Parte 2
Grammatica per LispKit

La grammatica

1 **Prog** ::= let Bind in Exp end | letrec Bind in Exp end
2 **Bind** ::= var = Exp X
3 **X** ::= and Bind | epsilon

4 **Exp** ::= Prog | lambda(Seq_Var) Exp | ExpA | OPP(Seq_Exp) | if Exp then Exp else Exp
5 **ExpA** ::= T E1
6 **E1** ::= OPA T E1 | epsilon
7 **T** ::= F T1
8 **T1** ::= OPM F T1 | epsilon
9 **F** ::= var Y | exp_const | (ExpA)
10 **Y** ::= (Seq_Exp) | epsilon
11 **OPA** ::= + |
12 **OPM** ::= * | /
13 **OPP** ::= cons | car | cdr | eq | leq | atom
14 **Seq_Exp** ::= Exp Seq_Exp | epsilon
15 **Seq_Var** ::= var Seq_Var | epsilon

Calcolo del First

Passo 0

Per ogni non terminale imposto il First come l'insieme vuoto

Passo 1

Si parte dal First calcolato precedentemente e si aggiungo i dati nuovi.

Non terminale	First Precendete	First Corrente
Prog		let, letrec
Bind		var
X		and, ϵ
Exp		First(Prog), lambda, First(ExpA), First(OPP), if
ExpA		First(T)
E1		First(OPA), ϵ
T		First(F)
T1		First(OPM), ϵ
F		var, exp_const. (
Y		(, ϵ
OPA		+, -
OPM		*, /

OPP		cons, car, cdr, eq, leq, atom
Seq_Exp		First(Exp), ϵ
Seq_Var		var, ϵ

Alcuni First dipendono da dei First precedenti che non sono ancora stati calcolati, biosgna quindi eseguire un altro passo.

Passo 2

Non terminale	**First Precedente**	**First Corrente**
Prog	let, letrec	-
Bind	var	-
X	and, ϵ	-
Exp	First(Prog), lambda, First(ExpA), First(OPP), if	lamba, if, let, letrec, First(T), cons. car, cdr, eq, leq, atom
ExpA	**First(T)**	**First(F)**
E1	First(OPA), ϵ	+, -, ϵ
T	First(F)	var, exp_const. (
T1	**First(OPM)**, ϵ	***, /, ϵ**
F	var, exp_const. (	-
Y	(, ϵ	-
OPA	+, -	-
OPM	*, /	-
OPP	cons, car, cdr, eq, leq, atom	-
Seq_Exp	**First(Exp)**, ϵ	**lamda, if, First(Prog), First(ExpA), First(OPP),** ϵ
Seq_Var	var, ϵ	-

Anche in questo caso non è stato raggiunto un punto fisso, quindi si deve eseguire un'altro passo.

Passo 3

Non terminale	**First Precedente**	**First Corrente**

Prog	let, letrec	-
Bind	var	-
X	and, ϵ	-
Exp	lamba, if, let, letrec, **First(ExpA)**, cons. car, cdr, eq, leq, atom	amba, if, let, letrec, **First(T)**, cons. car, cdr, eq, leq, atom
ExpA	**First(T)**	**var, exp_const. (**
E1	+, -, ϵ	-
T	var, exp_const. (	-
T1	*, /, ϵ	-
F	var, exp_const. (	-
Y	(, ϵ	-
OPA	+, -	-
OPM	*, /	-
OPP	cons, car, cdr, eq, leq, atom	-
Seq_Exp	lamda, if, **First(Prog)**, First(ExpA), First(OPP), ϵ	lamda, if, **let, letrec**, First(T), cons, car, cdr, eq, leq, atom, ϵ
Seq_Var	var, ϵ	-

Anche in questo caso non è stato raggiunto il punto fisso, serve un altro passo.

