

### 3.1 Introduction

Notre approche est fondé sur l'idée de traduire les programmes écrit en C vers le Promela afin de leur appliqués le spin model-checker. Bien sûr nous nous sommes limité à un sous-ensemble de langage C, qui sera présenté ainsi que sa grammaire. Nous exposons aussi, la grammaire de Promela qui est la langue input du Spin, tout en se limitons aux règles nécessaires.

Le principe de la traduction proposé est dûment expliqué et détaillé, tout en focalisant sur les contributions apportées pour contourner certaines limitations du Promela vis-à-vis des structures de données composées telles que les tableaux multidimensionnels ainsi que l'appel des fonctions récursives, et l'atomicité des séquences de codes.

### 3.2 Le Langage C

#### 3.2.1 Introduction

Le langage C reste un des langages les plus utilisés actuellement. C'est parce que le langage C est un langage comportant des instructions et des structures de haut niveau, tout en générant un code très rapide grâce à un compilateur très performant.

Selon le TIOBE (*programming community Index*) spécialisé dans l'évaluation de la popularité des langages de programmation, C est classé premier sur la liste des langages de programmation les plus populaires en May 2012 (voir le tableau 3.1 & Figure 3.1).

Position May 2012	Position May 2011	Delta in Position	Programming Language	Ratings May 2012	Delta May 2011	Status
1	2	↑	C	17.346%	+1.18%	A
2	1	↓	Java	16.599%	-1.56%	A
3	3	=	C++	9.825%	+0.68%	A
4	6	↑↑	Objective-C	8.309%	+3.30%	A
5	4	↓	C#	6.823%	-0.72%	A
6	5	↓	PHP	5.711%	-0.80%	A
7	8	↑	(Visual) Basic	5.457%	+0.96%	A
8	7	↓	Python	3.819%	-0.76%	A
9	9	=	Perl	2.805%	+0.57%	A

Tableau 3.1: Classement des langages de programmation les plus populaires [19]

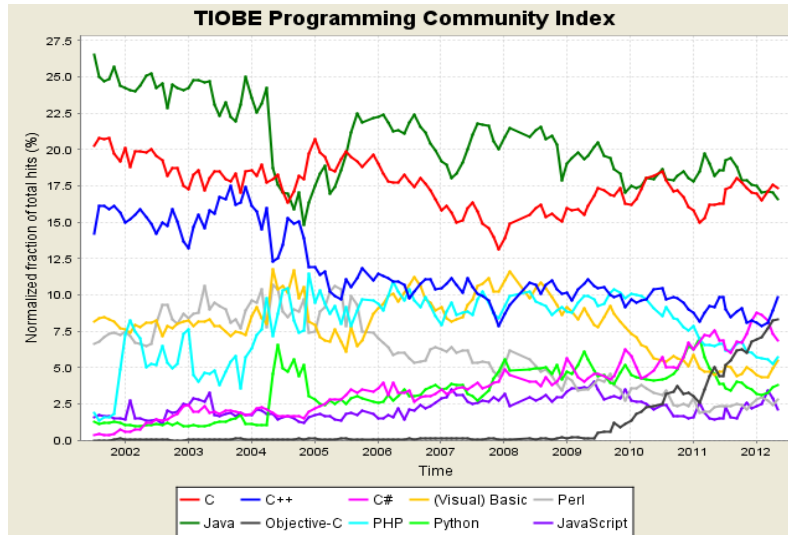


Figure 3.1: Les tendances à long terme pour les 10 premiers langages de programmation. [19]

Un des principaux intérêts du C est qu'il est un langage très portable. Un programme écrit en C en respectant la norme ANSI est portable sans modifications sur n'importe quel système d'exploitation disposant d'un compilateur C: tels que Windows, UNIX, VMS (système des VAX) ou encore OS/390 ou z/Os (l'OS des mainframes IBM).

Parmi les caractéristiques les plus importantes du langage C, la rapidité des programmes écrits en C. En plus le langage C est un langage "faiblement typé" : les types de données qu'il manipule sont très restreints, et proches de la représentation interne du processeur.

