Université Mohammed V – Souissi
École Nationale Supérieure d'Informatique et d'Analyse des Systèmes

Examen

Année Universitaire : 2013 - 2014
Filière : Ingénieur
Semestre : S3
Période : P2

Module : **M3.4** - Compilation
Élément de Module : **M3.4.1** - Compilation
Professeur : Karim BAÏNA

- **Date : 08/01/2013**
- **Durée : 1H30**

Consignes aux élèves ingénieurs :
- *Le barème est donné seulement à titre indicatif !!*
- *Les **réponses directes** et **synthétiques** seront appréciées*
- *Soignez votre **présentation** et **écriture** !!*

Exercice I : Syntaxe et Analyse Syntaxique *(8pts)*

Question	Réponse Oui / Non	Explication Pourquoi ou Comment ?
Soit le langage L1 (de la famille Ada -ex. Pascal, Oberon, etc.-). dont un exemple de programme est : X : integer ; .. **begin** X := 4 ; .. **end** Les terminaux begin et end sont-ils simplement du sucre syntaxique (non nécessaire) pour L1 ?	**Non**	Soit la première règle de la grammaire de L1 : PROG → LISTE_DECL **begin** LISTE_INST **end** **First**(LISTE_DECL) ⊇ {IDF} **First**(LISTE_INST) ⊇ {IDF} Si l'on supprime begin et end, on aura **Follow**(LISTE_DECL) ⊇ **First**(LISTE_INST) ⊇ {IDF} et donc **First**(LISTE_DECL) ∩ **Follow**(LISTE_DECL) ≠ ∅ et la grammaire ne sera pas analysable en LL(1)
Soit le langage L2 (de la famille C -ex. C++, Java, C#, etc.-). dont un exemple de programme est : { int X ; .. X = 4 ; .. } La syntaxe de déclaration TYPE IDF (ex. int X) dans L2 est-elle choisie pour la simple beauté syntaxique ou elle aurait pu être changée en IDF TYPE (ex. X int) ?	**Non**	Soit la première règle de la grammaire de L2 : PROG → LISTE_DECL LISTE_INST **First**(LISTE_INST) ⊇ {IDF} Si l'on intervertit IDF TYPE (ex. X int), on aura **First**(LISTE_DECL) ⊇ {IDF} **Follow**(LISTE_DECL) ⊇ **First**(LISTE_INST) ⊇ {IDF} et donc **First**(LISTE_DECL) ∩ **Follow**(LISTE_DECL) ≠ ∅ et la grammaire ne sera pas analysable en LL(1)
Soit EXPA et EXPB respectivement les non terminaux des expressions arithmétiques et booléennes et soit la règle des affectations ASSIGN → IDF = EXP EXP→ EXPA | EXPB La règle d'affectation est-elle ambiguë ?	**Oui**	« IDF = IDF » est un contre-exemple de mots ayant deux arbres syntaxiques. On pourra obtenir IDF de deux manières : ASSIGN ⇒ IDF = EXPB ⇒ IDF = IDF et ASSIGN ⇒ IDF = EXPA ⇒ IDF = IDF
Est-ce que la règle d'affectation pourra être analysable par un parser LL(1) ? et Comment ?	**Oui**	Lors de l'analyse syntaxique LL(1), on désambiguïsera puis factorisera, en confondant sans distinction les expressions arithmétiques et booléennes dans la même classe et on reportera la résolution des erreurs syntaxiques lors de l'analyse sémantique à base de la vérification de types

