Analyse statique de SCADE par interprétation abstraite

soutenance de stage

Alex AUVOLAT
ANSYS-Esterel Technologies
sous l'encadrement de Jean-Louis Colaço

Juin-Juillet 2014

Introduction

- SCADE : programmation de systèmes embarqués critique
- Forte motivation pour la vérification formelle
- Techniques : interprétation abstraite (Astrée), model checking (DV, Kind, GATeL)
- Objectif du stage : nouvelle technique d'analyse
- Nombreux avantages à analyser directement le code SCADE

Plan

Section 1

Généralités sur SCADE

Programme synchrone

- Le temps est discret
- Toutes les variables évoluent en même temps

Example

```
x = 0 -> (pre y + 1)
y = x + 1
```

Environnements mémoire

Soit un programme SCADE, on note :

- $\mathbb{X}$: ensemble des variables
- $\mathbb{V}$: ensemble des valeurs

Ce modèle simple suffit !

Definition

Environnement mémoire : fonction de $\mathbb{X} \to \mathbb{V} = \mathbb{M}$

Système de transition, fonction de transition

Un programme SCADE représente une fonction de transition :

$$s_0 \xrightarrow{i_1} s_1 \xrightarrow{i_2} s_2 \xrightarrow{i_3} \cdots$$

Autre écriture :

$$s_n = f(c(s_{n-1}) + i_n)$$

Sémantique collectrice

- But : prouver des propriétés générales
- On considère ensemble toutes les entrées possible du programme (première abstraction)
- On considère ensemble tous les moments de l'exécution du programme (seconde abstraction)
- Sémantique collectrice : $S \subset \mathcal{P}(\mathbb{M})$
- Elle peut s'exprimer comme un point fixe :

$$S = lfp_{S_0}(\lambda A.A \cup g(A))$$

Preuve de propriétés

Prouver une propriété P, c'est montrer que $\forall s \in S, s \models P$.

