2ème Année Cycle Supérieur (2CS)

Option : Systèmes Informatiques et Logiciels (SIL1)

Thème :

Analyse Syntaxique Manuelle LL(1)

ÉQUIPE 06

Réalisé par :

Djerfi Fatma
Mekircha Rafika Houda
Melliti Abdelmalek
Bounab Chaima

Demandé par :

M. ABDMEZIEM Riyadh

Année Universitaire 2025 / 2026

Table des matières

Liste des tableaux

Chapitre

1 Analyse Syntaxique

1.1 Introduction

L'analyse syntaxique constitue la deuxième étape majeure du processus de compilation. Son rôle est de vérifier si la suite de lexèmes (tokens) fournie par l'analyseur lexical respecte les règles grammaticales définies pour le langage **QueryLang**.

Dans ce chapitre, nous explorons deux approches complémentaires :

L'analyse manuelle : Mise en œuvre d'une table d'analyse pour un sous-ensemble du langage, afin de comprendre les mécanismes fondamentaux de la reconnaissance syntaxique.

L'intégration industrielle : Utilisation de l'outil **BISON** pour générer un analyseur LALR performant capable de traiter l'intégralité de la grammaire de QueryLang.

1.2 Analyse Syntaxique Manuelle

L'analyse syntaxique manuelle a été réalisée sur un sous-ensemble représentatif de QueryLang en utilisant la méthode LL(1).

1.2.1 Choix du Sous-ensemble et Méthode LL(1)

Nous avons sélectionné un sous-ensemble incluant les constructions essentielles du langage :

- Structure de programme (`begin program ... end program`)
- Déclarations de variables (`set`)

- Types de base (`integer`, `string`, `float`, `boolean`)
- Instructions d'affectation et d'affichage (`print`)
- Structures conditionnelles (`when-then-otherwise`)
- Expressions arithmétiques (`+`, `-`, `*`, `/`)
- Expressions de comparaison (`=`, `<>`, `<`, `>`, `<=`, `>=`)

La méthode LL(1) a été retenue pour sa simplicité d'implémentation et son adéquation à notre grammaire.

1.2.2 Grammaire Initiale

La grammaire initiale avec les tokens lexicaux est définie comme suit :

$$
\begin{aligned}
\text{PROGRAM} &\rightarrow \text{kw\_begin kw\_program identifier sep\_semicolon} \\
&\qquad \text{DECLS INSTRS kw\_end kw\_program sep\_semicolon} \\
\text{DECLS} &\rightarrow \text{DECLS DECL} \mid \varepsilon \\
\text{DECL} &\rightarrow \text{kw\_set identifier TYPE OPTINIT sep\_semicolon} \\
\text{OPTINIT} &\rightarrow \text{op\_eq\_tok EXPR} \mid \varepsilon \\
\text{TYPE} &\rightarrow \text{kw\_integer} \mid \text{kw\_string} \mid \text{kw\_float} \mid \text{kw\_boolean} \\
\text{INSTRS} &\rightarrow \text{INSTRS INSTR} \mid \varepsilon \\
\text{INSTR} &\rightarrow \text{ASSIGN} \mid \text{PRINT} \mid \text{WHEN} \mid \text{DECL} \\
\text{ASSIGN} &\rightarrow \text{identifier op\_eq\_tok EXPR sep\_semicolon} \\
\text{PRINT} &\rightarrow \text{kw\_print EXPR sep\_semicolon} \\
\text{WHEN} &\rightarrow \text{kw\_when COND kw\_then INSTRS OPTOTHERWISE} \\
&\qquad \text{kw\_end kw\_when sep\_semicolon} \\
\text{OPTOTHERWISE} &\rightarrow \text{kw\_otherwise INSTRS} \mid \varepsilon \\
\text{EXPR} &\rightarrow \text{EXPR op\_plus TERM} \mid \text{EXPR op\_minus TERM} \mid \text{TERM} \\
\text{TERM} &\rightarrow \text{TERM op\_mult FACTOR} \mid \text{TERM op\_div FACTOR} \mid \text{FACTOR} \\
\text{FACTOR} &\rightarrow \text{int\_literal} \mid \text{float\_literal} \mid \text{string\_literal} \\
&\qquad \mid \text{identifier} \mid \text{sep\_lparen EXPR sep\_rparen} \\
\text{COND} &\rightarrow \text{EXPR RELOP EXPR} \\
\text{RELOP} &\rightarrow \text{op\_eq\_tok} \mid \text{op\_neq\_tok} \mid \text{op\_lt\_tok} \\
&\qquad \mid \text{op\_gt\_tok} \mid \text{op\_lte\_tok} \mid \text{op\_gte\_tok}
\end{aligned}
$$

