Projet RNTL Averiles
Fourniture F1.1: Modèles

1 Problématique

Le projet AVERILES a comme objectif l´analyse des programmes avec mémoire dynamique, en utilisant des différents aspects de la théorie des automates étendus (automates à compteurs, automates d´arbre, etc.). Le programme d´entrée est décrit utilisant un sous-ensemble du langage C. Ce programme est ensuite traduit vers un automate étendu avec des opérations sur la mémoire (allocation et effacement des cellules, modification des champs sélecteurs, etc.). L´automate étendu résultant est analysé a l´aide des différents outils, développés dans le cadre du projet.

Comme décrit par la suite, on considère deux niveaux d´abstraction dans la définition de modèle:

1. un modèle concret, qui est un sous-ensemble rigoureusement défini du langage ANSI C, et

2. un modèle abstrait, qui décrit des automates étendues avec des opérations sur la mémoire.

Tout programme analysé par la plateforme AVERILES sera fourni dans le sous-ensemble de C défini par la section 2. Ce langage est spécifié principalement par sa syntaxe abstraite.

Le langage de description des automates étendues décrit en détail dans la section 3, permet de définir des structures de graphe de flot de contrôle, où les noeuds sont des points de contrôle et les arcs sont étiquetés avec des instructions de programme. Dans notre cas, les instructions sont de la forme condition-affectation, dont la syntaxe est très simple.

Le modèle d´automate étendu est utilisé par la suite comme point de référence pour les trois outils d´analyse qui ont été développés dans le cadre du projet: L2CA, ARTMC (développés à Verimag) et TOPICS (développé au LSV). Chaque de ces outils a été conçu pour traiter une classe particulière de programmes. Notamment, L2CA et TOPICS traitent les programmes qui manipulent les listes simplement chaînées, tandis que ARTMC traite les programmes avec des structures arborescentes, et même des structures plus

complexes que les arbres, comme les listes doublement chaînées, les arbres dont les feuilles sont chaînées dans une liste, etc. Par conséquent, chaque outil d´analyse utilise un sous-ensemble des instructions spécifiées dans le modèle des automates étendues.

2 Langage d´entrée C restreint

Le langage d´entrée de la plate-forme AVERILES fait l´objet de cette section. Ce langage est un sous-ensemble strict du langage C ANSI, qui retient les aspects les plus intéressants du point de vue de l´analyse de la mémoire. Le langage est décrit formellement par sa syntaxe abstraite, son système de type et sa sémantique opérationnelle.

2.1 Syntaxe

Un programme décrit dans le langage C restreint consiste dans une liste de déclarations et de définitions. Notamment, on encontre les déclarations de types, variables globales et fonctions, ainsi que les définitions des fonctions.

program	::=	$\left\{ \textit{ declaration } \right\}^*$
declaration	::=	*type-declaration*
	|	*var-declaration*
	|	*function-declaration*
	|	*function-definition*
type-declaration	::=	**typedef** *type-name* ***** *identifier* **;**
	|	**typedef struct** *identifier* **{** $\left\{$ struct-field $\right\}^*$ **}** ***** *identifier* **;**
struct-field	::=	*type-name identifier* **;**
	|	**struct** *identifier* ***** *identifier* **;**
type-name	::=	**int**
	|	*identifier*
var-declaration	::=	*type-name identifier* $\left\{$ **,** *identifier* $\right\}^*$ **;**
function-declaration	::=	*return-type identifier* **(** $\left[$ *fpar* $\left\{$ **,** *fpar* $\right\}^*$ $\right]$ **)** **;**
return-type	::=	*type-name*
	|	**void**
fpar	::=	*type-name identifier*
function-definition	::=	*return-type identifier* **(** $\left[$ *fpar* $\left\{$ **,** *fpar* $\right\}^*$ $\right]$ **)** *block*

Remarque 1 Pour le terme *type-declaration*, dans le cas **typedef struct** *identifier*, nous autorisons un *struct-field* de la forme **struct** *id1* ***** *id1* seulement

si *id1* est l'*identifier* dans le terme **typedef struct** *identifier* considéré.

Remarque 2 Chaque champ (ou sélecteur) apparaissant dans les structures de données doit porter un nom différent, par exemple il n'est pas possible de déclarer deux structures de liste qui ont toutes deux un champ dénommé `next`.

Remarque 3 Il n'est pas possible d'utiliser un type qui n'a pas été déclaré auparavant, sauf dans le cas exceptionnel des structures de données récursives comme l'indique la remarque 1.

Les déclarations de types sont utilisées pour introduire des nouveaux types dans le programme. On peut notamment déclarer des types de tableaux, ainsi que de types de structure. Chaque type introduit par une telle déclaration est un type de pointeur. Autrement dit, une variable du type tableau est en effet un pointeur vers un vecteur, et une variable du type structure est un pointeur vers une structure. Afin d´utiliser ces variables, elles doivent être initialisées par des opérations d´allocation dynamique. A part les variables de type pointeur, un programme peut utiliser des variables du type entier.

La partie opérationnelle du programme se trouve dans les définitions des fonctions. Chaque fonction doit être déclarée avant son appel, raison pour laquelle on trouve dans la syntaxe des déclarations de fonctions. A présent on interdit les appels récursives de fonctions. L´enlèvement de cette restriction fait objet des travaux en cours.