### 3.2.2 Sous-C proposé

Nous présentons le sous-ensemble C couvert par notre projet. La partie de la syntaxe du C prise en compte est la suivante :

- Les programmes C ne sont pas autorisés à avoir les opérations, telles que *scanf*, puisqu'il n'existe aucune procédure en Promela pour la lecture, et la bibliothèque des fonction des inputs du standards ANSI n'est pas supportée.
- Les structures de données supportées sont les integers, array, structure qui permettent de former des listes liées) ainsi que les tableaux multidimensionnels qu'ils ne sont pas supportés en Promela, où nous présentons une méthode contournons cette limitation.
- Les expressions peuvent être construites avec les opérations ordinaires, eg ++, %, >, and +=.

- Les types supportés sont limités aux type de données déjà soutenus au-dessus.
- L'affectation, les appels de fonctions, les sauts ( goto, break, return), les sélection et les itérations peuvent être utilisés comme des statements.
- Les définitions de fonction, les déclarations, les appels et les retours sont supportés.

La grammaire du Langage C est décrite dans l'Annexe A, dans ce qui suit nous présentons La syntaxe détaillée du sous-ensemble C considéré :

### 3.2.2.1 Mots Clés :

Un certain nombre de mots clés sont qualifiés et réservés au langage C (c'est-à-dire, ils ne peuvent pas être utilisés comme une déclaration de variables, de fonctions ou bien de structures (voir le tableau 3.1)

<b>int</b>	<b>double</b>	<b>signed</b>	<b>Unsigned</b>
<b>long</b>	<b>short</b>	<b>void</b>	<b>Struct</b>
<b>const</b>	<b>static</b>	<b>switch</b>	<b>Case</b>
<b>default</b>	<b>if</b>	<b>else</b>	<b>For</b>
<b>do</b>	<b>while</b>	<b>goto</b>	<b>Break</b>
<b>return</b>			

Tableau 3.2 : Mots-clés utilisés en sous-C proposé.

### 3.2.2.2 Types de données :

#### Types de données Primitives :

Notre travail est axé sur l'abstraction des modèles de vérification Promela à partir des programmes C, Nous devons donc prendre en compte que certaines variables supportées par le langage C ne sont pas aussi supportées par Promela. (Voir le Tableau 3.3)

Type de donnée	Signification	Soutenu.
char	Caractère	Non
short	Entier court	Oui
int	Entier	Oui
signed	Entier signé	Oui
unsigned	Entier non signé	Oui
long	Entier long	Oui
foat	réel	Non
double	double	Non

Tableau 3.3 : Type de données prises en charge en sous-C.

Les tableaux (Arrays) :

- Les tableaux **unidimensionnels** en C sont déclarés comme suit :

```
type Nom_du_tableau [Nombre d'éléments]
    int array[100];
```

- Les tableaux **multidimensionnels** (Arrays) :

- 

```
type Nom_du_tableaux [Nombre d'éléments] [Nombre d'éléments]
    int array[100];
```

Les structures :

La déclaration des structures en C:

<pre>struct Nom_Structure {     type_champ1 Nom_Champ1;     type_champ2 Nom_Champ2;     ... } ;</pre>	<pre>struct MaStructure {     int Age;     char Sexe;     char Nom[12];}</pre>
	<pre>struct MaStructure {     int Age;     char Age;     struct MaStructure     StructBis;};</pre>

Tableau 3.4: Les structures dans C.

### 3.2.2.3 Opérateurs et expressions :

Les expressions peuvent être variables et des constantes ou des compositions d'entre eux en utilisant les opérateurs figurant dans le Tableau 3.5.

Expressions arithmétiques	Opérateurs d'arithmétiques	+, -, *, /, %
	incrémentement décrémentement	++ --
Expressions booléennes	Opérateurs booléennes	==, !=, >, <, <=, >=   , &&, !
Expressions généralisées	Affectations	=

Tableau 3.5: Expressions et opérations prises en charge en sous-C.

