Analyse statique de SCADE par interprétation abstraite

Alex AUVOLAT

Juillet 2014

Introduction

- On cherche à vérifier des propriétés sur des programmes SCADE (propriétés générales ou propriétés spécifiées dans le code)
- Différentes techniques d'analyse existent : interprétation abstraite (Astrée), model checking (DV, Kind, GATeL)
- Motivation : en faisant l'analyse directement sur SCADE, on se passe d'un certain nombre de problèmes spécifiques à C (modèle mémoire, initialisation, ...)

Plan

Section 1

Généralités sur SCADE

Environnements mémoire

Soit un programme SCADE, on note :

- $\mathbb{X}$ l'ensemble de ses variables ;
- $\mathbb{V}$ l'ensemble des valeurs qu'elles peuvent prendre.

Definition

Un envrionnement mémoire est une fonction de $\mathbb{X} \to \mathbb{V} = \mathbb{M}$

Dans notre étude, on est ammené à étudier des ensembles d'environnements mémoire, c'est-à-dire des éléments de $\mathcal{P}(\mathbb{M})$.
On bénéficie de la structure de treillis de $(\mathcal{P}(\mathbb{M}), \emptyset, \mathbb{M}, \subseteq, \cup, \cap)$.

Système de transition, fonction de transition

Un programme SCADE représente une fonction de transition :

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

On peut réécrire cette sémantique de transition avec deux fonctions : c, qui copie une partie de s_{n-1} dans s_n pour garder des mémoires, et f qui prend ces mémoires ainsi que les entrées pour calculer s_n :

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

Sémantique collectrice

On s'intéresse aux états du programme pour toutes les entrées possibles :

$$g(A) = \{f(c(s) + i), s \in A, i \in \mathbb{I}\}$$

Puis on s'intéresse à tous les états du programme, à tous les cycles :

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

$$S \in \mathcal{P}(\mathbb{M})$$

Où S_0 est l'ensemble des états mémoire initiaux pour le programme.

Preuve de propriétés

La sémantique collectrice S représente une première abstraction de notre programme : c'est l'ensemble de tous les états mémoire accessible par notre programme, quelque soient les entrées, et à tout moment de l'exécution.
Prouver une propriété P, c'est montrer que $\forall s \in S, P(S)$.

Section 2

Interprétation abstraite

Abstraction

Pour pouvoir travailler sur $\mathcal{P}(\mathbb{M})$ de manière algorithmique, c'est-à-dire faire une analyse qui termine, on procède par abstraction.
Pour cela on représente les valeurs dans un domaine abstrait $\mathcal{D}^\sharp$.