Exercice II : Grammaire Attribuée et Analyse Sémantique *(2pts)*

Grammaire attribuée récursive gauche pour la reconnaissance des nombres binaires et leur conversion en décimal (méthode 1)		Grammaire attribuée récursive gauche pour la reconnaissance des nombres binaires et leur conversion en décimal (méthode 2 avec un nombre d'attributs minimaux)	
R1: Number → Sign List *List.pos = 0* *if (Sign.neg = true) Number.val = - List.val* *else Number.val = List.val*		**R1: Number → Sign List** .....**idem** *List.pos = 0* …... .....**idem** *if (Sign.neg = true) Number.val = - List.val* …….....**idem** *else Number.val = List.val*	
R2: Sign → +	*Sign.neg = false*	**R2: Sign → +**	..... **idem** *Sign.neg = false*
R3: Sign → -	*Sign.neg = true*	**R3: Sign → -**	.....**idem** *Sign.neg = true*
R4: List → Bit	*Bit.pos = List.pos* *List.val = Bit.val*	**R4: List → Bit**	.....**idem** *List.val = Bit.val*
R5: List0 → List1 Bit	*Bit.pos = List0.pos* *List1.pos = List0.pos + 1* *List0.val = List1.val + Bit.val*	**R5: List0 → List1 Bit**	.....*List0.val = (2*List1.val) + Bit.val*
R6: Bit → 0	*Bit.val = 0*	**R6: Bit → 0**	.....**idem** *Bit.val = 0*
R7: Bit → 1	*Bit.val = 2Bit.pos*	**R7: Bit → 1**	.....*Bit.val = 1*

Exercice III : Analyse Sémantique, Représentations Intermédiaires et Génération de pseudo-code *(10pts)*

Soit la grammaire hors-contexte LL(1) G des opérations arithmétiques +,-, *, / vue en cours

ADDSUBAUX → epsilon
| – MULTDIV ADDSUBAUX
| + MULTDIV ADDSUBAUX

MULTDIV → AUX MULTDIVAUX

MULTDIVAUX → epsilon
| * AUX MULTDIVAUX
| / AUX MULTDIVAUX

NULLABLE (MULTDIVAUX)= true

Follow (MULTDIVAUX) = { '+' , '-' , ';' , ')' }
First (MULTDIVAUX) = { '*', '/' }

Soit le type AST vu en cours et en TP et Soient les deux variantes des actions sémantiques de construction de l'arbre syntaxique abstrait (représentation intermédiaire) réalisant deux grammaires attribuées étendant la grammaire hors-contexte G

// Actions sémantiques de la grammaire attribuée V1

boolean **_multdiv**(AST *past){
```
boolean result;
AST *past1 = (AST *) malloc(sizeof(AST));
AST *past2 = (AST *) malloc(sizeof(AST));
(*past1) = (AST) malloc (sizeof(struct Exp));
if (_aux(past1)){
   token = _lire_token();
   if ((*past)->noeud.op.expression_gauche == NULL) (*past) = *past1;
   else (*past)->noeud.op.expression_droite = *past1;
   if (_multdivaux(past) == true){
      if ((arbre_droit(*past) != NULL) && (arbre_gauche(*past) != NULL)) {
         if (type(arbre_gauche(*past)) == type(arbre_droit(*past))){
            (*past)->typename = type(arbre_gauche(*past));
         }else (*past)->typename = Double;
      }else {(*past) = *past1;}
   result = true;
   }else result = false;
}else result = false;
return result;
} // end _multdiv
```

boolean **_multdivaux**(AST *past){
```
boolean result;
// traitement des follows
if ((token==PLUS)||(token==MIN)||(token==PVIRG)||(token==PCLOSE)){
   follow_token = true;
   result = true;
// traitement des firsts
}else if (token == MULT) {
   token = _lire_token();
   *past = creer_noeud_operation('*', *past, NULL, type(*past));
   if (_multdiv(past)){
      result = true;
   }else result = false;
} else if (token == DIV) {
   token = _lire_token();
   *past = creer_noeud_operation('/', *past, NULL, type(*past));
   if (_multdiv(past)) result = true;
   else result = false;
} else result = false;
return result;
} // end _multdivaux
```