Passo 4

Non terminale	**First Precedente**	**First Corrente**
Prog	let, letrec	-
Bind	var	-
X	and, ϵ	-
Exp	lamba, if, let, letrec, **First(T)**, cons. car, cdr, eq, leq, atom	lamba, if, let, letrec, **var, exp_const, (**, cons. car, cdr, eq, leq, atom
ExpA	var, exp_const. (	-
E1	+, -, ϵ	-
T	var, exp_const. (	-

T1	*, /, ϵ	-
F	var, exp_const. (	-
Y	(, ϵ	-
OPA	+, -	-
OPM	*, /	-
OPP	cons, car, cdr, eq, leq, atom	-
Seq_Exp	lamda, if, let, letrec, **First(T)**, cons, car, cdr, eq, leq, atom, ϵ	lamda, if, let, letrec, **var, exp_const. (**, cons, car, cdr, eq, leq, atom, ϵ
Seq_Var	var, ϵ	-

Passo5

Non terminale	**First Precedente**	**First Corrente**
Prog	let, letrec	-
Bind	var	-
X	and, ϵ	-
Exp	lamba, if, let, letrec, var, exp_const, (, cons. car, cdr, eq, leq, atom	-
ExpA	var, exp_const. (	-
E1	+, -, ϵ	-
T	var, exp_const, (	-
T1	*, /, ϵ	-
F	var, exp_const, (	-
Y	(, ϵ	-
OPA	+, -	-
OPM	*, /	-
OPP	cons, car, cdr, eq, leq, atom	-
Seq_Exp	lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, ϵ	-

Seq_Var	var, ϵ	-

Calcolo dei Follow

Passo 0

Follow_0(Start) = $
Follow_0(X) = vuoto, per ogni X non terminale

Passo 1

Follow_i = Follow_i-1

per ogni non terminale X e per ogni produzione della grammatica Y ::= w1Xw2
si aggiunge a Follow_i(X) First(w2) (tranne ϵ)
se ϵ appartiene al First(w2) allora si aggiunge a Follow_i(X), Follow_i-1(Y)

L'algoritmo termina quando si raggiunge un punto fisso

Non terminale	**Follow Precendete**	**FollowCorrente**
Prog	$	$, Follow(Exp)
Bind		in, Follow(X),
X		Follow(Bind)
Exp		end, and, Follow(Exp), Follow(X), then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, Follow(Seq_Exp)
ExpA		Follow(Exp),),
E1		Follow(ExpA), Follow(E1)
T		+, -, Follow(E1)
T1		Follow(T), Follow(T1)
F		*, /, Follow(T1)
Y		Follow(F)
OPA		var, exp_const, (
OPM		var, exp_const, (
OPP		(
Seq_Exp		), Follow(Seq_Exp)
Seq_Var		), Follow(Seq_var)

Non è ancora stato raggiunto un punto fisso, è necessaria un'altra iterazione

Passo 2

Non terminale	**Follow Precendete**	**FollowCorrente**
Prog	$, **Follow(Exp)**	$, **end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, Follow(X), Follow(Seq_Exp)**
Bind	in, **Follow(X),**	in, **Follow(Bind)**
X	**Follow(Bind)**	**in, Follow(X)**
Exp	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, **Follow(X),** Follow(Seq_Exp)	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, **Follow(Bind),**)
ExpA	**Follow(Exp)**,),	**end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, Follow(X),**)
E1	**Follow(ExpA)**	**Follow(Exp),),**
T	+, -, **Follow(E1)**	+, -, **Follow(ExpA)**
T1	**Follow(T)**	**+, -, Follow(E1)**
F	*, /, **Follow(T1)**	*, /, **Follow(T)**
Y	**Follow(F)**	***, /, Follow(T1)**
OPA	var, exp_const, (	-
OPM	var, exp_const, (	-
OPP	(	-
Seq_Exp	)	-
Seq_Var	)	-

Anche in questo caso ci sono state delle modifiche, quindi è necessario un ulteriore passo dell'algoritmo.

Passo 3

Non terminale	**Follow Precendete**	**FollowCorrente**

Prog	$, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, **Follow(X)**, Follow(Seq_Exp)	$, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, **in,**)
Bind	in	-
X	in	-
Exp	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, **Follow(Bind)**,)	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, **in,**)
ExpA	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, **Follow(X)**,)	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, **in**,)
E1	**Follow(Exp)**,),	**end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, Follow(X), Follow(Seq_Exp),**)
T	+, -, **Follow(ExpA)**	+, -, **end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, Follow(X),)**
T1	+, -, **Follow(E1)**	+, -, **Follow(Exp),)**
F	*, /, **Follow(T)**	* , /, **+, -, Follow(ExpA)**
Y	*, /, **Follow(T1)**	*, /, **+, -, Follow(E1)**
OPA	var, exp_const, (	-
OPM	var, exp_const, (	-
OPP	(	-
Seq_Exp	)	-
Seq_Var	)	-

Sono ancora stati effettuati dei cambiamenti, pertanto è necessario andare a eseguire un ulteriore passo dell'algoritmo.