Definition

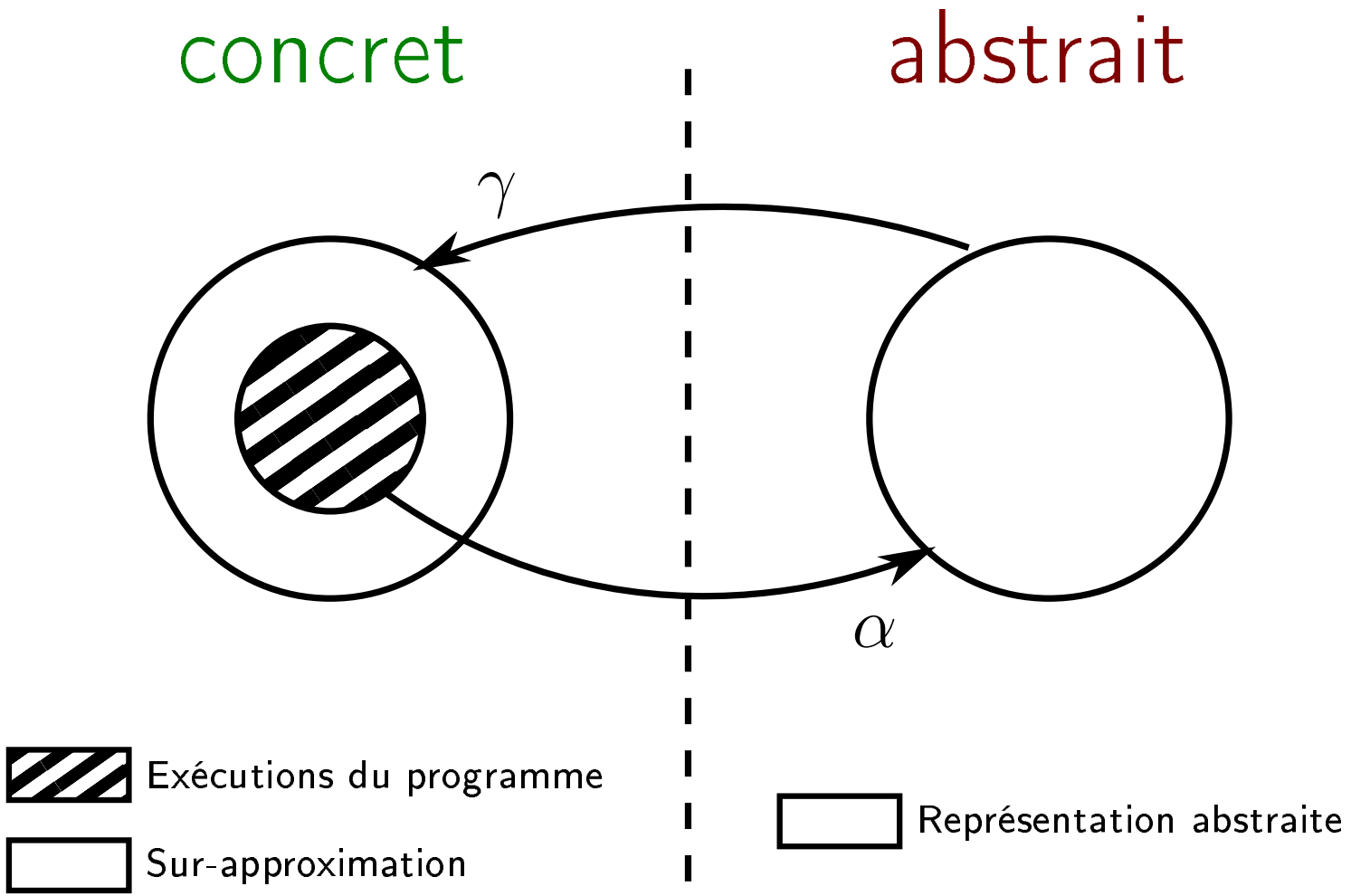
Fonctions d'abstraction et de concretisation :

$$\alpha : \mathcal{P}(\mathbb{M}) \to \mathcal{D}^\sharp$$

$$\gamma : \mathcal{D}^\sharp \to \mathcal{P}(\mathbb{M})$$

Pour représenter $s \in \mathcal{P}(\mathbb{M})$, on utilise toujours une valeur abstraite $s^\sharp$ *sûre*, c'est-à-dire que $s \subset \gamma(s^\sharp)$.

Abstraction



Structure de treillis

On définit deux valeurs abstraites particulières :

- $\bot : \gamma(\bot) = \emptyset$
- $\top : \gamma(\top) = \mathbb{M}$

On définit également des opérateurs sur nos valeurs abstraites :

- $\sqcup : \gamma(s) \cup \gamma(s') \subset \gamma(s \sqcup s')$
- $\sqcap : \gamma(s) \cap \gamma(s') \subset \gamma(s \sqcap s')$

Ce qui munit $\mathcal{D}^\sharp$ d'une structure de treillis : $(\mathcal{D}^\sharp, \bot, \top, \sqsubseteq, \sqcup, \sqcap)$.
On cherche à chaque fois à trouver une meilleure approximation possible de $\gamma(s) \cup \gamma(s')$ ou $\gamma(s) \cap \gamma(s')$, sans jamais rien perdre (propriété de sûreté).

