Rapport technique du projet AVERILES : Ajout à la fourniture F1.1 Description Modèle

Arnaud Sangnier

9 septembre 2008

1 Introduction

Ce document présente la syntaxe du C essentiel concurrent traité dans le projet AVERILES ainsi que le format interne correspondant. Ce document fait suite à la fourniture F1.1 en étendant la syntaxe avec les aspects concurrence (utilisant des primitives *pthreads* de *POSIX*)

2 Le C essentiel concurrent

2.1 Déclarations

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**
	|	**pthread_t**
	|	**pthread_mutex_t**
	|	*identifier*
var-declaration	: :=	*type-name identifier* $\left\{$ **,** *identifier* $\right\}^*$ **;**
function-declaration	: :=	*return-type identifier* **(** $\left[$ *fpar* $\left\{$ **,** *fpar* $\right\}^*$ $\right]$ **) ;**
	|	**void *** *identifier* **(** **void *** *identifier* **) ;**
return-type	: :=	*type-name*
	|	**void**
fpar	: :=	*type-name identifier*
function-definition	: :=	*return-type identifier* **(** $\left[$ *fpar* $\left\{$ **,** *fpar* $\right\}^*$ $\right]$ **)** *block*
	|	**void *** *identifier* **(** **void *** *identifier* **)** *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.

2.2 Instructions

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

statement : := *var-declaration*

|	/* empty */ **;**
|	*identifier* = **(** *type-name* ***** **)** *identifier* **;**
|	*identifier* = **(** **struct** *type-name* ***** **)** *identifier* **;**
|	*lvalue-expression* == *rvalue-expression* **;**
|	*identifier* == **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*
|	*identifier* == **pthread_create** **(** **&***identifier* **,** **NULL** **,** *identifier* **,** **NULL** **)** **;**
|	*identifier* == **pthread_create** **(** **&***identifier* **,** **NULL** **,** *identifier* **,** **(** **void *** **)** *identifier* **)** **;**
|	*identifier* == **pthread_join** **(** *identifier* **,** **NULL** **)** **;**
|	*identifier* == **pthread_mutex_init** **(** **&***identifier* **,** **NULL** **)** **;**
|	*identifier* == **pthread_mutex_destroy** **(** **&***identifier* **)** **;**
|	*identifier* == **pthread_mutex_lock** **(** **&***identifier* **)** **;**
|	*identifier* == **pthread_mutex_trylock** **(** **&***identifier* **)** **;**
|	*identifier* == **pthread_mutex_unlock** **(** **&***identifier* **)** **;**

2.3 Expressions

malloc-expression	: :=	*size-expression-point*
	|	*integer* ***** *size-expression-tab*
	|	*size-expression-tab* ***** *integer*
size-expression-point	: :=	**sizeof** (**struct** *identifier*)
size-expression-tab	: :=	**sizeof** (*type-name*)
index-expression	: :=	*integer*
	|	*identifier*
lvalue-expression	: :=	*identifier*
	|	*identifier* [*index-expression*]
	|	*identifier* -> *identifier*
term	: :=	*lvalue-expression*
	|	*integer*
	|	**NULL**
rvalue-expression	: :=	*term*
	|	*term* + *term*
	|	*term* - *term*
	|	**any**
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*)

2.4 Tokens

identifier	: :=	[a-zA-Z][a-zA-Z0-9]*
integer	: :=	$[0\text{-}9]^+$
label	: :=	[a-zA-Z][a-zA-Z0-9]*

3 Format Interne Averiles

La syntaxe du format interne AVERILES peut être résumée de la façon suivante :

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

$$
\begin{array}{lcl}
rule & := & guard\ ?\ action | create | join \\
guard & := & test \mid guard \wedge guard \mid guard \vee guard \mid \neg guard \\
test & := & expr = expr \mid expr \neq expr \mid expr < expr \mid expr > expr \mid \\
 & & expr >= expr \mid expr <= expr \\
action & := & lval := expr \mid t := \text{malloc}(nexpr) \mid p := \text{malloc} \mid \text{free}(p) \mid \text{free}(t) \\
 & & \mid i := mutex - init(m) \mid i := mutex - lock(m) \\
 & & \mid i := mutex - unlock(m) \mid i := mutex - trylock(m) \\
 & & \mid i := mutex - destroy(m) \\
lval & := & t \mid p \mid i \mid t[iexpr] \mid p \rightarrow s \mid m \\
nexpr & := & n \in \mathbb{Z} \\
iexpr & := & n \in \mathbb{Z} \mid i \\
expr & := & lval \mid n \in \mathbb{Z} \mid \text{null} \mid expr + expr \mid expr - expr
\end{array}
$$

La traduction à partir du C essentiel concurrent vers un automate étendu (au format intern), dans lequel chaque transition est étiquetée par une règle *rule*, est réalisée automatiquement par l'outil C2AVERILES. 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 essentiel 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 essentiel.