Passo 4

Non terminale	**Follow Precendete**	**FollowCorrente**
Prog	$, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,)	-

Bind	in	-
X	in	-
Exp	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,)	-
ExpA	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,)	-
E1	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, **Follow(X)**, Follow(Seq_Exp),)	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, **in**,)
T	+, -, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, **Follow(X)**,)	+, -, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, **in**,)
T1	+, -, **Follow(Exp)**,)	+, -, **end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,**)
F	* , /, +, -, **Follow(ExpA)**	* , /, +, -, **end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,)**
Y	*, /, +, -, **Follow(E1)**	*, /, +, -, **$, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, Follow(X), Follow(Seq_Exp),)**
OPA	var, exp_const, (	-
OPM	var, exp_const, (	-
OPP	(	-
Seq_Exp	)	-
Seq_Var	)	-

Passo 5

Non terminale	**Follow Precendete**	**FollowCorrente**
Prog	$, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,)	-

Bind	in	-
X	in	-
Exp	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,)	-
ExpA	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,)	-
E1	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,)	-
T	+, -, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,)	-
T1	+, -, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,)	-
F	* , /, +, -, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,)	-
Y	*, /, +, -, $, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, **Follow(X),** Follow(Seq_Exp),)	*, /, +, -, $, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, **in**,)
OPA	var, exp_const, (	-
OPM	var, exp_const, (	-
OPP	(	-
Seq_Exp	)	-
Seq_Var	)	-

Passo 6

Non terminale	**Follow Precendete**	**FollowCorrente**
Prog	$, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,)	-

Bind	in	-
X	in	-
Exp	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,)	-
ExpA	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,)	-
E1	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,)	-
T	+, -, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,)	-
T1	+, -, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,)	-
F	* , /, +, -, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,)	-
Y	*, /, +, -, $, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in,)	-
OPA	var, exp_const, (	-
OPM	var, exp_const, (	-
OPP	(	-
Seq_Exp	)	-
Seq_Var	)	-

Tabella di Parsing

N righe quanti sono i non terminali
M colonne quanti sono i terminali
La cella C[X,a] contiene la produzione X::=w quando *a* appartiene a First(w) oppure quando ϵ appartiene a First(w) e *a* appartiene al Follow(X)

\	$	let	in	end	letrec	var	=	and	lambda	(
Prog		1.1			1.2					
Bind						2.1				
X			3.2					3.1		
Exp		4.1			4.1	4.3			4.2	4.3
ExpA						5.1				5.1
E1		6.2	6.2	6.2	6.2	6.2		6.2	6.2	6.2
T						7.1				7.1
T1		8.2	8.2	8.2	8.2	8.2		8.2	8.2	8.2
F						9.1				9.3
Y		10.2	10.2	10.2	10.2	10.2		10.2	10.2	10.1 10.2
OPA										
OPM										
OPP										
Seq_Exp		14.1			14.1				14.1	14.1
Seq_Var						15.1				

\	)	if	then	else	exp_const	+	-	*	/	cons
Prog										
Bind										
X										
Exp		4.5			4.3					4.4
ExpA					5.1					
E1	6.2	6.2	6.2	6.2	6.2	6.1	6.1			6.2
T					7.1					
T1	8.2	8.2	8.2	8.2	8.2	8.2	8.2	8.1	8.1	8.2
F					9.2					

Y	10.2	10.2	10.2	10.2	10.2	10.2	10.2	10.2	10.2	10.2
OPA						11.1	11.2			
OPM								12.1	12.2	
OPP										13.1
Seq_Exp	14.2	14.1			14.1					14.1
Seq_Var	15.2									

\	car	cdr	eq	leq	atom
Prog					
Bind					
X					
Exp	4.4	4.4	4.4	4.4	4.4
ExpA					
E1	6.2	6.2	6.2	6.2	6.2
T					
T1	8.2	8.2	8.2	8.2	8.2
F					
Y	10.2	10.2	10.2	10.2	10.2
OPA					
OPM					
OPP	13.2	13.3	13.4	13.5	13.6
Seq_Exp	14.1	14.1	14.1	14.1	14.1
Seq_Var					

Dalla tabella si riesce a capire che la grammatica non è LL(1) in quanto per la cella [*Y,(*] sono presenti due produzioni.
Il che vuol dire che quando l'analizzatore sta espandendo il non terminale Y e incontra il carattere "(" non riesce a decidere quale produzione utilizzare tra 10.1 e 10.2 per proseguire l'analisi.