Abstraction non relationnelle

On peut utiliser une abstraction qui traîte chaque variable séparément, en indiquant dans quel intervalle de valeurs elle évolue :

$$\mathcal{D}^\sharp = \mathbb{X} \to itv(\mathbb{Z})$$

($itv(\mathbb{Z})$ est lui-même muni d'une structure de treillis)

Example

Par exemple si on a rencontré les environnements $\{x = 0, y = 5\}$ et $\{x = 1, y = 4\}$, on peut les représenter par la valeur abstraite s telle que :

$$s(x) = [0, 1]$$

$$s(y) = [4, 5]$$

Abstraction relationnelle

On peut utiliser une abstraction qui mémorise des relations entre les variables. La plus célèbre est l'ensemble des polyèdres convexes, qui représente un ensemble d'environnements comme une conjonction d'égalités ou d'inégalités linéaires sur les variables.

Example

Par exemple si on a rencontré les environnements $\{x = 0, y = 5\}$ et $\{x = 1, y = 4\}$, on peut les représenter par la valeur abstraite s qui contient les contraintes suivantes :

$$0 \leq x \leq 1$$

$$y = 5 - x$$

Opérateur de widening

L'opérateur de widening, habituellement noté ∇, est un opérateur crucial pour faire terminer l'analyse des boucles.
Pour les intervalles, il s'agit généralement d'envoyer une borne à l'infini :

$$[1, 3] \nabla [1, 4] = [1, \infty]$$

Pour les polyèdres, c'est plus complexe mais globalement cela peut se comprendre comme la suppression d'une ou de plusieurs contraintes.

Example

Par exemple si on étudie un compteur qui est incrémenté à chaque tour de boucle, on va rencontrer les valeurs $x = 1$, $x = 2$, $x = 3$, ...
L'opérateur ∇ permet d'extrapoler pour pouvoir dire : x varie dans $\mathbb{N}^*$.

Abstraction de la sémantique collectrice, preuve de propriétés

Si on souhaite prouver par exemple qu'une variable booléenne p est toujours vraie, alors on a besoin de montrer que :