1.2.3 Correspondance Tokens Lexicaux Grammaire

N°	Token Lexical	Utilisation Grammaire	Exemple
t1	KW_BEGIN	Début de programme	BEGIN PROGRAM
t2	KW_PROGRAM	Déclaration programme	BEGIN PROGRAM TestProg
t3	KW_END	Fin de programme	END PROGRAM;
t4	KW_SET	Déclaration variable	SET x INTEGER = 10;
t5	KW_INTEGER	Type entier	SET x INTEGER;
t6	KW_STRING	Type chaîne	SET msg STRING;
t7	KW_FLOAT	Type réel	SET pi FLOAT = 3.14;
t8	KW_BOOLEAN	Type booléen	SET actif BOOLEAN = true;
t9	KW_WHEN	Condition	WHEN x > 5 THEN
t10	KW_THEN	Suite condition	WHEN x > 5 THEN
t11	KW_OTHERWISE	Alternative	OTHERWISE
t12	KW_PRINT	Affichage	PRINT x;
t13	IDENTIFIER	Noms de variables	x, message, counter
t14	INT_LITERAL	Constantes entières	10, 42, -5
t15	FLOAT_LITERAL	Constantes réelles	3.14, 0.5
t16	STRING_LITERAL	Constantes chaînes	’Hello’, ’World’
t17	OP_EQ_TOK	Égalité/Affectation	x = 10, WHEN x = 5
t18	OP_NEQ_TOK	Différent	WHEN x <> 0
t19	OP_LT_TOK	Inférieur	WHEN x < 10
t20	OP_GT_TOK	Supérieur	WHEN x > 5
t21	OP_LTE_TOK	Inférieur ou égal	WHEN x <= 100
t22	OP_GTE_TOK	Supérieur ou égal	WHEN x >= 0
t23	OP_PLUS	Addition	x + 5
t24	OP_MINUS	Soustraction	x - 2
t25	OP_MULT	Multiplication	x * 3
t26	OP_DIV	Division	x / 2

N°	Token Lexical	Utilisation Grammaire	Exemple
t27	SEP_SEMICOLON	Fin d'instruction	;
t28	SEP_LPAREN	Parenthèse ouvrante	(x + 5)
t29	SEP_RPAREN	Parenthèse fermante	(x + 5)
t30	END_OF_FILE	Fin de fichier	# (symbole spécial)

Table 1.1: Correspondance entre tokens lexicaux numérotés et leur utilisation dans la grammaire

1.2.4 Non-Terminaux de la Grammaire

N°	Non-Terminal	Description	Exemple de dérivation
NT1	PROGRAM	Programme complet	Tout le fichier .ql
NT2	DECLS	Liste de déclarations	Séquence de SET
NT3	DECL	Une déclaration	SET x INTEGER = 10;
NT4	TYPE	Type de variable	INTEGER, STRING, etc.
NT5	OPTINIT	Initialisation optionnelle	= 10 ou vide
NT6	INSTRS	Liste d'instructions	Séquence d'instructions
NT7	INSTR	Une instruction	Assign, Print, WHEN ou Decl
NT8	ASSIGN	Affectation	x = 5;
NT9	PRINT	Affichage	PRINT x;
NT10	WHEN	Condition	WHEN x > 5 THEN ... END WHEN;
NT11	OPTOTHERWISE	Partie ELSE optionnelle	OTHERWISE ... ou vide
NT12	EXPR	Expression arithmétique	x + 5 * 2
NT13	EXPRPRIME	Suite d'expression	+ 5, - 3, ou vide
NT14	TERM	Terme	x * 2
NT15	TERMPRIME	Suite de terme	* 2, / 3, ou vide
NT16	FACTOR	Facteur	x, 10, (x+5)
NT17	COND	Condition booléenne	x > 5, y = 10
NT18	RELOP	Opérateur de comparaison	=, <>, <, etc.