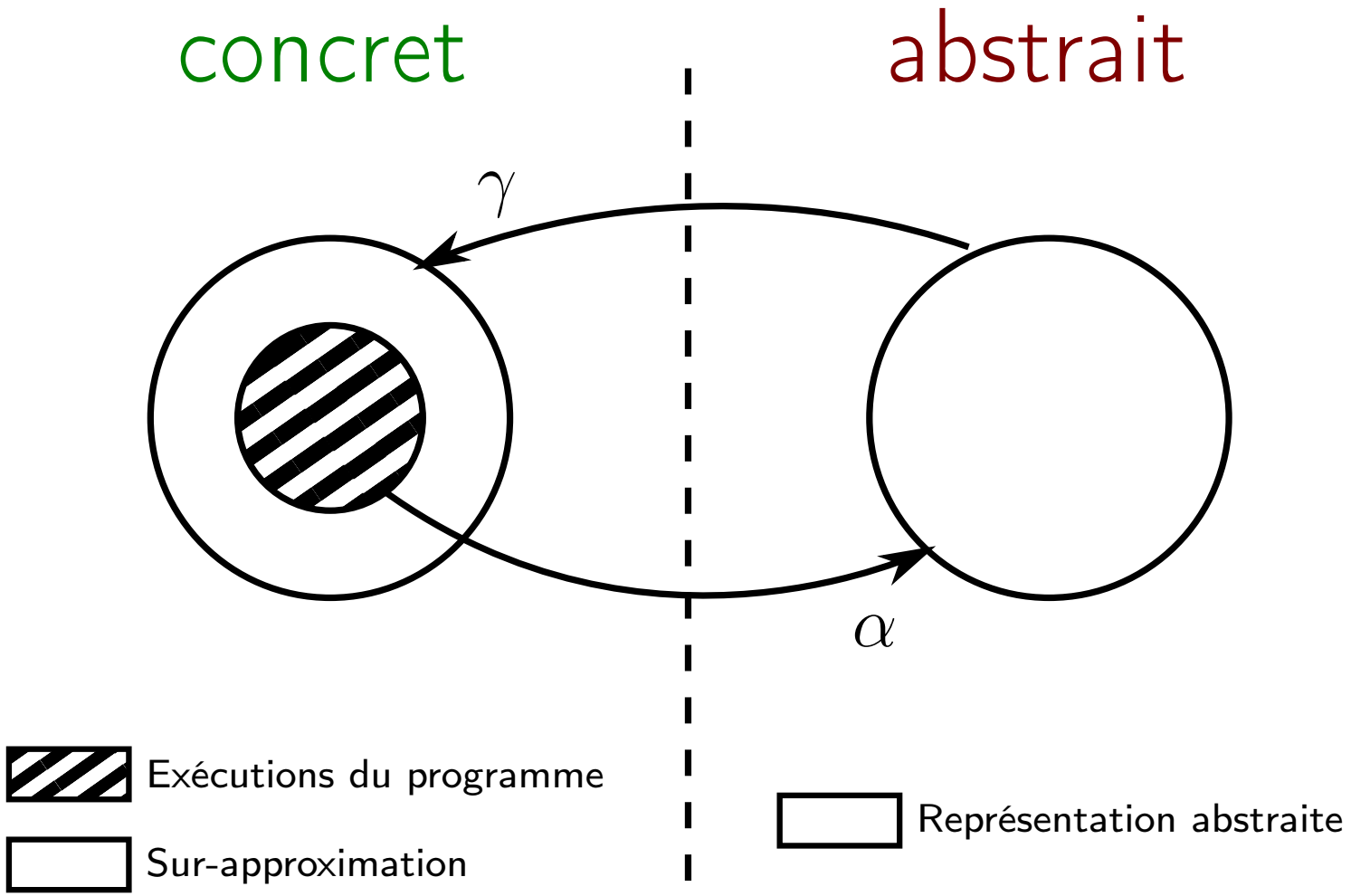
Section 2

Interprétation abstraite

Abstraction

- Pas d'algorithmes sur $\mathcal{P}(\mathbb{M})$ (problème de terminaison)
- Domaine abstrait $\mathcal{D}^\sharp$: représentation pouvant exprimer une classe d'éléments de $\mathcal{P}(\mathbb{M})$.
- Fonction de concrétisation γ : permet de comprendre ce que représente une valeur abstraite.
- Abstraction sûre : $s^\sharp \models s \iff s \subset \gamma(s^\sharp)$

Abstraction



Domaines abstraits numériques

- Plusieurs domaines bien connus : intervalles, polyèdres, octogones.
- Implémentation dans la bibliothèque Apron
- Ne savent gérer que les variables numériques (entières ou rationnelles)

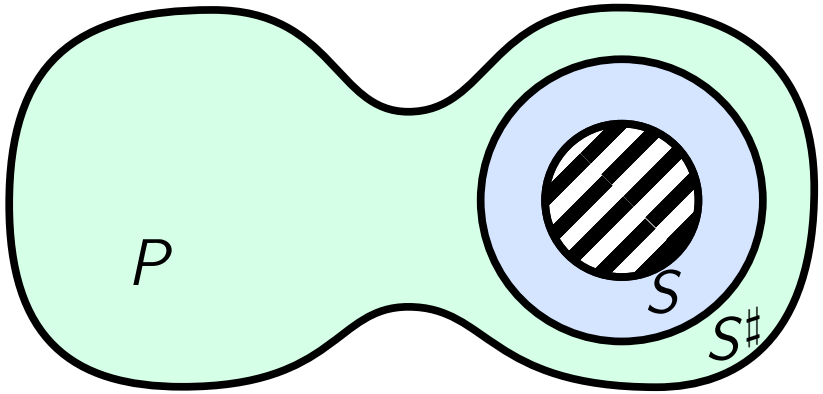
Abstraction de la sémantique collectrice, preuve de propriétés

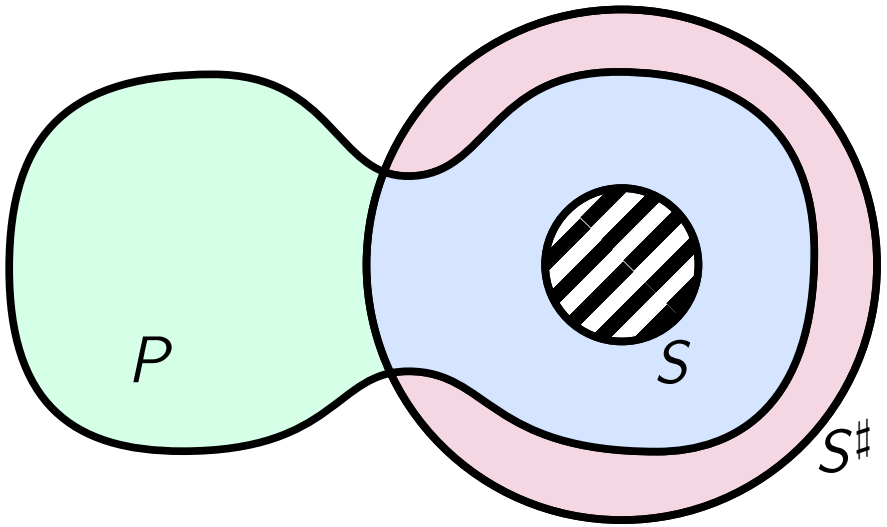
- On abstrait S par $S^\sharp$ tel que $S \subset \gamma(S^\sharp)$.
- $S^\sharp$ est également défini comme un point fixe :

$$S^\sharp = lfp_{S_0^\sharp}(\lambda s.s^\sharp \sqcup g^\sharp(s))$$

- Si on montre que $p = true$ dans $S^\sharp$, c'est gagné.