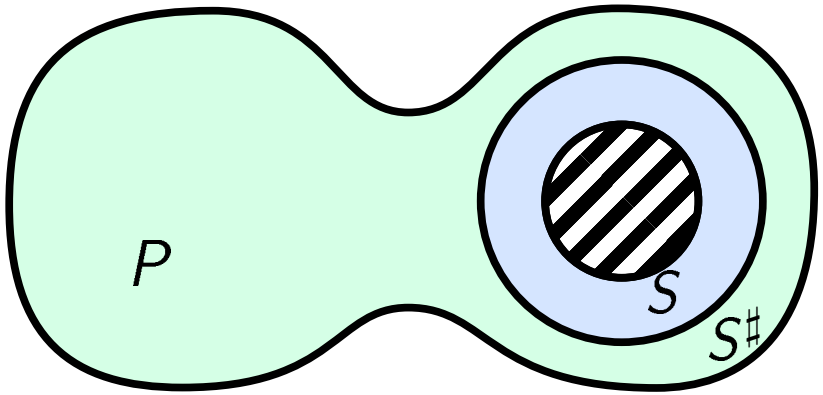
$$\forall s \in S, s(p) = true$$

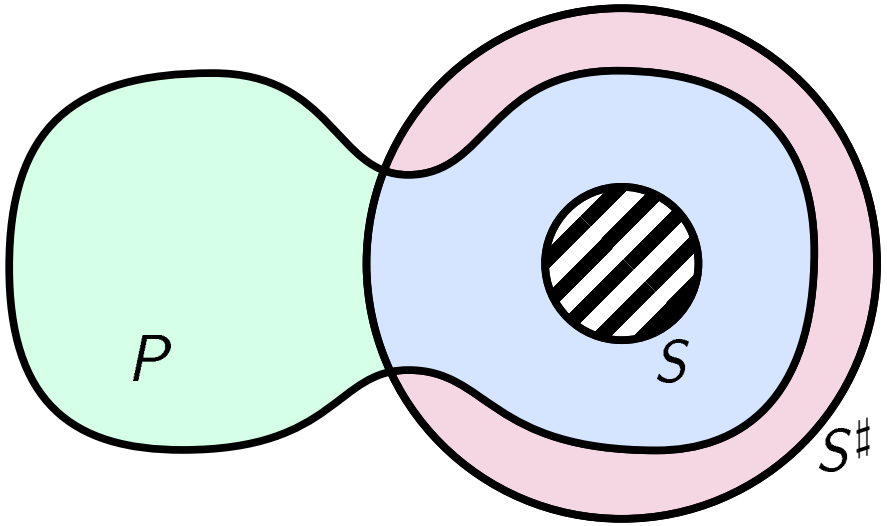
On ne peut pas manipuler S directement, mais on peut en manipuler une abstraction $S^\sharp$ telle que $S \subset \gamma(S^\sharp)$.
$S^\sharp$ est également défini comme un point fixe :

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

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

- On veut être capable de traîter correctement l'initialisation
- On veut être capable d'étudier les états d'automates ou les branches de conditions séparément
- On veut être capable d'utiliser des variables booléennes pour partitionner notre représentation abstraite

Les domaines relationnels à base de polyèdres ou d'octogones ne sont pas à même de représenter des contraintes qui mettent en relation des variables énumérées et des variables numériques.
Il a donc fallu développer de nouveaux domaines abstraits mieux adaptés.

Un exemple

Étudions un programme simple :

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

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

Itérations sur cet exemple

Dans le cas où l'on ne fait pas de disjonctions, nos itérations vont ressembler à ça :

iter	**init**	**lx**	**c**	**x**
0	tt	0	ff	1
1	$\top$	$[0, 1]$	ff	$[1, 2]$
2	$\top$	$[0, 2]$	ff	$[1, 3]$
3	$\top$	$[0, 3]$	ff	$[1, 4]$
4	$\top$	$[0, \infty[$	$\top$	$[0, \infty[$
5	$\top$	$[0, \infty[$	$\top$	$[0, \infty[$

On n'a pas réussi à prouver une borne supérieure sur x !

Disjonctions de cas selon des variables

On se done un ensemble $\mathbb{X}_d$ de variables énumérées (prenant leurs valeurs dans $\mathbb{V}_d$ un ensemble fini).
On note $\mathbb{M}_d = \mathbb{X}_d \to \mathbb{V}_d$.

Definition

On définit le domaine abstrait suivant :

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

Chaque combinaison possible pour les variables de $\mathbb{V}_d$ est traîtée séparément. En pratique tous les cas ne sont pas toujours accessibles.

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	$0 \leq lx \leq 9; x = lx + 1$

Opérateurs correspondants

La fonction de concrétisation donne la sémantique du domaine :

$$\forall s \in \mathcal{D}_d^\sharp, \gamma(s) = \bigcup_{d \in \mathbb{M}_d} \{e \in \gamma(s(d)) \mid \forall x \in \mathbb{X}_d, e(x) = d(x)\}$$

Si on considère une contrainte sur des variables énumérées du type $x \equiv y$, on est capable de la traîter comme suit :

$$[\![x \equiv y]\!](s) = \lambda d. \begin{cases} s(d) & \text{si } d(x) = d(y) \\ \bot & \text{sinon} \end{cases}$$

Les opérateurs $\sqcap, \sqcup$ et les valeurs $\top, \bot$ ne posent pas de problème. Par exemple, $s \sqcup s' = \lambda d.s(d) \sqcup s'(d)$.

Principe d'itération

On pourait faire des itérations globales sur notre environnement de $\mathcal{D}_d^\sharp$ jusqu'à trouver un point fixe, mais pour accélerer l'analyse et pour en améliorer la précision, on considère les cas un par un.