Table 1.2: Non-terminaux numérotés de la grammaire QueryLang

1.2.5 Transformations pour LL(1)

La grammaire initiale contient plusieurs problèmes de récursion gauche qui empêchent l'analyse LL(1). Nous avons appliqué les transformations suivantes :

1.2.5.1 Élimination de la récursion gauche

1. Transformation de DECLS :

La règle DECLS $\rightarrow$ DECLS DECL | ε est transformée en inversant l'ordre, convertissant la récursion gauche en récursion droite,parce que la règle originale est équivalente à une répétition (Kleene star) DECLS $\rightarrow$ DECL* :

DECLS $\rightarrow$ DECL DECLS | ε

2. Transformation de INSTRS :

De manière similaire, INSTRS $\rightarrow$ INSTRS INSTR | ε devient : INSTRS $\rightarrow$ INSTR INSTRS | ε

3. Transformation de EXPR :

Pour EXPR $\rightarrow$ EXPR op_plus TERM | EXPR op_minus TERM | TERM, nous introduisons un nouveau non-terminal EXPRPRIME :

$$\text{EXPR} \rightarrow \text{TERM EXPRPRIME}$$

$$\text{EXPRPRIME} \rightarrow \text{op\_plus TERM EXPRPRIME} \mid \text{op\_minus TERM EXPRPRIME} \mid \varepsilon$$

Cette transformation préserve la priorité des opérateurs et permet l'analyse LL(1).

4. Transformation de TERM :

De la même façon, TERM $\rightarrow$ TERM op_mult FACTOR | TERM op_div FACTOR | FACTOR devient :

$$\text{TERM} \rightarrow \text{FACTOR TERMPRIME}$$

$$\text{TERMPRIME} \rightarrow \text{op\_mult FACTOR TERMPRIME} \mid \text{op\_div FACTOR TERMPRIME} \mid \varepsilon$$

1.2.6 Grammaire après transformation LL(1)

Après élimination de la récursion gauche et factorisation, nous obtenons la grammaire LL(1) suivante. Chaque production est numérotée pour faciliter la construction de la table d'analyse :

N°	**Production**
(1)	PROGRAM → kw_begin kw_program identifier sep_semicolon DECLS INSTRS kw_end kw_program sep_semicolon
(2)	DECLS → DECL DECLS
(3)	DECLS → ε
(4)	DECL → kw_set identifier TYPE OPTINIT sep_semicolon
(5)	TYPE → kw_integer
(6)	TYPE → kw_string
(7)	TYPE → kw_float
(8)	TYPE → kw_boolean
(9)	OPTINIT → op_eq_tok EXPR
(10)	OPTINIT → ε
(11)	INSTRS → INSTR INSTRS
(12)	INSTRS → ε
(13)	INSTR → ASSIGN
(14)	INSTR → PRINT
(15)	INSTR → WHEN
(16)	INSTR → DECL
(17)	ASSIGN → identifier op_eq_tok EXPR sep_semicolon
(18)	PRINT → kw_print EXPR sep_semicolon
(19)	WHEN → kw_when COND kw_then INSTRS OPTOTHERWISE kw_end kw_when sep_semicolon
(20)	OPTOTHERWISE → kw_otherwise INSTRS
(21)	OPTOTHERWISE → ε

N°	Production
(22)	EXPR → TERM EXPRPRIME
(23)	EXPRPRIME → op_plus TERM EXPRPRIME
(24)	EXPRPRIME → op_minus TERM EXPRPRIME
(25)	EXPRPRIME → ε
(26)	TERM → FACTOR TERMPRIME
(27)	TERMPRIME → op_mult FACTOR TERMPRIME
(28)	TERMPRIME → op_div FACTOR TERMPRIME
(29)	TERMPRIME → ε
(30)	FACTOR → int_literal
(31)	FACTOR → float_literal
(32)	FACTOR → string_literal
(33)	FACTOR → identifier
(34)	FACTOR → sep_lparen EXPR sep_rparen
(35)	COND → EXPR RELOP EXPR
(36)	RELOP → op_eq_tok
(37)	RELOP → op_neq_tok
(38)	RELOP → op_lt_tok
(39)	RELOP → op_gt_tok
(40)	RELOP → op_lte_tok
(41)	RELOP → op_gte_tok