Preuve de propriété





Section 3

Domaines à disjonction de cas

Besoin d'une abstraction performante

- Problème : le programme est considéré comme un seul lieu
- On veut faire des disjonctions de cas
- On veut étudier les conditions qui passent d'un cas à l'autre
- En résumé : notre programme encode un système de transitions (automate)

Un exemple

Étudions un programme simple :

```
lx = 0 -> pre x
c = (lx >= 10)
x = if c then 0 else lx + 1
```

On aimerait prouver $x \in [0, 10]$

Disjonctions de cas selon des variables

- On se done un ensemble $\mathbb{X}_d$ de variables énumérées
- On considère séparément chaque valuation de ces variables

$$\mathcal{D}_d^\sharp = \mathbb{M}_d \to \mathcal{D}^\sharp$$

- Chaque valuation correspond à un état du système de transition

Exemple

Une représentation appropriée pour traiter notre exemple serait :

init	**c**	
tt	tt	$\bot$
tt	ff	$lx = 0; x = 1$
ff	tt	$lx = 10; x = 0$
ff	ff	$0 \leq lx \leq 9; x = lx + 1$

Principe d'itération

On construit $S^\sharp$ incrémentalement.

- Prendre un cas nouveau q
- Étudier la boucle $q \to q$: on trouve un point fixe précis
- Étudier les nouveaux cas accessibles $q \to q'$.
- Répéter

C'est le principe des *itérations chaotiques*.

Itérations chaotiques sur notre exemple

		init	lx	c	x
0	$\star$	tt	0	ff	1
1		*tt*	0	ff	1
	$\star$	*ff*	1	ff	2
2		*tt*	0	ff	1
		ff	$[1, 9]$	ff	$[2, 10]$
	$\star$	*ff*	10	tt	0
3		*tt*	0	ff	1
	$\star$	*ff*	$[0, 9]$	ff	$[1, 10]$
		ff	10	tt	0
4		*tt*	0	ff	1
		ff	$[0, 9]$	ff	$[1, 10]$
		ff	10	tt	0