- Prendre un cas nouveau
- Rechercher un point fixe pour ce cas-là ; on profite alors des conditions qui font que l'on reste dans ce cas d'un cycle au suivant, à la manière de conditions de sortie de boucle, pour affiner l'analyse
- Effectuer un cycle complet sans restriction, en partant de ce point fixe, et considérer tous les nouveaux cas que l'on a découvert

C'est le principe des *itérations chaotiques*.

Itérations chaotiques sur notre exemple

Reprenons notre exemple. Maintenant, on a :

		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

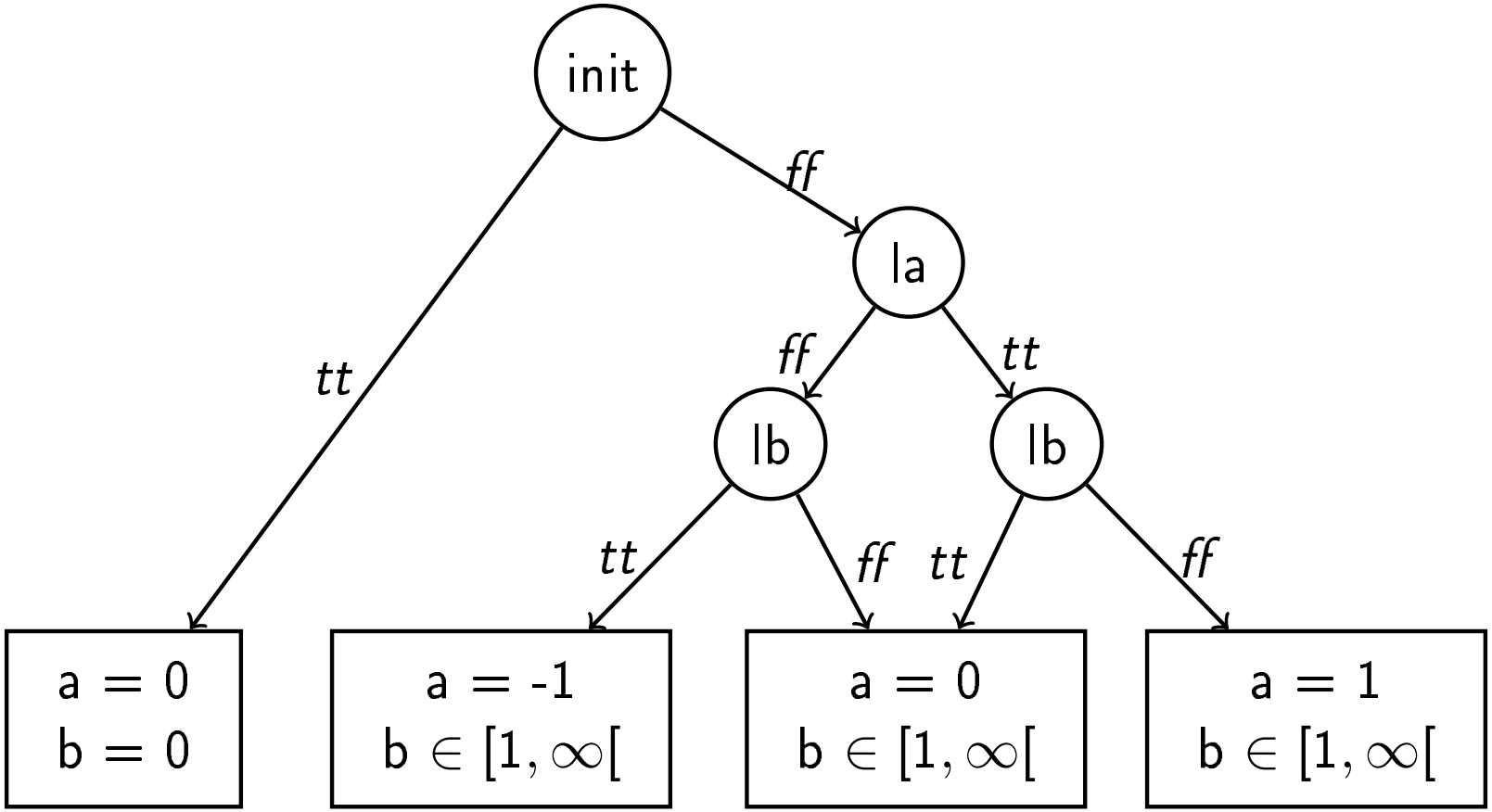
Problème d'explosion

On peut avoir à représenter des valeurs abstraites comme la suivante :

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

On numérote les variables de $\mathbb{V}_d$: $x_1, x_2, \ldots, x_m$.
On considère un type somme qui représente un graphe de décision :

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

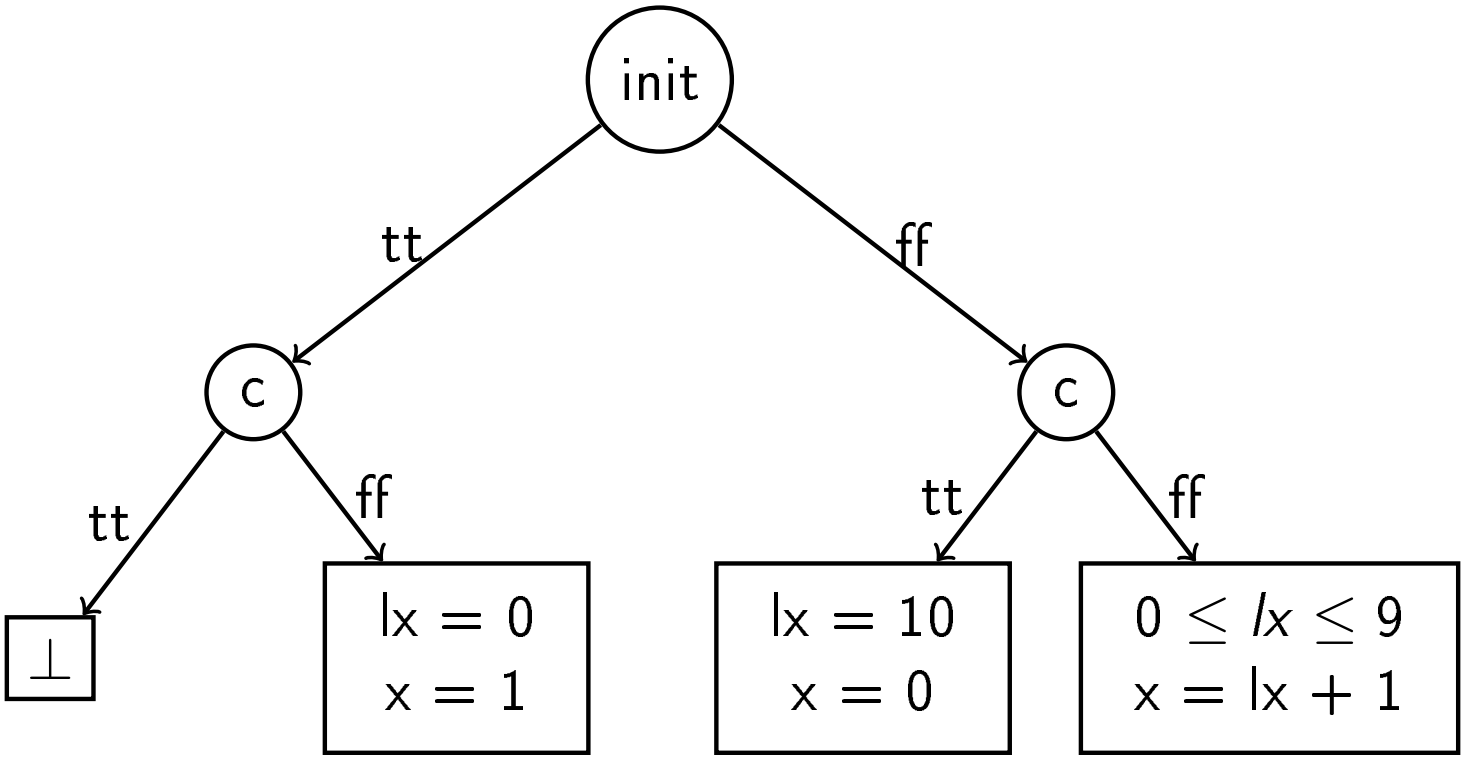
La sémantique est donnée par la fonction de concrétisation :

