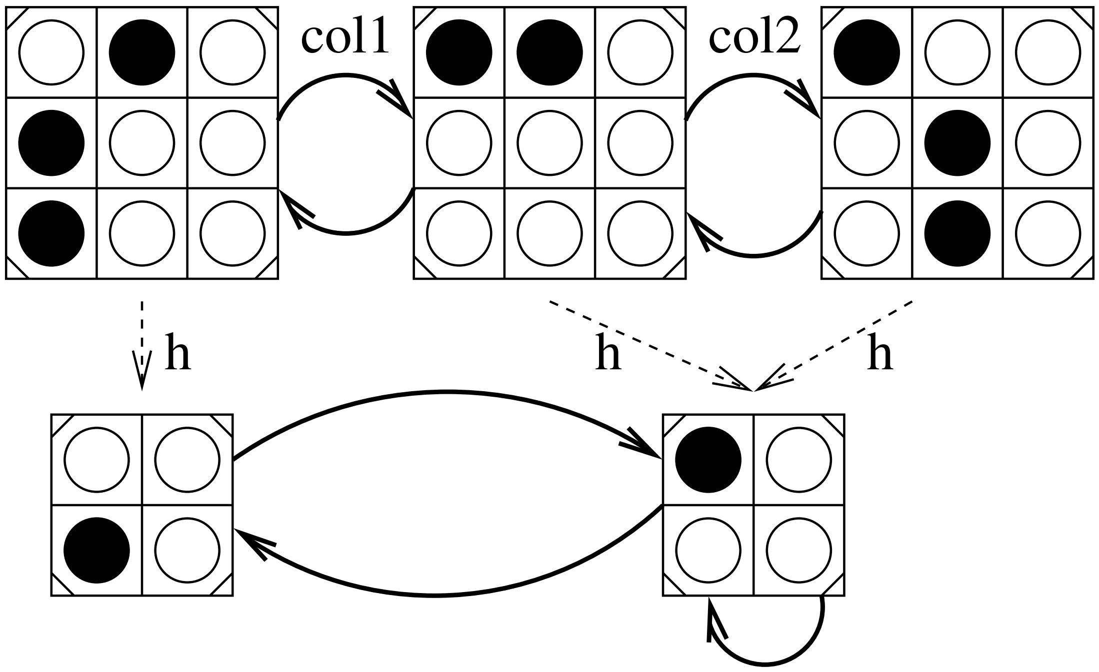
Le problème : $S \models_? \phi$ donne `out_of_memory` …Comment faire ?

- **Solution 1** : Simplification : $S' \models_? \phi'$, sans garantie
- **Solution 2** : Interprétation abstraite : ex intervalles d'évolution de variables mais approximations !
- **Solution 3** : Abstraction : $S' \models_? \phi'$, en garantissant que $S' \models \phi' \Rightarrow S \models \phi$, méthode "supportée" par SMV

Dans ce stage . . .

- Méthode choisie : abstraction.
- **Objectifs du stage** : comprendre ce que fait SMV lors d'une abstraction et réaliser des exemples de taille non triviale pour pouvoir caractériser la méthode (facilité d'emploi, puissance . . .)

Qu'est-ce-que l'abstraction ? - Un exemple intuitif



Intuitivement, lors d'une abstraction ...

– on "colle" des états (diminution du nombre et augmentation du nombre de comportements),
– on ajoute (éventuellement) des transitions.

Qu'est-ce-que l'abstraction ? - Formellement

Soient $M = (D, I, R)$ un système de transitions et une surjection $h : D \to \widehat{D}$. Soit un autre système de transitions $\widehat{M} = (\widehat{D}, \widehat{I}, \widehat{R})$.

DÉFINITION $\widehat{M}$ est une abstraction de M noté $M \sqsubseteq_h \widehat{M}$ si :

1. $\forall d \in D, d \in I \Rightarrow \widehat{d} \in \widehat{I}$,
2. $\forall d_1, d_2 \in D, d_1 R d_2 \Rightarrow \widehat{d_1} \widehat{R} \widehat{d_2}$

Découverte de l'abstraction selon SMV

On *a montré* durant ce stage que :

- la fonction $h = \mathrm{Id}$,
- en SMV l'abstraction est sur chaque variable, autrement dit, pour chaque variable x_i on montre $R_{|x_i} \subseteq \widehat{R_{|x_i}}$,
- cette vérification est bien correcte,
- la validation ne se fait que sur les états atteignables.

Mais surtout : on a mis en place un cadre formel pour la description et le codage de l'abstraction en SMV[a].

[a] C'est plus important que les théorèmes cités plus haut !