2.1.1 Instructions et Expressions

L´ensemble des instructions qu´on peut utiliser dans un programme C restreint se partage entre:

- *Affectations* Une affectation comporte une partie gauche, qui peut être soit une variable, soit une référence dans un tableau, soit un accès a une structure, et une partie droite qui est une expression, décrite par la suite.

- *Gestion de mémoire dynamique* Une instruction d´allocation (`malloc`) prend comme argument une expression particulière de la forme $N * S$ indiquant le nombre de cellules N de taille S qui seront allouées, et retourne un pointeur à la zone de mémoire allouée. Une instruction d´effacement (free) prend comme argument un pointeur et efface la zone de mémoire référencée. Dans le cas ou le pointeur n´a pas été alloué auparavant, l´exécution du programme se termine avec une erreur.

- *Flot de contrôle* On considère des structures conditionnelles (`if`) et des boucles (`while`) avec les instructions de saut conditionnel (break,

continue) et inconditionnel (goto). Les structures de contrôle `if` et `while` utilisent des expressions booléennes. Notons de plus que lorsque qu'un goto est utilisé, il ne peut faire référence qu'à un label situé dans le même bloc.

block ::= **{** { *statement* }* **}**

statement ::= *var-declaration*
| /* empty */ **;**
| *lvalue-expression* **=** *rvalue-expression* **;**
| *lvalue-expression* **=** **malloc** **(** *malloc-expression* **)** **;**
| **free** **(** *identifier* **)** **;**
| [*lvalue-expression* **=**] *identifier* **(** [*term* { **,** *term* }*] **)** **;**
| **break** **;**
| **continue** **;**
| **goto** *label* **;**
| **return** [*term*] **;**
| **if** **(** *boolean-expression* **)** *statement*
| **if** **(** *boolean-expression* **)** *statement* **else** *statement*
| **while** **(** *boolean-expression* **)** *statement*
| *label* **:** *statement*
| *block*

malloc-expression ::= **sizeof** **(** **struct** *identifier* **)**
| *integer* ***** **sizeof** **(** *type-name* **)**
| **sizeof** **(** *type-name* **)** ***** *integer*

index-expression ::= *integer*
| *identifier*

lvalue-expression ::= *identifier*
| *identifier* **[** *index-expression* **]**
| *identifier* **->** *identifier*

term ::= *lvalue-expression*
| *integer*
| **NULL**

rvalue-expression ::= *term*
| *term* **+** *term*
| *term* **-** *term*

boolean-expression ::= *term* **==** *term*
| *term* **!=** *term*
| *term* **<=** *term*
| *term* **>=** *term*
| *term* **<** *term*
| *term* **>** *term*
| **any**
| **!** *boolean-expression*
| *boolean-expression* **&&** *boolean-expression*
| *boolean-expression* **||** *boolean-expression*
| **(** *boolean-expression* **)**

3 Langage de description pour les automates étendus

Un automate étendu est un graphe de flot de contrôle où les noeuds sont des points de contrôle et les arcs sont étiquetés par des instructions de type condition-action, spécifiées dans le langage décrit par la suite. Les variables dans ce langage se divisent en quatre types: *tableaux*, *pointeurs*, *sélecteurs* et *entiers*.

Les variables de type tableau peuvent apparaître dans des allocations (`malloc`) et effacements (`free`) de mémoire, ainsi que dans des expressions d´accès, ou l´index dans le tableau est donné par une variable entière.

Les variables de type pointeur peuvent apparaître dans des allocations (`malloc`) et effacements (`free`) de mémoire, ainsi que dans des expressions de déréferencement, $p \rightarrow s$, où s doit être une variable sélecteur.

Les variables entiers peuvent apparaître dans des expressions arithmétiques. En particuliers, les gardes des actions sont des combinaisons booléennes de comparaisons d´expressions arithmétiques.

$$
\begin{array}{lcl}
t, t', \ldots & \in & \text{ArrayVariables} \\
p, p', \ldots & \in & \text{PointerVariables} \\
s, s', \ldots & \in & \text{SelectorVariables} \\
i, i', \ldots & \in & \text{IntegerVariables}
\end{array}
$$

$$
\begin{array}{lcl}
rule & := & guard \ ? \ action \\
guard & := & test \mid guard \wedge guard \mid guard \vee guard \mid \neg guard \\
test & := & expr = expr \mid expr \neq expr \mid expr < expr \mid expr > expr \mid \\
 & & expr \leq expr \mid expr \geq expr \\
action & := & lval := expr \mid t := \text{malloc}(nexpr) \mid p := \text{malloc} \mid \text{free}(p) \mid \text{free}(t) \\
lval & := & t \mid p \mid i \mid t[iexpr] \mid p \rightarrow s \\
nexpr & := & n \in \mathbb{Z} \\
iexpr & := & n \in \mathbb{Z} \mid i \\
expr & := & lval \mid n \in \mathbb{Z} \mid \text{null} \mid expr + expr \mid expr - expr
\end{array}
$$

La traduction à partir du C restreint vers un automate étendu (au format XML), dans lequel chaque transition est étiquetée par une règle $rule$, est réalisée automatiquement par l'outil C2XML. Cet outil a été développé en JAVA en utilisant le lexer JFLEX et le parser JavaCUP. Il prend en entrée un fichier au format C restreint ainsi que le nom de la fonction (présente dans le fichier) dont on souhaite obtenir la représentation sous forme d'un automate. Si l'outil détecte un appel récursif de fonctions, il arrête son exécution de même si le fichier donné en entrée ne respecte pas la syntaxe du C restreint.