$$\begin{array}{rcl} \gamma(V(s)) & = & \gamma(s) \\ \gamma(C(x, v_1 \to T_1, \ldots, v_k \to T_k) & = & \bigcup_{i=1}^{k} \{e \in \gamma(T_i) \mid e(x) = v_i\} \end{array}$$

On impose que si un noeud $C(x_i, \ldots)$ est parent d'un noeud $C(x_j, \ldots)$, alors $i < j$, ce qui implique l'unicité de l'arbre.

Retour à notre exemple

Notre exemple peut être représenté par la structure suivante :



Opérateurs correspondants

- On fait du partage sur les feuilles et les sous-arbres identique, ce qui transforme notre structure en un DAG.
- Les opérateurs $\sqcup$ et $\sqcap$ sont implémentés par récurrence, mais il faut utiliser une procédure de mémoisation pour éviter que la complexité soit exponentielle (on profite du partage).
- L'application d'une condition sur les énumérés $x \equiv y$ s'effectue en faisant le $\sqcap$ avec un graphe qui représente cette contrainte.

Principe d'itération

On pourait de même faire des itérations globales, mais on a préféré implémenter les itérations chaotiques pour les mêmes raisons.
Pour identifier les différents cas, on utilise la fonction suivante, qui extrait d'un graphe la fonction booléenne sur les variables énumérées qui mène à une feuille $V(s_0)$ (partagée) :

$$\begin{aligned} \rho(V(s), s_0) &= \begin{cases} \top & \text{si } s = s_0 \\ \bot & \text{sinon} \end{cases} \\ \rho(C(x_i, v_1 \to T_1, \dots), s_0) &= C(x_i, v_1 \to \rho(T_1, s_0), \dots) \end{aligned}$$

Section 4

Résultats et conclusion

Comparaison avec Astrée

- Astrée implémente un domaine abstrait permettant de traiter différents cas dans un arbre de disjonctions, mais les feuilles de cet arbre ne peuvent être que des intervalles de valeurs (ils ne gèrent pas les relations entre les variables). En conséquence, une propriété comme celle du double updown est difficile à montrer.
- Dans Astrée, les variables sont considérées par *packs*. Pour les octogones, des packs sont générés automatiquement. Pour l'arbre de disjonctions, il faut spécifier manuellement que l'on veut que l'analyseur considère un pack comportant telles et telles variables. (L'ensemble des propriétés prouvées est la conjonction des propriétés prouvées à l'aide de chaque pack).
- Astrée travaille sur du C, et se pose des questions inutiles (typiquement les questions d'initialisation ou de bornes)

Implémentation

$\exists$ prototype.

Résultats

Plein de preuves sympathiques :

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

Limitations

- Pour certains problèmes, il faut rajouter manuellement des variables booléennes pour faire des disjonctions de cas
- On ne sait pas comment ça se comporte sur des gros programmes
- Aucune analyse des propriétés des nombres flottants
- Analyseur non vérifié : il manque quelques preuves concernant les abstractions (structures de treillis, connection de Galois, correction et terminaison des principes d'itérations chaotiques)

Perspectives

- Développement d'une heuristique qui détermine quelles variables représentent la structure de contrôle du programme et sont pertinentes à considérer
- Partitionnement dynamique
- Analyses par packs de variables, comme Astrée

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.*