Table 1.3: Grammaire LL(1) de QueryLang avec numérotation des productions

1.2.7 Calcul des Ensembles FIRST et FOLLOW

1.2.7.1 Tableau des débuts et suivants

Le tableau suivant regroupe les ensembles de terminaux nécessaires à la construction de la table d'analyse LL(1) :

1.2.7.2 La table des ensembles FIRST et FOLLOW

Non-Terminal	**Débuts**	**Suivants**
PROGRAM	{KW_BEGIN}	{#}
DECLS	{KW_SET, ε}	{IDENTIFIER, KW_PRINT, KW_WHEN, KW_SET, KW_END}
DECL	{KW_SET}	{KW_SET, IDENTIFIER, KW_PRINT, KW_WHEN, KW_END}
TYPE	{KW_INTEGER, KW_STRING, KW_FLOAT, KW_BOOLEAN}	{OP_EQ_TOK, SEP_SEMICOLON}
OPTINIT	{OP_EQ_TOK, ε}	{SEP_SEMICOLON}
INSTRS	{IDENTIFIER, KW_PRINT, KW_WHEN, KW_SET, ε}	{KW_END, KW_OTHERWISE}
INSTR	{IDENTIFIER, KW_PRINT, KW_WHEN, KW_SET}	{IDENTIFIER, KW_PRINT, KW_WHEN, KW_SET, KW_END, KW_OTHERWISE}
ASSIGN	{IDENTIFIER}	{IDENTIFIER, KW_PRINT, KW_WHEN, KW_SET, KW_END, KW_OTHERWISE}
PRINT	{KW_PRINT}	{IDENTIFIER, KW_PRINT, KW_WHEN, KW_SET, KW_END, KW_OTHERWISE}
WHEN	{KW_WHEN}	{IDENTIFIER, KW_PRINT, KW_WHEN, KW_SET, KW_END, KW_OTHERWISE}
OPTOTHERWISE	{KW_OTHERWISE, ε}	{KW_END}

Non-Terminal	**Débuts**	**Suivants**
EXPR	{INT_LITERAL, FLOAT_LITERAL, STRING_LITERAL, IDENTIFIER, SEP_LPAREN}	{SEP_SEMICOLON, SEP_RPAREN, OP_EQ_TOK, OP_NEQ_TOK, OP_LT_TOK, OP_GT_TOK, OP_LTE_TOK, OP_GTE_TOK, KW_THEN}
EXPRPRIME	{OP_PLUS, OP_MINUS, ε}	{SEP_SEMICOLON, SEP_RPAREN, OP_EQ_TOK, OP_NEQ_TOK, OP_LT_TOK, OP_GT_TOK, OP_LTE_TOK, OP_GTE_TOK, KW_THEN}
TERM	{INT_LITERAL, FLOAT_LITERAL, STRING_LITERAL, IDENTIFIER, SEP_LPAREN}	{OP_PLUS, OP_MINUS, SEP_SEMICOLON, SEP_RPAREN, OP_EQ_TOK, OP_NEQ_TOK, OP_LT_TOK, OP_GT_TOK, OP_LTE_TOK, OP_GTE_TOK, KW_THEN}
TERMPRIME	{OP_MULT, OP_DIV, ε}	{OP_PLUS, OP_MINUS, SEP_SEMICOLON, SEP_RPAREN, OP_EQ_TOK, OP_NEQ_TOK, OP_LT_TOK, OP_GT_TOK, OP_LTE_TOK, OP_GTE_TOK, KW_THEN}