### 3.2.2.4 Contrôles d'exécution

C dispose des structures de contrôle d'exécution classiques: alternatives, répétitions, sélections, etc. Les instructions sont illustrées dans le tableau 3.6

Alternative	If, else	if( <condition> ) <traitement1> else <traitement2>
Sélection	switch, case, default	switch( <expression> ) { case <valeur1> : ... traitement 1 case <valeur2> : ... traitement 2 default : ... traitement par défaut }
Répétitions	while	while( <condition> ) { ... }
	for	for( <expr1>; <expr2>; <expr3> ) { ... }
	do..while	do { ... } while( <condition> );
Branchement	goto	goto <label>;
	break	Break ;
	return	return <expression>;
Appel de fonction	[variable =] functionName( <parameter-list> );	

Tableau 3.6 : Structures de contrôle d'exécution dans C.

### 3.2.2.5 Les Fonctions

La définition de fonction se compose d'un type retourné. (avoir si aucune valeur n'est retournée), un nom unique, une liste de paramètres entre parenthèses avec leurs types:

```
<return-type> functionName( <parameter-list> )  
{  
    <statements>  
    return <expression of type return-type>;  
}
```

Les types de retour (return) pourraient être *int*, *struct*, *void*.

### 3.3 Promela

Le Langage input du SPIN est le Promela. Un modèle décrit par le langage Promela sera composé de trois types d'objets: les processus, les canaux et les variables [4]. Les canaux sont utilisés pour communiquer entre les différents processus. Les processus décrivent le comportement du système modélisé et les canaux et les variables décrivent l'environnement où les processus seront s'exécutent. Ces aspects importants de Promela seront décrite dans ce chapitre.

Une définition de la grammaire de la langue complète Promela peuvent être trouvées dans l'annexe B [11, 13].

#### 3.3.1 Mots clés

Les identificateurs suivants sont réservés pour une utilisation en tant que mots-clés. Ils ne représentent pas tous les mots-clés dans le spin, mais seulement les mots que nous allons adopter dans notre travail. (Voir le Tableau 3.7)

atomic		bit		break
chan	do	d_step	else	fi
goto	init	int	if	od
Printf	proctype	run	skip	typedef

Tableau 3.7: Mots-clés de Promela utilisés

#### 3.3.2 Types de données

##### Types de données primitifs :

Les types de base de Promela sont : `bit` ou `bool`, `byte`, `short` et `int` ; Les variables arithmétiques implicitement initialisée à 0, Pas de `float`, aucune variable `String` (uniquement dans les états d'impression).

```
typename varname [ initializer ]  
int a = 5;
```

##### Les tableaux (Arrays) :

Le Modèle Promela peut également avoir des tableaux, Promela ne supporte que les tableaux à une dimension. Comme en C, le premier objet d'un tableau est toujours index zéro. Le nombre d'objets dans le tableau doit être spécifié avec une constante numérique dans la déclaration.

```
typename varname '[' const ']  
int table [5];
```

### Les structures :

Pour déclarer un type de structure de données en utilise le mot `typedef`.

```
typedef name { decl_lst }  
  
typedef structure {int a, b};
```

### Les canaux (Channels):

Un canal dans Promela est un type de données avec deux opérations, d'**envoyer** et de **recevoir**. Les canaux sont utilisés pour transférer des messages entre les processus actifs. Chaque canal sera associé à un type de message, une fois un canal a été initialisé, Il peut seulement envoyer et recevoir des messages de son propre type de message. Tout au plus 255 canaux peuvent être créés [20].

```
chan varname = '[' const ']' of { typename }  
  
chan ch_bloc = [5] of {int}
```

Il y a deux types de canaux avec une sémantique différente:

- Les canaux de rendez-vous de capacité zéro, par exemple  
chan ret\_main = [0] {int},
- Les canaux tampon de capacité supérieure à zéro, par exemple  
chan comm = [ 9]{ int }.

Dans le cadre de ce projet, les canaux seront uniquement utilisés pour la synchronisation et la transmission de valeurs entre les processus que simulent les fonctions, de sorte que seuls les canaux de rendez-vous seront introduits.