// Actions sémantiques de la grammaire attribuée V2

boolean **_multdiv**(AST *past){
```
boolean result;
*past = NULL;
AST *past1 = (AST *) malloc(sizeof(AST));
AST *past2 = (AST *) malloc(sizeof(AST));
if (_aux(past1)){
   token = _lire_token();
   if (_multdivaux(past2) == true){
      if ( (*past1 != NULL) && (*past2 != NULL) ){
         char op = ((top(*past2)==plus)?'+':((top(*past2)==moins)?'-'
((top(*past2)==mult)?'*':'/')));
         if (type(*past1) == type(arbre_droit(*past2))){
            *past = creer_noeud_operation(op, *past1, arbre_droit(*past2), type(*past1));
         } else
            *past = creer_noeud_operation(op, *past1, arbre_droit(*past2), Double);
      }else *past = *past1;
      result = true;
   }else result = false;
}else result = false;
return result;
} // end _multdiv
```

boolean **_multdivaux**(AST *past){
```
boolean result;
*past = NULL;
AST *past1 = (AST *) malloc (sizeof(AST));
// traitement des follows
if ((token==PLUS)||(token==MIN)||(token==PVIRG)||(token==PCLOSE)){
   follow_token = true;
   result = true;
// traitement des firsts
}else if (token == MULT) {
   token = _lire_token();
   if (_multdiv(past1)){
      if ( (*past1 != NULL) ) *past = creer_noeud_operation('*', NULL, *past1, type(*past1));
      result = true;
   }else result = false;
} else if (token == DIV) {
   token = _lire_token();
   if (_multdiv(past1)){
      if (*past1 != NULL) *past = creer_noeud_operation('/', NULL, *past1, type(*past1));
      result = true;
   }else result = false;
} else result = false;
return result;
} // end _multdivaux
```

Grammaire Hors-Contexte G Récursive	(**A**) Gauche (**B**) Droite ……**B**……	
Priorités des opérateurs	(**A**) + >> - >> * >> / (**B**) + ≡ - >> * >> / (**C**) + ≡ - >> * ≡ / (**D**) * ≡ / >> + ≡ - ……**C**……	
Grammaire Attribuée	V1 est (**A**) S-Attribuée, (**B**) L-Attribuée……**B**……	V2 est (**A**) S-Attribuée, (**B**) L-Attribuée ……**A**……
Arbre Syntaxique Abstrait	AST V1 est (**A**) Gauche, (**B**) Droit ……**A**……	AST V2 est (**A**) Gauche, (**B**) Droit ……**B**……

REM Programme ZZ
… begin
hauteur_triangle = 15;
base_triangle = 10 ;
surface_triangle =
hauteur_triangle*base_triangle/2;
 	
print surface_triangle;
rayon_cercle = 100;
perimetre_cercle = 2 * PI * rayon_cercle;
 	
print perimetre_cercle; end
	REM pseudo-code généré depuis l'AST V1
	… begin:
	PUSH 15.000000 STORE hauteur_triangle
	PUSH 10.000000 STORE base_triangle
	……LOAD hauteur_triangle
	……LOAD base_triangle
	……MULT
	……PUSH 2.000000
	……SWAP
	……DIV
	……STORE surface_triangle
	LOAD surface_triangle PRINT
	PUSH 100.000000 STORE rayon_cercle
	……PUSH 2.000000
	……LOAD PI
	……MULT
	……LOAD rayon_cercle
	……MULT
	STORE perimetre_cercle
	LOAD perimetre_cercle PRINT end:
		REM pseudo-code généré depuis l'AST V2
		… begin:
		PUSH 15.000000 STORE hauteur_triangle
		PUSH 10.000000 STORE base_triangle
		……LOAD hauteur_triangle
		……LOAD base_triangle
		……PUSH 2.000000
		……SWAP
		……DIV
		……MULT
		……STORE surface_triangle
		LOAD surface_triangle PRINT
		PUSH 100.000000 STORE rayon_cercle
		……PUSH 2.000000
		……LOAD PI
		……LOAD rayon_cercle
		……MULT
		……MULT
		STORE perimetre_cercle
		LOAD perimetre_cercle PRINT end