La produzione 10.1 potrebbe essere scelta perché "(" appartiene al First(Y), mentre 10.2 può essere scelta perché First(Y) contiene ϵ e "(" è presente nel Follow(Y).

Un modo per risolvere l'ambiguità è quello di rimuovere "(" dal Follow(Y).

Ripercorrendo i passi dell'algoritmo che calcola il Follow(Y) si trova che questo contiene "(" a causa del Follow(Exp).

A sua volta il Follow(Exp) contiene il First(Seq_Exp), questo a causa della produzione 14.1:

Seq_Exp ::= Exp Seq_Exp

Se ci fosse un separatore tra i vari elementi della sequenza, il Follow(Exp) andrebbe a contenere il First del carattere separatore piuttosto che il First(Seq_Exp), risolvendo così l'ambiguità.
Una possibile modifica alla produzione è la seguente:

Seq_Exp ::= Exp ; Seq_Exp

Tuttavia questa modifica impone che le stringhe prodotte da Seq_Exp abbiano la forma:

Exp ; Exp ; ... ; Exp ; ϵ

Mentre nella versione ambigua erano

Exp Exp … Exp ϵ

Per evitare di avere il ";" tra Exp e ϵ è possibile modificare ulteriormente la grammatica:

Seq_Exp ::= Exp Sepatator | ϵ
Separator ::= **;** Exp Separator | ϵ

In questo modo le stringhe generabili da Seq_Exp divenano:

Exp ; Exp ; ... ; Exp ϵ

Cioè non è più necessario il ";" tra l'ultimo Exp e ϵ.

Versione LL(1)

Per trasformare la grammatica in LL(1) si è scelto di proseguire con la seconda proposta, cioè di modificare la produzione 14 in:

Seq_Exp ::= Exp Exp_Sep | ϵ

e di aggiungere una nuova produzione

16 Exp_Sep ::= , Exp Exp_Sep | ϵ

La nuova grammatica diventa:

1 **Prog** ::= let Bind in Exp end | letrec Bind in Exp end
2 **Bind** ::= var = Exp X
3 **X** ::= and Bind | epsilon
4 **Exp** ::= Prog | lambda(Seq_Var) Exp | ExpA | OPP(Seq_Exp) | if Exp then Exp else Exp
5 **ExpA** ::= T E1
6 **E1** ::= OPA T E1 | epsilon
7 **T** ::= F T1
8 **T1** ::= OPM F T1 | epsilon
9 **F** ::= var Y | exp_const | (ExpA)
10 **Y** ::= (Seq_Exp) | epsilon
11 **OPA** ::= + |
12 **OPM** ::= * | /
13 **OPP** ::= cons | car | cdr | eq | leq | atom
14 **Seq_Exp** ::= Exp Exp_Sep| epsilon
15 **Seq_Var** ::= var Seq_Var | epsilon
16 **Exp_Sep** ::= , Exp Exp_Sep | ϵ

E' quindi necessario rieseguire il calcolo dei First e dei Follow.

Calcolo dei First

Per ogni produzione di tutti i non terminali X della grammatica, si considera il primo termine:

- se questo è un terminale, viene aggiunto al first(X)
- se questo è un non terminale Y, viene aggiunto il first(Y) al first(X)
    - se first(Y) contine ϵ, allora è necessario considerare anche il secondo termine della produzione
- si ripete finché non viene raggiunto un punto fisso

Indicativamente il first di un non terminale rappresenta l'insieme dei non terminale che possono comparire all'inizio di una produzione del non terminale.