- **Envoi:**        **ch\_bloc!expr**  
expr est envoyé sur ch\_bloc s'il reste de la place dans la file d'attente du canal.  
Instruction bloquante sinon.
- **Réception :**   **ch\_bloc?variable**  
variable reçoit la valeur du premier message du canal. Si aucun message sur le canal,  
instruction bloquante.

### 3.3.3 Opérateurs et expressions :

Les opérateurs dans le tableau 3.8 sont utilisés pour construire des expressions, mais les expressions pourraient aussi être juste une variable ou une valeur.

Expressions arithmétiques	Opérateurs d'arithmétiques	+, -, *, /, %
	incrémentement décrémentement	++ --
Expressions booléennes	Opérateurs booléennes	==, !=, >, <, <=, >=, &,   , &&, !
Expressions généralisées	Affectations	=

Tableau 3.8 : Expressions et opérations en Promela.

### 3.3.4 Contrôles d'exécution :

#### Séquence

La Séquence est utilisée pour envelopper un nombre arbitraire de bloc de code (instructions). Quatre types autorisés en séquence, qui sont les statements, les déclarations de variables et les opérations exclusives d'envoi et de réception sur les canaux.

Les étapes de la séquence sont séparés par un point-virgule (;) ou, de façon équivalente, une flèche (->). La flèche est parfois utilisée pour indiquer une relation de causalité entre des déclarations successives et aussi dans les états de sélection et répétition pour séparer les gardes de ses déclarations successives.

$(a==b) ; b++$  :  $b$  n'est incrémenté que si  $a$  est égal à  $b$ . Sinon, on dit que l'instruction  $(a==b)$  est bloquante. On préférera ici la notation  $(a==b) -> b++$ .

- **Séquence atomiques** : Toutes les instructions sont effectuées en un seul pas : aucune autre instruction ne peut être effectuée entre les instructions de « **atomic** ».

```
atomic { sequence }
```

#### Sélection :

Une sélection commence par un mot-clé **if**, et se termine avec un **fi**. La sélection est Activable si l'un des gardes est activable, si plusieurs gardes activables, choix non-déterministe, si aucun garde n'est activable et **else** présente, l'exécution des instructions suivant **else**.

```
if  
:: sequence  
[ :: sequence ]*  
Fi;
```

### Répétitions :

La répétition a une syntaxe similaire à celle de la sélection, sauf les mots-clés de départ et de fin. Pour sortir de la boucle, on peut utiliser une instruction **break** ou **goto**.

```
do
  :: sequence
[ :: sequence ]*
od
```

- **Break (break):** sautez à la fin de la boucle interne do.
- **Goto (goto )** saut inconditionnel à une instruction étiquetée (Lable).

```
goto labelname
```

- **Lable :** Toute construction déclaration ou de contrôle de flux peut être précédée par une étiquette. L'étiquette peut, mais ne doit pas, être utilisée en tant que destination d'un **goto**.

•

### 3.3.5 Processus:

Les Processus servent comme des «unités de fonction» d'un modèle Promela, et ne peuvent pas avoir des tableaux ou des structures avec des tableaux à l'intérieur comme paramètres formels. Au maximum, 255 processus pourraient être créés pour un modèle Promela [13]. Il existe trois types de processus en Promela, basic, active et initial, mais seuls les processus de basic et initial seront utilisés dans ce mémoire.

- **Processus de base :**

```
proctype nom_proc (paramètres formels) {
  /* Déclarations des variables locales et instructions */
}
```

Ces processus devraient être instanciés par l'instruction "**run**".

```
run nom_proc (paramètres effectifs)
```

- **Processus initial :**

```
init {
  /* Déclarations des variables locales et instructions */
}
```

### 3.4 Le principe de la translation C vers Promela

Le translateur prend en entrée un programme C et le convertit en un programme Promela afin qu'il puisse être vérifié avec le Spin model-checker.