Un exemple d'abstraction selon SMV

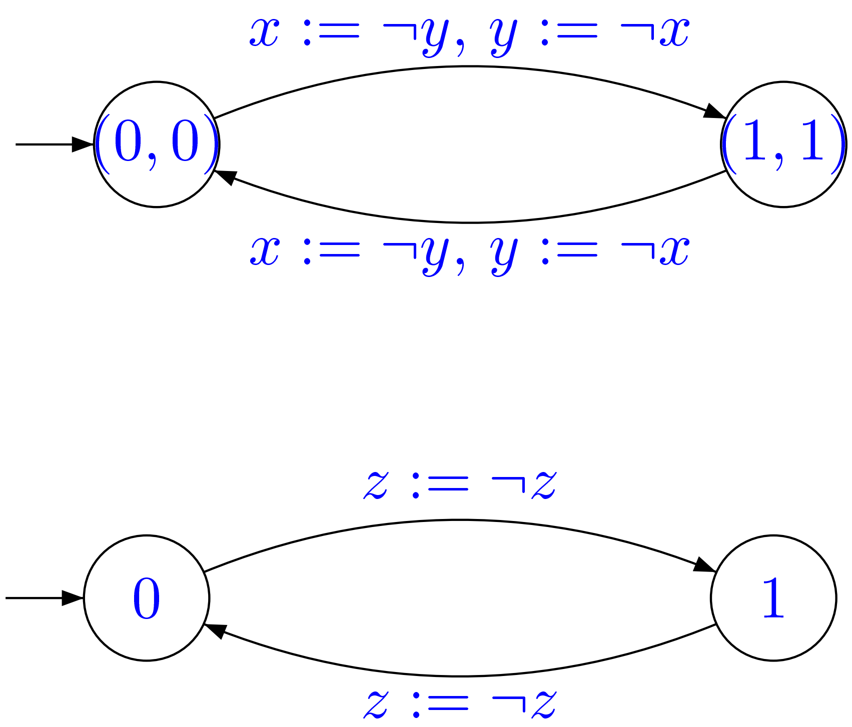


FIG. 1 – Les automates M et $\widehat{M}$

L'exemple en SMV

Déclaration

```
module main() {

/* declaration des variables */
   x,y,z : boolean ;
   z := x & y ;

/* implementation : M */
   init(x) := 0 ;
   init(y) := 0 ;
   next(x) := !y ;
   next(y) := !x ;
```

```
/* abstraction */
   layer M_chapeau : {
      z : boolean ;
      init(z) := 0 ;
      next(z) := !z ; }}
```

Validation de l'abstraction

```
SMV : Prop - Verify z//M_chapeau

Model checking results
======================
z//M_chapeau..................true
```

Que vérifie SMV ?

Vérifier `z//M_chapeau` signifie vérifier :

1. $(x_0 = 0) \wedge (y_0 = 0) \wedge (z_0 = x_0 \wedge y_0) \Rightarrow (z_0 = 0)$
2. $(x' = \neg y) \wedge (y' = \neg x) \wedge (z' = x' \wedge y') \Rightarrow (z' = \neg z)$

. . . sur les automates restreints aux états atteignables !

Résultats expérimentaux

Objectifs

- Réaliser un exemple conséquent qui permet d'atteindre les limites de SMV dans une utilisation "classique"
- Faire un bilan du gain que permet l'abstraction :
  - en terme de temps ...
  - en terme d'espace ...

Réalisation

- Codage d'un ascenseur à deux cabines qui font en sorte d'atteindre tous les étages où elles ont été appelées (l'appel est commun : 1 appel par étage).
- Il n'existe pas de requêtes provenant de l'intérieur d'une cabine.
- La cabine la plus proche de la requête se déplace.
- Nombre d'étages variable pour augmenter le coût des vérifications.
- Utilisation de l'abstraction pour limiter ces coûts.

Résultats

- `time1.smv` sans abstraction, `time2.smv` avec abstraction de 2 variables par étage.
- Gain en temps/nombre d'itérations et en espace (nombre de BDD-nœuds alloués) :

Prop	`time1.smv`	`time2.smv`
`dir_ok1`	0.27 s / 14 it. / 17740 n.	0,26 s / 7 it. / 4113 n.
`hyp_stop`	18,27 s / 637 it. / 239887 n.	3,09 s / 342 it. / 38347 n.
`ok1[2]`	1,49 s / 10 it. / 10036 n.	1,41 s / 6 it. / 4052 n.

- Coût de la validation de l'abstraction : $0,27s$ par variable.

Discours de la méthode

Une démarche expérimentale

- SMV = “boîte noire”dont il s’agit de déterminer le comportement.
- $\Rightarrow$ nombreux tests

Problèmes rencontrés

- Une documentation peu structurée, nombreux exemples sans commentaire cohérent ni explication du cadre d'utilisation de l'abstraction.
- Une définition "personnelle" de l'abstraction non décrite.

Perspectives

- Utilisation de SMV pour la conception d'algorithmes (travail antérieur au code, vérifications de formules simples *forcément vraies*)
- 3 approches de SMV distinctes : abstraction, modularité (pour la spécification, pas la vérification), hypothèses/conclusion. Approfondir leur combinaison.
- ...

Sur le stage. . .

Déroulement

- Stage “classique” de 6 semaines
- Conférence CAV’01 18 $\rightarrow$ 22 juillet 2001

Remerciements

- LSV, laboratoire d'accueil (ENS Cachan)
- Philippe Schnoebelen, CR au LSV, directeur de stage

Enfin . . .

Vous pouvez retrouver le rapport de stage et ces transparents sur la page :

`http://www.ens-lyon.fr/~ldanthon/stageMIM1/stageMIM1_fst.html`