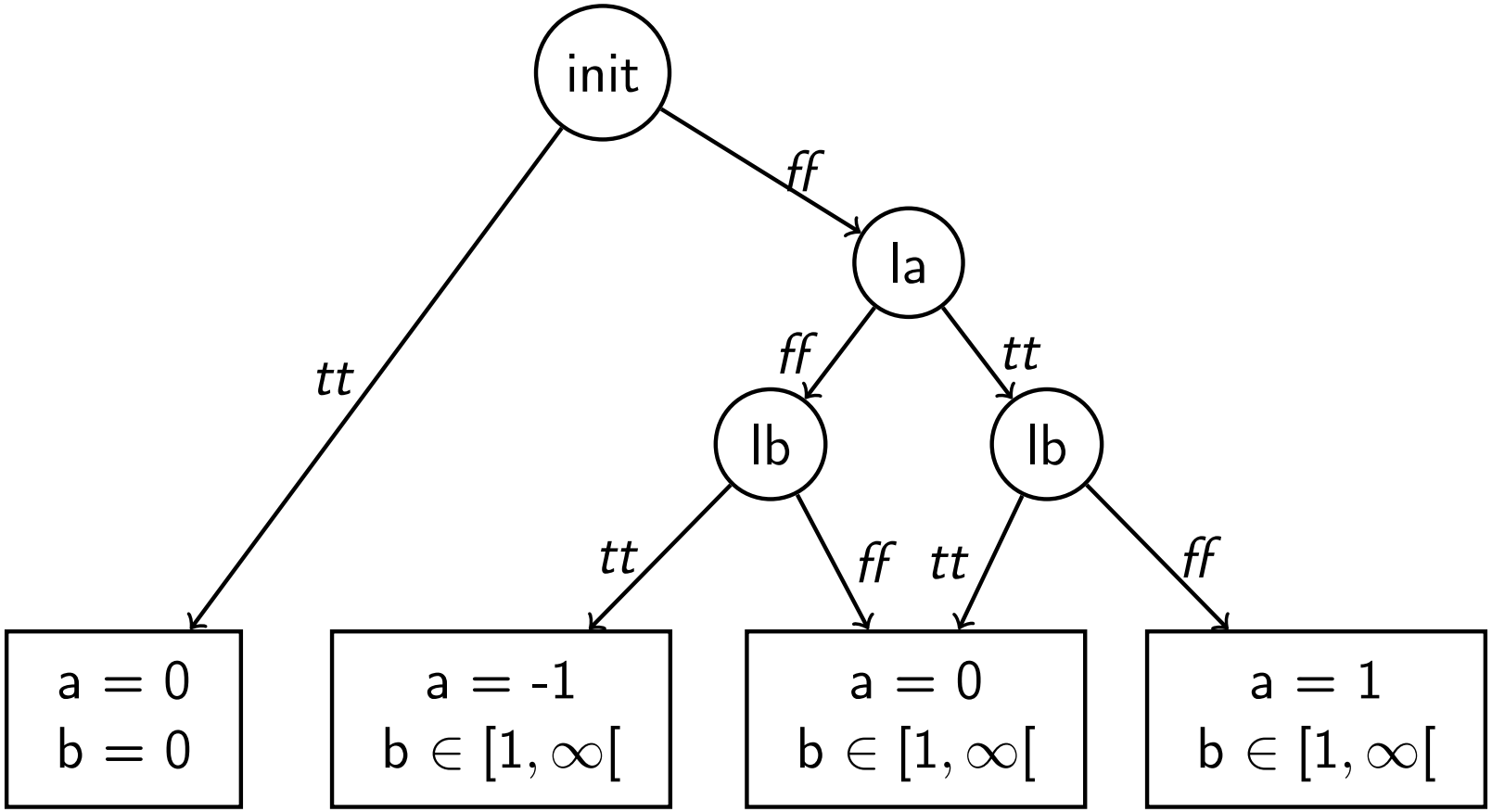
Problème d'explosion

On est souvent confronté à des représentations comme celle-ci :

init	**la**	**lb**	
tt	tt	tt	$a = 0; b = 0$
tt	tt	ff	$a = 0; b = 0$
tt	ff	tt	$a = 0; b = 0$
tt	ff	ff	$a = 0; b = 0$
ff	tt	tt	$a = 0; b \in [1, \infty[$
ff	tt	ff	$a = 1; b \in [1, \infty[$
ff	ff	tt	$a = -1; b \in [1, \infty[$
ff	ff	ff	$a = 0; b \in [1, \infty[$

Il y a beaucoup de redondance !