Le programme C est vu par le translateur comme une suite de Tokens. Un token représente un des mots suivant :

- Mot-clé (réservé)
- Identifiant (variable, constante, nom\_fonction)
- Opérateur (arithmétique, booléenne et de comparaison)
- Affectation
- Littéral (chiffre et nombre)
- La ponctuation ( point-virgule, point, virgule, les accolades, les parenthèses, les commentaires)

La translation s'appuie sur la nature du token pour prévoir les constructions grammaticales possibles (chemin grammatical) en référençons sur la grammaire du C. la méthode garantie que toutes les possibilités sont prises en compte. La traduction est réalisée d'une façon directe et incrémentale, le flux de la traduction est dirigé par la sélection de l'un des chemins grammaticaux possibles. Au fur et à mesure que le programme C est lu, sa version Promela est éditée. Il faut préciser que la traduction sera réalisée une fois qu'on atteint un chemin grammatical unique, qui sera traduit en entier. Ce chemin représente en fait un bloc grammatical du C. le processus de la traduction procède de l'externe vers l'interne jusqu'à arrivé au bloc de bases. Et en séquence du début à la fin.

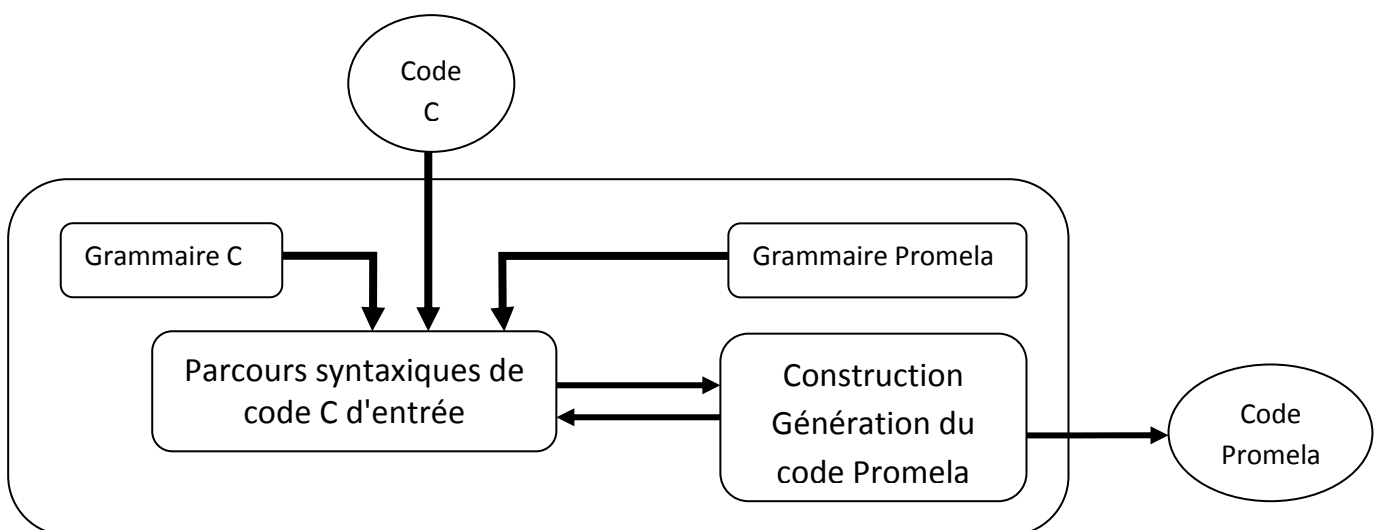
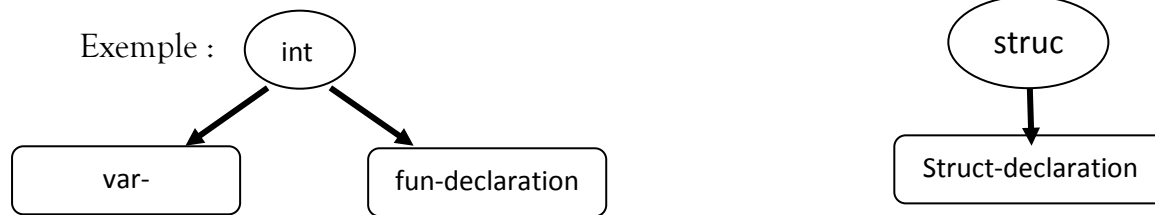


Figure 3.2 : Le principe de translation

### 3.4.1 Les chemins grammaticaux :

Un chemin grammatical décrit une possibilité de construction envisageable grammaticalement correcte au respect des règles de la grammaire C.