Per come è definito il nuovo non terminale basta calcolare il suo First che risulta essere:
First(Exp_Sep) = { "," "ϵ"}

Il First di Seq_Exp non subisce modifiche dal momento che il First(Exp) non contiene ϵ.
E' stato necessario controllare il First(Exp) a causa della produzione Seq_Exp ::= Exp Exp_Sep. Se First(Exp) avesse contenuto ϵ, sarebbe stato necessario prendere in considerazione anche First(Exp_Sep)

Calcolo dei Follow

Passo 0

Follow_0(Start) = $
Follow_0(X) = vuoto, per ogni X non terminale

Passo 1

Follow_i = Follow_i-1

per ogni non terminale X e per ogni produzione della grammatica Y ::= w1Xw2
si aggiunge a Follow_i(X) First(w2) (tranne ϵ)
se ϵ appartiene al First(w2) allora si aggiunge a Follow_i(X), Follow_i-1(Y)

L'algoritmo termina quando si raggiunge un punto fisso

Non terminale	Follow Precendete	Follow Corrente
Prog	$	$, Follow(Exp)
Bind		in, Follow(X)
X		Follow(Bind)
Exp		end, and, Follow(Exp), Follow(X), then, else, "," ,Follow(Exp_Sep)
ExpA		Follow(Exp),),
E1		Follow(ExpA), Follow(E1)
T		+, -, Follow(E1)
T1		Follow(T), Follow(T1)
F		*, /, Follow(T1)
Y		Follow(F)
OPA		var, exp_const, (
OPM		var, exp_const, (
OPP		(
Seq_Exp		)
Seq_Var		), Follow(Seq_var)

Exp_Sep		Follow(Seq_Exp), Follow(Exp_Sep)

Non è ancora stato raggiunto un punto fisso, è necessaria un'altra iterazione

Passo 2

Non terminale	Follow Precendete	FollowCorrente
Prog	$, **Follow(Exp)**	$, **end, and, Follow(X), then, else, ",", Follow(Exp_Sep)**
Bind	in, **Follow(X),**	in, **Follow(Bind)**
X	**Follow(Bind)**	**in, Follow(X)**
Exp	end, and, Follow(X), then, else, "," , **Follow(Exp_Sep)**	end, and, Follow(Bind), then, else, **)**
ExpA	**Follow(Exp)**,),	**end, and, Follow(X), then, else, "," , Follow(Exp_Sep),**)
E1	**Follow(ExpA)**	**Follow(Exp),),**
T	+, -, **Follow(E1)**	+, -, **Follow(ExpA)**
T1	**Follow(T)**	**+, -, Follow(E1)**
F	*, /, **Follow(T1)**	*, /, **Follow(T)**
Y	**Follow(F)**	***, /, Follow(T1)**
OPA	var, exp_const, (	-
OPM	var, exp_const, (	-
OPP	(	-
Seq_Exp	)	-
Seq_Var	)	-
Exp_Sep	**Follow(Seq_Exp)**	**)**

Anche in questo caso ci sono state delle modifiche, quindi è necessario un ulteriore passo dell'algoritmo.

Passo 3

Non terminale	**Follow Precendete**	**FollowCorrente**

Prog	$, end, and, **Follow(X)**, then, else, “,”, Follow(Exp_Sep)	$, end, and, **in**, then, else, “,”,)
Bind	in	**-**
X	in	**-**
Exp	end, and, **Follow(Bind)**, then, else, “,”,)	end, and, **in**, then, else, “,”,)
ExpA	end, and, **Follow(X)**, then, else, Follow(Exp_Sep),)	end, and, **in**, then, else, “,”,)
E1	**Follow(Exp)**,)	**end, and, Follow(Bind), then, else, “,”,**)
T	+, -, **Follow(ExpA)**	+, -, **end, and, Follow(X), then, else, “,”, Follow(Exp_Sep),)**
T1	+, -, **Follow(E1)**	+, -, **Follow(Exp),)**
F	*, /, **Follow(T)**	*, /, **+, -, Follow(ExpA)**
Y	*, /, **Follow(T1)**	*, /, **+, -**, **Follow(E1)**
OPA	var, exp_const, (	-
OPM	var, exp_const, (	-
OPP	(	-
Seq_Exp	)	-
Seq_Var	)	-
Exp_Sep	)	-

Sono ancora stati effettuati dei cambiamenti, pertanto è necessario andare a eseguire un ulteriore passo dell’algoritmo.