La solution



Disjonctions de cas avec graphe de décision

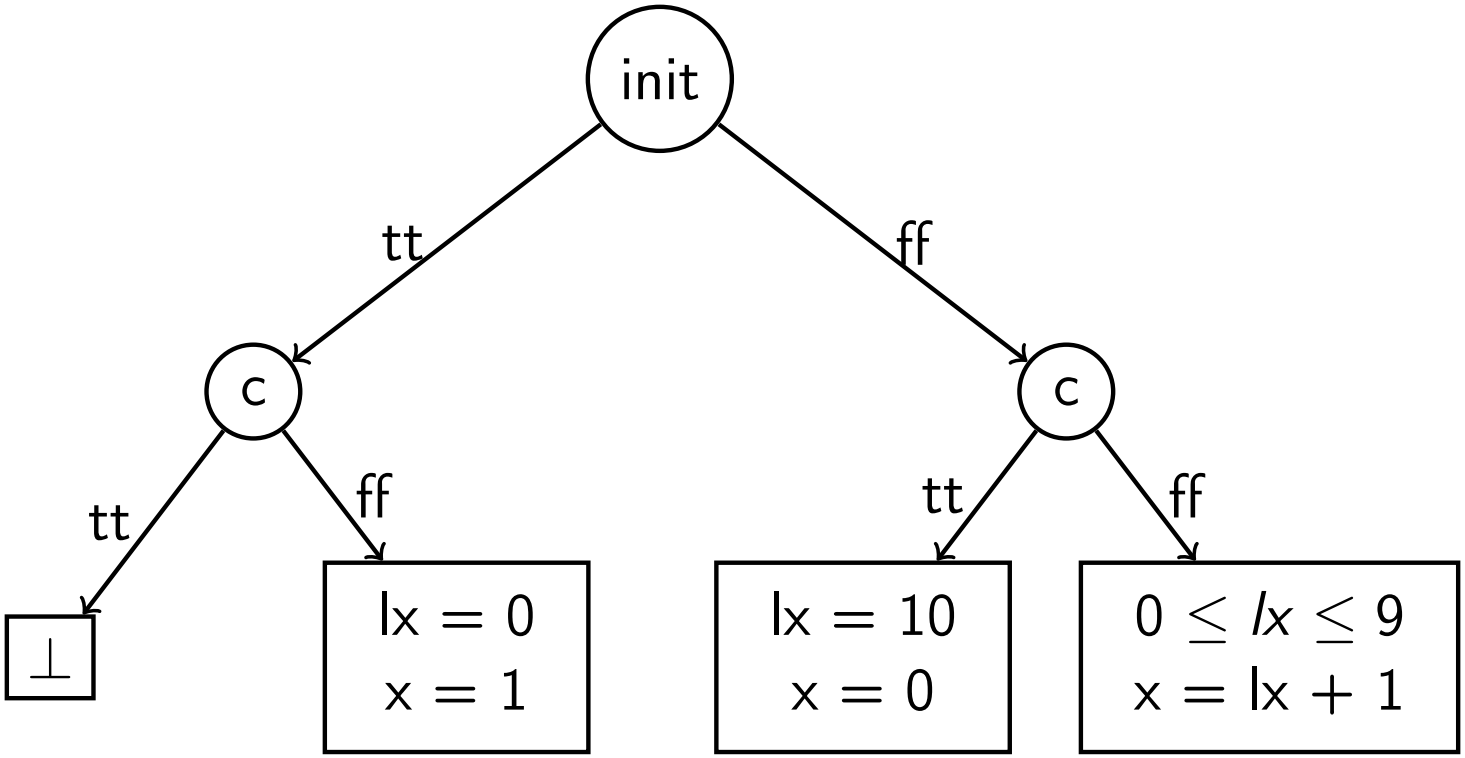
- Graphe de décision, représenté par un type somme :

$$\begin{array}{rcl} \mathsf{T} & := & \mathsf{V}(\mathsf{s}),\ \mathsf{s} \in \mathcal{D}^\sharp \\ & | & \mathsf{C}(\mathsf{x}_i, v_1 \to T_1, \ldots, v_k \to T_k) \end{array}$$

- Unicité de la représentation garantie par une condition simple
- Le partage de sous-graphes permet de gagner en compacité

Retour à notre exemple

Notre exemple peut être représenté par la structure suivante :



Principe d'itération

- Problème : un cas ne correspond plus à une valuation des variables $\mathbb{X}_d$. Que faire ?
- On définit un cas comme étant tous les chemins menant à une feuille donnée.

Section 4

Contribution personnelle

Contribution théorique

- Formalisation de la sémantique concrète de SCADE (d'un sous-ensemble)
- Représentation d'un programme SCADE sous la forme d'une formule logique, adaptée pour l'interprétation abstraite
- Domaine abstrait à base de graphes de décision et itérations chaotiques associées

Implémentation

- Sélection d'un sous-ensemble de SCADE
- Implémentation d'un interprète concret
- Implémentation de la transformation en formule logique
- Implémentation du domaine abstrait à disjonction de cas de façon naïve
- Ré-implémentation avec les graphes de décision

Comparaison avec Astrée

- Astrée : analyse de C généré par SCADE.
- Astrée n'implémente pas d'itérations chaotiques : ça ne convient pas du tout à l'analyse de C
- Astrée implémente un domaine avec des graphes de décision, mais il est moins puissant.
- Astrée implémente de nombreuses techniques permettant de travailler sur de gros programmes (en particulier le variable packing)
- Astrée se pose des questions inutiles en SCADE (typiquement les questions d'initialisation de mémoire)

Résultats

Quelques programmes sur lesquels j'ai pu prouver des propriétés intéressantes :

- compteurs
- updown
- double updown
- suite de cartes
- un train (Halbwachs, 1994)
- un demi-tour pour des rames de metro (Halbwachs, Lagnier, & Ratel, 1992)

Références

Halbwachs, N. (1994, September). About synchronous programming and abstract interpretation. In B. LeCharlier (Ed.), *International symposium on static analysis, sas'94.* Namur (belgium) : LNCS 864, Springer Verlag.

Halbwachs, N., Lagnier, F., & Ratel, C. (1992, September). Programming and verifying real-time systems by means of the synchronous data-flow programming language lustre. *IEEE Transactions on Software Engineering, Special Issue on the Specification and Analysis of Real-Time Systems*.