### 3.4.2 La sélection

La question pertinente est comment choisir le bon chemin parmi tous les chemins grammaticaux possibles. La sélection est faite en s'articulant sur la nature des tokens lus pour identifier les règles grammaticales correspondantes. Afin de cerner le nombre des tokens nécessaires à l'identification d'une façon unique d'un chemin grammatical, nous avons constaté que au maximum il faut lire trois token. Ce qui veut dire qu'on général un chemin unique est sélectionné si le nombre des token est  $\leq 3$ ; exemples :

- Cas un seul token suffit : struct
- Cas deux tokens nécessaires et suffisantes : a = ou a ==
- Cas trois tokens nécessaires et suffisantes : int a ( , int a;

Exemple : les chemins possibles dans La déclaration de fonction et Variable

declaration d'une fonction	declaration d'un Variable
int a ( ){ ...}	int a ;
<pre> graph TD     Decl[Declaration] --&gt; VarDecl[var-declaration]     Decl --&gt; FunDecl[fun-declaration]     VarDecl --&gt; TS1[type-specifier (int)]     TS1 --&gt; VDI1[var-decl-id (a)]     VDI1 --&gt; X1[X]     FunDecl --&gt; TS2[type-specifier (int)]     TS2 --&gt; P['(params) '(']   </pre>	<pre> graph TD     Decl[Declaration] --&gt; VarDecl[var-declaration]     Decl --&gt; FunDecl[fun-declaration]     VarDecl --&gt; TS1[type-specifier (int)]     TS1 --&gt; VDI1[var-decl-id (a)]     VDI1 --&gt; ID[ID (a)]     FunDecl --&gt; TS2[type-specifier (int)]     TS2 --&gt; VDI2[var-decl-id (a)]     VDI2 --&gt; X2[X]   </pre>

Tableau 3.9 les chemins possibles dans La déclaration de fonction et Variable

- Un exemple plus détaillé est présenté en Annexe C.

### 3.5 La translation de C vers Promela

Le translateur proposé assure la translation des types de données simples ( int et boolean) , ainsi que les types composés comme les tableaux et les structures. Ces structures de données sont supportés par les deux langages. Les tableaux multidimensionnels sont supportés par le C et non pas par le Promela. nous proposons une structure logique combinant les structure déjà existante en Promela ( Struct et table) pour l'implémenter.

Les structures de contrôle de sélections et de répétition et itérative sont converties en leur analogues du Promela. La notion d'atomicité présente en Promela et non pas en C, est prise en compte pour rendre atomique certaines séquence du C. nous proposons aussi de convertir le contrôle récursive. Et l'adapté pour l'appelle multiple de fonctions dans une même expression.

#### 3.5.1 Les type de données

##### 3.5.1.1 Types de données primitives

Seules les variables intégrales primitives seront prises en considération, Presque la déclaration de variables en langage C est similaire à Promela.

int a; short x; bool x;	var-declaration → type-specifier var-decl-list ;	<a href="#">one_decl</a> : [ <a href="#">visible</a> ] <a href="#">typename</a> <a href="#">ivar</a> [',' <a href="#">ivar</a> ] *	int a; int x; bool x;
C		Promela	

Tableau 3.10 Exemple de traduction des types de données primitifs.

##### 3.5.1.2 Les types de données composées

Les structures :

C et Promela permettent de manipuler des données structurées. L'accès aux champs se fait par la notation symbolique classique `nom.champ`

<b>struct</b> S1{ int value; };	<b>typedef</b> S1{ int value; }
C	Promela

Tableau 3.11 Exemple de traduction des structures.