Passo 4

Non terminale	**Follow Precendete**	**FollowCorrente**
Prog	$, end, and, in, then, else, “,”,)	$, end, and, in, then, else, “,”,)
Bind	in	-
X	in	-
Exp	end, and, in, then, else, “,”,)	end, and, in, then, else, “,”,)
ExpA	end, and, in, then, else, “,”,)	end, and, in, then, else, “,”,)

E1	end, and, **Follow(Bind)**, then, else, “,”)	end, and, **in**, then, else, “,”,)
T	end, and, **Follow(X)**, then, else, Follow(Exp_Sep), “,”,)	end, and, **in**, then, else, “,”)
T1	+, -, **Follow(Exp)**,)	+, -, **end, and, in, then, else, “,”**,)
F	*, /, +, -, **Follow(ExpA)**	*, /, +, -, **end, and, in, then, else, “,”,)**
Y	*, /, +, -, Follow(E1)	*, /, +, -, **end, and, Follow(Bind), then, else, “,”,)**
OPA	var, exp_const, (	-
OPM	var, exp_const, (	-
OPP	(	-
Seq_Exp	)	-
Seq_Var	)	-
Exp_Sep	)	-

Passo 5

Non terminale	**Follow Precendete**	**FollowCorrente**
Prog	$, end, and, in, then, else, “,”,)	-
Bind	in	-
X	in	-
Exp	end, and, in, then, else, “,”,)	-
ExpA	end, and, in, then, else, “,”,)	-
E1	end, and, in, then, else, “,”,)	-
T	end, and, in, then, else, “,”,)	-
T1	+, -, end, and, in, then, else, “,”,)	-
F	*, /, +, -, end, and, in, then, else, “,”,)	-
Y	*, /, +, -, end, and, **Follow(Bind)**, then, else, “,”,)	*, /, +, -, end, and, **in**, then, else, “,”)

OPA	var, exp_const, (	-
OPM	var, exp_const, (	-
OPP	(	-
Seq_Exp	)	-
Seq_Var	)	-
Exp_Sep	)	-

Passo 6

Non terminale	**Follow Precendete**	**FollowCorrente**
Prog	$, end, and, in, then, else, “,”,)	-
Bind	in	-
X	in	-
Exp	end, and, in, then, else, “,”,)	-
ExpA	end, and, in, then, else, “,”,)	-
E1	end, and, in, then, else, “,”,)	-
T	end, and, in, then, else, “,”,)	-
T1	+, -, end, and, in, then, else, “,”,)	-
F	*, /, +, -, end, and, in, then, else, “,”,)	-
Y	*, /, +, -, end, and, in, then, else, “,”,)	-
OPA	var, exp_const, (	-
OPM	var, exp_const, (	-
OPP	(	-
Seq_Exp	)	-
Seq_Var	)	-
Exp_Sep	)	-

L’esecuzione del passo non ha portato a cambiamenti, quindi il calcolo dei Follow è terminato.

Tabella di Parsing

N righe quanti sono i non terminali
M colonne quanti sono i terminali
La cella C[X,a] contiene la produzione X::=w quando *a* appartiene a First(w) oppure quando ϵ appartiene a First(w) e *a* appartiene al Follow(X)

\	$	let	in	end	letrec	var	=	and	lambda	(
Prog		1.1			1.2					
Bind						2.1				
X			3.2					3.1		
Exp		4.1			4.1	4.3			4.2	4.3
ExpA						5.1				5.1
E1			6.2	6.2				6.2		
T						7.1				7.1
T1			8.2	8.2				8.2		
F						9.1				9.3
Y			10.2	10.2				10.2	10.2	10.1
OPA										
OPM										
OPP										
Seq_Exp		14.1			14.1				14.1	14.1
Seq_Var						15.1				
Exp_Sep										

\	)	if	then	else	exp_const	+	-	*	/	cons
Prog										
Bind										

X										
Exp		4.5			4.3					4.4
ExpA					5.1					
E1	6.2		6.2	6.2		6.1	6.1			
T					7.1					
T1	8.2		8.2	8.2		8.2	8.2	8.1	8.1	
F					9.2					
Y	10.2		10.2	10.2		10.2	10.2	10.2	10.2	
OPA						11.1	11.2			
OPM								12.1	12.2	
OPP										13.1
Seq_Exp	14.2	14.1			14.1					14.1
Seq_Var	15.2									
Exp_Sep										

\	car	cdr	eq	leq	atom	,
Prog						
Bind						
X						
Exp	4.4	4.4	4.4	4.4	4.4	
ExpA						
E1						6.2
T						
T1						8.2
F						
Y						10.2
OPA						
OPM						

OPP	13.2	13.3	13.4	13.5	13.6	
Seq_Exp	14.1	14.1	14.1	14.1	14.1	
Seq_Var						
Exp_Sep						

Come atteso, nella nuova tabella di parsing non ci sono celle doppie, quindi la grammatica si conferma essere LL(1)