Non-Terminal	**Débuts**	**Suivants**
FACTOR	{INT_LITERAL, FLOAT_LITERAL, STRING_LITERAL, IDENTIFIER, SEP_LPAREN}	{OP_MULT, OP_DIV, OP_PLUS, OP_MINUS, SEP_SEMICOLON, SEP_RPAREN, OP_EQ_TOK, OP_NEQ_TOK, OP_LT_TOK, OP_GT_TOK, OP_LTE_TOK, OP_GTE_TOK, KW_THEN}
COND	{INT_LITERAL, FLOAT_LITERAL, STRING_LITERAL, IDENTIFIER, SEP_LPAREN}	{KW_THEN}
RELOP	{OP_EQ_TOK, OP_NEQ_TOK, OP_LT_TOK, OP_GT_TOK, OP_LTE_TOK, OP_GTE_TOK}	{INT_LITERAL, FLOAT_LITERAL, STRING_LITERAL, IDENTIFIER, SEP_LPAREN}

1.2.8 Construction de la Table d'Analyse LL(1)

La table d'analyse LL(1) est construite selon le principe : pour chaque production `A → `, on ajoute cette production à `M[A, t]` pour tout terminal `t` dans `FIRST()`. Si `FIRST()`, on ajoute également la production pour tout terminal dans `FOLLOW(A)`.

1.2.8.1 Propriétés de la Table

La table résultante contient :

- **18 non-terminaux** (NT1 à NT18) : PROGRAM, DECLS, DECL, TYPE, OPTINIT, INSTRS, INSTR, ASSIGN, PRINT, WHEN, OPTOTHERWISE, EXPR, EXPRPRIME, TERM, TERMPRIME, FACTOR, COND, RELOP
- **30 terminaux** (t1 à t30) : tous les tokens définis dans l'analyse lexicale
- **41 productions** numérotées de (1) à (41)
- **Aucun conflit** : chaque case contient au maximum une production

1.2.8.2 Table d'Analyse LL(1) Complète

La table suivante présente l'ensemble des décisions de l'analyseur syntaxique. Chaque entrée M[NTi, tj] indique quelle production appliquer lorsque le non-terminal NTi est en sommet de pile et le terminal tj est lu en entrée. Les cases vides indiquent une erreur syntaxique.

	NT1	NT2	NT3	NT4	NT5	NT6	NT7	NT8	NT9	NT10	NT11	NT12	NT13	NT14	NT15	NT16	NT17	NT18
t1	1																	
t2																		
t3		3				12					21		25		29			
t4		2	4			11	16											
t5				5														
t6				6														
t7				7														
t8				8														
t9						11	15			19								
t10													25		29			
t11						12					20							
t12		3				11	14		18									
t13		3				11	13	17				22		26		33	35	
t14												22		26		30	35	
t15												22		26		31	35	
t16												22		26		32	35	
t17					9								25		29			36
t18													25		29			37
t19													25		29			38
t20													25		29			39
t21													25		29			40
t22													25		29			41
t23													23		29			
t24													24		29			
t25															27			
t26															28			
t27					10								25		29			
t28												22		26		34	35	
t29													25		29			
t30																		

Table 1.5: Table d'analyse LL(1) complète pour QueryLang

1.2.8.3 Légende de la Table

Non-terminaux (colonnes) : 3

- NT1 = PROGRAM
- NT2 = DECLS
- NT3 = DECL
- NT4 = TYPE
- NT5 = OPTINIT

- NT6 = INSTRS
- NT7 = INSTR
- NT8 = ASSIGN
- NT9 = PRINT
- NT10 = WHEN
- NT11 = OPTOTHERWISE
- NT12 = EXPR
- NT13 = EXPRPRIME
- NT14 = TERM
- NT15 = TERMPRIME
- NT16 = FACTOR
- NT17 = COND
- NT18 = RELOP