## Chapitre 3 : Le principe de Translation du C vers Promela

### Les tableaux (Arrays) :

#### Tableau unidimensionnel:

int a[5];	1 var-declaration → type-specifier var-decl-list ; 2 var-decl-list → var-decl-list, var-decl-id   var-decl-id 3 var-decl-id → ID   ID [NUM ]	<u>one_decl</u> : [ <u>visible</u> ] <u>typename</u> <u>ivar</u> [ ',' <u>ivar</u> ] * ivar: <u>name</u> [ '[' <u>const</u> ']' ] ['=' <u>any_expr</u>   '=' <u>ch_init</u> ]	int a[5];
C		Promela	

Tableau 3.12 Exemple de traduction des Tableau unidimensionnel

#### Les tableaux multidimensionnels

Les tableaux multidimensionnels ne sont pas supportés par le Promela, pour contourner cette limitation on a proposé une structure en Promela qui peut supporter les tableaux multidimensionnels du C d'une façon adéquate. Cette structure de données adopte toutes les opérations permises sur les tableaux multidimensionnels du C.

La structure du tableau multidimensionnel est construire logiquement comme suit :

- Une des deux dimension ( la colonne) est construite avec *Struct*.
- Et l'autre dimension ( la ligne) est construite avec *table*.

La structure résultante est une **Table de Struct**

```
typedef VECTOR {int vector[10] };  
VECTOR matrix[5];  
  
matrix[3].vector[6] = 17;
```

Ici, la variable `matrix [5]` peut être considérée comme un tableau à deux dimensions. Lorsque on utilise une case spécifique du tableau, nous pouvons invoquer `matrix[x].vector[y]` où une donnée peut être stockée.

int t1[6][7];	typedef line_t1{int col[7]}; line_t1 t1[6];
C	Promela

Tableau 3.13 Exemple de traduction des Tableau multidimensionnel

### 3.5.2 Expressions et opérations :

Les opérateurs du C et Promela sont presque similaire, ce qui fait que la conversion est simple. La translation est décrite dans les tableaux suivants :

<pre>a = 4+c; a = c%b; a = a&amp;&amp; b; a++; a--;</pre>	<pre>a = 4 + c ; a = c % b ; a = a &amp;&amp; b ; a++ ; a-- ;</pre>
C	Promela

Tableau 3.14 Exemple de traduction de différentes expressions

### Expressions conditionnelles :

<pre>a = (b &gt; c) ? b : c;</pre>	<pre>if :: (b &gt; c) -&gt; a = b :: else -&gt; a = c fi;</pre>
C	Promela

Tableau 3.15 Exemple de traduction des Expressions conditionnelles

### 3.5.3 Contrôles d'exécution :

Les structures de contrôles des deux langages C et Promela sont différents, donc la translation est faite avec soin.

#### Sélection :

##### 1. If :

<pre>if(a&gt;b) b++; else a++;</pre>	<pre>if :: (a&gt;b) -&gt; b++ :: else -&gt; a++ fi</pre>
C	Promela

Tableau 3.16 Exemple de traduction de l'instruction « **if** ».

### 2. Switch:

<pre>switch(x) {     case 0:         a++;     case 1:         a- -;     default:         break; }</pre>	<pre>if :: (x ==0) -&gt;     a++;     b++ :: (x == 1) -&gt;     a- -;     b- -:: else -&gt;         break fi</pre>
C	Promela

Tableau 3.17 Exemple de traduction de l'instruction « **switch** ».

### Répétitions :

#### 1. For:

<pre>for(int i=0; i&lt; n; i++) {     a = a + i; };</pre>	<pre>int i=0; do :: !(i &lt; n) -&gt; break :: else -&gt;     a = a + i;     i++ od</pre>
C	Promela

Tableau 3.18 Exemple de traduction de la boucle « **for** ».

#### 2. While:

<pre>int i = 0; while (i&lt;5){     i = i + 1; }</pre>	<pre>int i = 0; do :: (i &lt; 5) -&gt;     i = i + 1; :: else -&gt; break; od;</pre>
C	Promela

Tableau 3.19 Exemple de traduction de la boucle « **while** ».

### 3.5.4 La notion d'atomicité :

A quelle granularité de l'exécution l'entrelacement apparaissent ? Une question qui suggère un raisonnement par réfutation, c'est-à-dire, à quelle granularité l'exécution l'entrelacement n'est pas permis. La notion d'atomicité exprime cette façon de voir.