Terminaux (lignes) : 3

- t1 = KW_BEGIN
- t2 = KW_PROGRAM
- t3 = KW_END
- t4 = KW_SET
- t5 = KW_INTEGER
- t6 = KW_STRING
- t7 = KW_FLOAT

- t8 = KW_BOOLEAN
- t9 = KW_WHEN
- t10 = KW_THEN
- t11 = KW_OTHERWISE
- t12 = KW_PRINT
- t13 = IDENTIFIER
- t14 = INT_LITERAL
- t15 = FLOAT_LITERAL
- t16 = STRING_LITERAL
- t17 = OP_EQ_TOK
- t18 = OP_NEQ_TOK
- t19 = OP_LT_TOK
- t20 = OP_GT_TOK
- t21 = OP_LTE_TOK
- t22 = OP_GTE_TOK
- t23 = OP_PLUS
- t24 = OP_MINUS
- t25 = OP_MULT
- t26 = OP_DIV
- t27 = SEP_SEMICOLON
- t28 = SEP_LPAREN

- t29 = SEP_RPAREN
- t30 = #

Exemples d'utilisation de la table :

- M[NT1, t1] = 1 signifie : pour PROGRAM avec BEGIN en entrée, appliquer la production (1)
- M[NT7, t13] = 13 signifie : pour INSTR avec IDENTIFIER en entrée, appliquer la production (13) qui donne ASSIGN
- M[NT13, t23] = 23 signifie : pour EXPRPRIME avec OP_PLUS en entrée, appliquer la production (23)
- Une case vide indique une erreur syntaxique

1.2.8.4 Validation de la Table

La table d'analyse LL(1) ainsi construite possède les propriétés suivantes :

1. **Table mono-définie** : chaque entrée de la table contient au plus une règle de production de la grammaire, ce qui confirme que la grammaire est LL(1).
2. **Couverture complète** : pour chaque production A $\rightarrow$, la règle est ajoutée à M[A,a] pour tout terminal a dans DEBUT(), et à M[A,b] pour tout b SUIVANT(A) si DEBUT().
3. **Détection d'erreurs** : toutes les entrées non définies sont marquées "erreur", permettant d'identifier immédiatement les erreurs syntaxiques.
4. **Analyse déterministe** : au cours de toute dérivation, il existera au plus une possibilité pour remplacer un non terminal par une de ses parties droites selon le caractère courant de l'entrée à analyser.

1.2.9 Validation de la Grammaire LL(1)

La grammaire transformée vérifie les conditions LL(1) :

1. **Absence de récursivité à gauche** : toutes les récursions gauches ont été éliminées par transformation en récursions droites.
2. **Factorisation gauche effectuée** : la grammaire ne nécessitait pas de factorisation à gauche car aucune production ne partageait un préfixe commun.
3. **Table d'analyse mono-définie** : chaque entrée de la table contient au plus une règle de production de la grammaire.
4. **Grammaire non ambiguë** : si une grammaire G est LL(1) alors G est non ambiguë.
5. **Analyse descendante sans retour arrière** : au cours de toute dérivation, il existera au plus une possibilité pour remplacer un non terminal par une de ses parties droites selon le caractère courant de l'entrée à analyser.

Cette table d'analyse permet maintenant d'implémenter un analyseur syntaxique descendant prédictif qui reconnaît les programmes QueryLang valides du sous-ensemble défini.

1.2.9.1 Exemple d'Analyse

Considérons l'analyse du fragment de programme QueryLang suivant :

```
BEGIN PROGRAM TestProg ;
    SET x INTEGER = 5 ;
END PROGRAM ;
```