L'atomicité est définie comme une expression ou un statement d'un processus qui s'exécute entièrement sans la possibilité d'entrelacement.

L'atomicité en Promela :

- L'affectation, jumps, skip, et les expressions sont atomiques.
- Les commandes gardées sont non atomiques.

Le Langage C n'est pas orienté concurrence de ce fait la notion d'atomicité n'est pas prévue. Puisque le Promela assure cette notion, on projette de s'en servir dans la traduction d'une séquence qui serait plus naturelle de s'exécutée d'une façon indivisible (pour prévoir le parallélisme du Promela).

Puisque la notion de l'atomicité n'est présente dans le langage C, on propose d'indiquer la séquence qui devrait être atomique par deux commentaires:

`/*atomic*/`                      Sequence                      `/*endatomic*/`

*Exemple*

<pre>int fun (int X,int Y) {     int Z =0 ;     " <b>atomic</b>     Z=Y ;     Y=X ;     X=Z ;     " <b>endatomic</b> }</pre>	<pre>proctype fun(chan in_fun;int X,int Y) {     int Z  = 0;     <b>atomic</b>     {         Z = Y;         Y = X;         X = Z;     } }</pre>
C	Promela

Tableau 3.20 Exemple de traduction de blocks Atomique « **atomic** »

### 3.5.5 Les Fonctions

Chaque définition de la fonction en C sera traduite à une déclaration de type de processus en Promela. Les variables locales seront conservées dans les définitions de processus correspondants, et seront valables dans le cadre de ses instantiations. Les paramètres dans les appels de fonction sont convertis en arguments dans la déclaration d'instanciation de processus.

<pre>void some(int a , int b) {     int some = a + b; }</pre>	<pre>proctype some (int a ; int b ) {     int some = a + b; }</pre>
<b>C</b>	<b>Promela</b>

Tableau 3.21 Exemple de traduction de fonction **void**

#### La récursivité :

Le mécanisme d'appel de fonctions en C et en Promela est différent, le passage des paramètres est distinct :

- En C : les paramètres sont passés par valeur (copie) ou adresse (originale).
- En Promela : le passage des paramètres inputs vers les fonctions peut se faire d'une façon ordinaire. ou bien par la définition de certains canaux, qui serviront à véhiculer les valeurs voulues envoyées vers et reçues des fonctions.

Cette différence de mécanismes d'appels implique une certaine délicatesse dans la traduction de la récursivité. Les valeurs retournées par les fonctions sont délivrées à travers des canaux, cette façon de faire impose des limites dans la manipulation des valeurs résultats des appels récursifs, qui réduit notre liberté dans la manipulation de telles données.

Dans le cas d'un seul appel dans une expression, le problème ne se pose pas. Cependant si l'appel concerne plus de deux fonctions, on n'est confronté à une situation qui n'est pas supportée par le Promela. Nous proposons d'imposer une contrainte sur la syntaxe du C comme suite : le résultat de chaque fonction appelée est sauvegardé dans une variable que nous déclarons au niveau du code C en question. Ainsi, Manipuler les valeurs des variables avec les canaux est plus souple qu'avec les fonctions.

<pre>int gcd(int x, int y) {     if(y==0) return x;     else { int temp;           temp =gcd(y, x % y);           return temp} }</pre>	<pre>proctype gcd( chan in_gcd int x; int y ) {     chan ret_gcd = [0] of {int};     if     :: (y == 0) -&gt;    in_gcd ! x;     :: else -&gt;      int temp ;     run gcd( ret_gcd, y , x % y );     ret_gcd ? temp     in_gcd ! temp;     fi ; }</pre>
<b>C</b>	<b>Promela</b>

Tableau 3.22 : Exemple de Traduction de programme récursive (GCD)

### 3.6 Conclusion

Maintenant que le principe de la technique utilisée est bien exhibé et bien expliqué notre souci se penche vers l'implémentation, où on a opté pour une programmation orienté objet afin de remplir les objectifs prédéfinis.