Contenu Pile **(sommet pile à droite)**	**Restant de chaîne** **à analyser**	**Action**
# PROGRAM	kw_begin kw_program identifier ... #	Remplacer PROGRAM par production (1)
# sep_semicolon kw_program kw_end INSTRS DECLS sep_semicolon identifier kw_program kw_begin	kw_begin ... #	avancer
# sep_semicolon kw_program kw_end INSTRS DECLS sep_semicolon identifier kw_program	kw_program identifier ... #	avancer
# sep_semicolon kw_program kw_end INSTRS DECLS sep_semicolon identifier	identifier sep_semicolon kw_set ... #	avancer
# sep_semicolon kw_program kw_end INSTRS DECLS sep_semicolon	sep_semicolon kw_set ... #	avancer
# sep_semicolon kw_program kw_end INSTRS DECLS	kw_set identifier kw_integer ... #	Remplacer DECLS par DECL DECLS
# sep_semicolon kw_program kw_end INSTRS DECLS DECL	kw_set identifier kw_integer ... #	Remplacer DECL par production (4)
# sep_semicolon kw_program kw_end INSTRS DECLS sep_semicolon OPTINIT TYPE identifier kw_set	kw_set identifier kw_integer ... #	avancer
# sep_semicolon kw_program kw_end INSTRS DECLS sep_semicolon OPTINIT TYPE identifier	identifier kw_integer op_eq_tok ... #	avancer
# sep_semicolon kw_program kw_end INSTRS DECLS sep_semicolon OPTINIT TYPE	kw_integer op_eq_tok int_literal ... #	Remplacer TYPE par kw_integer
# sep_semicolon kw_program kw_end INSTRS DECLS sep_semicolon OPTINIT kw_integer	kw_integer op_eq_tok int_literal ... #	avancer
# sep_semicolon kw_program kw_end INSTRS DECLS sep_semicolon OPTINIT	op_eq_tok int_literal sep_semicolon ... #	Remplacer OPTINIT par op_eq_tok EXPR

Table 1.6: Trace d'analyse LL(1) - Partie 1/2

Contenu Pile (sommet pile à droite)	**Restant de chaîne à analyser**	**Action**
# sep_semicolon kw_program kw_end INSTRS DECLS sep_semicolon EXPR op_eq_tok	op_eq_tok int_literal sep_semicolon ... #	avancer
# sep_semicolon kw_program kw_end INSTRS DECLS sep_semicolon EXPR	int_literal sep_semicolon kw_end ... #	Remplacer EXPR par TERM EXPRPRIME
# sep_semicolon kw_program kw_end INSTRS DECLS sep_semicolon EXPRPRIME TERM	int_literal sep_semicolon kw_end ... #	Remplacer TERM par FACTOR TERMPRIME
# sep_semicolon kw_program kw_end INSTRS DECLS sep_semicolon EXPRPRIME TERMPRIME FACTOR	int_literal sep_semicolon kw_end ... #	Remplacer FACTOR par int_literal
# sep_semicolon kw_program kw_end INSTRS DECLS sep_semicolon EXPRPRIME TERMPRIME int_literal	int_literal sep_semicolon kw_end ... #	avancer
# sep_semicolon kw_program kw_end INSTRS DECLS sep_semicolon EXPRPRIME TERMPRIME	sep_semicolon kw_end kw_program #	Dépiler TERMPRIME (production)
# sep_semicolon kw_program kw_end INSTRS DECLS sep_semicolon EXPRPRIME	sep_semicolon kw_end kw_program #	Dépiler EXPRPRIME (production)
# sep_semicolon kw_program kw_end INSTRS DECLS sep_semicolon	sep_semicolon kw_end kw_program #	avancer
# sep_semicolon kw_program kw_end INSTRS DECLS	kw_end kw_program sep_semicolon #	Dépiler DECLS (production)
# sep_semicolon kw_program kw_end INSTRS	kw_end kw_program sep_semicolon #	Dépiler INSTRS (production)
# sep_semicolon kw_program kw_end	kw_end kw_program sep_semicolon #	avancer
# sep_semicolon kw_program	kw_program sep_semicolon #	avancer
# sep_semicolon	sep_semicolon #	avancer
#	#	Chaîne acceptée

Table 1.7: Trace d'analyse LL(1) - Partie 2/2

1.2.9.2 Remarques sur l'Implémentation

- L'analyseur utilise une pile explicite pour gérer les symboles de la grammaire.
- À chaque étape, la décision est prise uniquement en consultant la table M avec le sommet de pile et le symbole d'entrée courant.

- Lorsqu'une production est appliquée, le non-terminal en sommet de pile est remplacé par les symboles du membre droit de la production, empilés de droite à gauche.
- Les symboles terminaux sont consommés du flot d'entrée lorsqu'ils correspondent au sommet de pile.
- Les cases vides de la table déclenchent une erreur syntaxique, permettant une détection précoce des erreurs.

Cette implémentation garantit une analyse déterministe en temps linéaire pour tout programme du sous-ensemble défini de QueryLang.