

# Parte 2

## Grammatica per LispKit

[La grammatica](#)

[Calcolo del First](#)

[Passo 0](#)

[Passo 1](#)

[Passo 2](#)

[Passo 3](#)

[Passo 4](#)

[Passo5](#)

[Calcolo dei Follow](#)

[Passo 0](#)

[Passo 1](#)

[Passo 2](#)

[Passo 3](#)

[Passo 4](#)

[Passo 5](#)

[Passo 6](#)

[Tabella di Parsing](#)

[Versione LL\(1\)](#)

[Calcolo dei Follow](#)

[Passo 0](#)

[Passo 1](#)

[Passo 2](#)

[Passo 3](#)

[Passo 4](#)

[Passo 5](#)

[Passo 6](#)

[Tabella di Parsing](#)

## 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 Precedete	First Corrente
Prog		let, letrec
Bind		var
X		and, €
Exp		First(Prog), lambda, First(ExpA), First(OPP), if
ExpA		First(T)
E1		First(OPA), €
T		First(F)
T1		First(OPM), €
F		var, exp_const. (
Y		(, €
OPA		+, -
OPM		*, /
OPP		cons, car, cdr, eq, leq, atom
Seq_Exp		First(Exp), €

Seq_Var		var, €
---------	--	--------

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

### Passo 2

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

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, €	-
Exp	lambda, if, let, letrec, <b>First(ExpA)</b> , cons. car, cdr, eq, leq, atom	lambda, if, let, letrec, <b>First(T)</b> , cons. car, cdr, eq, leq, atom
ExpA	<b>First(T)</b>	<b>var, exp_const. (</b>
E1	+, -, €	-
T	var, exp_const. (	-
T1	*, /, €	-
F	var, exp_const. (	-
Y	(, €	-
OPA	+, -	-
OPM	*, /	-
OPP	cons, car, cdr, eq, leq, atom	-
Seq_Exp	lamda, if, <b>First(Prog)</b> , <b>First(ExpA)</b> , <b>First(OPP)</b> , €	lamda, if, <b>let, letrec</b> , <b>First(T)</b> , <b>cons, car, cdr, eq, leq, atom</b> , €
Seq_Var	var, €	-

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, €	-
Exp	lambda, if, let, letrec, <b>First(T)</b> , cons. car, cdr, eq, leq, atom	lambda, if, let, letrec, <b>var, exp_const, (</b> , cons. car, cdr, eq, leq, atom
ExpA	var, exp_const. (	-
E1	+, -, €	-
T	var, exp_const. (	-
T1	*, /, €	-
F	var, exp_const. (	-

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

### Passo5

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

## Calcolo dei Follow

### Passo 0

$\text{Follow}_0(\text{Start}) = \$$

$\text{Follow}_0(X) = \text{vuoto}$ , per ogni  $X$  non terminale

### Passo 1

$\text{Follow}_i = \text{Follow}_{i-1}$

per ogni non terminale  $X$  e per ogni produzione della grammatica  $Y ::= w_1 X w_2$

si aggiunge a  $\text{Follow}_i(X)$   $\text{First}(w_2)$  (tranne  $\epsilon$ )

se  $\epsilon$  appartiene al  $\text{First}(w_2)$  allora si aggiunge a  $\text{Follow}_i(X)$ ,  $\text{Follow}_{i-1}(Y)$

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, <b>Follow(X)</b> , <b>Follow(Seq_Exp)</b> )	\$, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, <b>in</b> , )
Bind	in	-
X	in	-
Exp	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, <b>Follow(Bind)</b> , )	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, <b>in</b> , )
ExpA	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, <b>Follow(X)</b> , )	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, <b>in</b> , )
E1	<b>Follow(Exp)</b> , )	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, <b>Follow(X)</b> , <b>Follow(Seq_Exp)</b> , )
T	+, -, <b>Follow(ExpA)</b>	+, -, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, <b>Follow(X)</b> , )
T1	+, -, <b>Follow(E1)</b>	+, -, <b>Follow(Exp)</b> , )
F	*, /, <b>Follow(T)</b>	*, /, +, -, <b>Follow(ExpA)</b>
Y	*, /, <b>Follow(T1)</b>	*, /, +, -, <b>Follow(E1)</b>
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, <b>Follow(X)</b> , <b>Follow(Seq_Exp)</b> , )	end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, <b>in</b> , )
T	+, -, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, <b>Follow(X)</b> , )	+, -, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, <b>in</b> , )
T1	+, -, <b>Follow(Exp)</b> , )	+, -, <b>end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in, )</b>
F	<b>*</b> , /, +, -, <b>Follow(ExpA)</b>	<b>*</b> , /, +, -, <b>end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, in, )</b>
Y	<b>*</b> , /, +, -, <b>Follow(E1)</b>	<b>*</b> , /, +, -, <b>\$, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, Follow(X), Follow(Seq_Exp), )</b>
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, <b>Follow(X), Follow(Seq_Exp),</b> )	*, /, +, -, \$, end, and, then, else, lamda, if, let, letrec, var, exp_const. (, cons, car, cdr, eq, leq, atom, <b>in,</b> )
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  $\text{First}(w)$  oppure quando  $\epsilon$  appartiene a  $\text{First}(w)$  e  $a$  appartiene al  $\text{Follow}(X)$

\	\$	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 € 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:

$$\text{Seq\_Exp} ::= \text{Exp Seq\_Exp}$$

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:

$$\text{Seq\_Exp} ::= \text{Exp} ; \text{Seq\_Exp}$$

Tuttavia questa modifica impone che le stringhe prodotte da Seq\_Exp abbiano la forma:

$$\text{Exp} ; \text{Exp} ; \dots ; \text{Exp} ; \epsilon$$

Mentre nella versione ambigua erano

$$\text{Exp Exp} \dots \text{Exp} \epsilon$$

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

$$\begin{aligned} \text{Seq\_Exp} &::= \text{Exp Separator} \mid \epsilon \\ \text{Separator} &::= ; \text{Exp Separator} \mid \epsilon \end{aligned}$$

In questo modo le stringhe generabili da Seq\_Exp divenano:

$$\text{Exp} ; \text{Exp} ; \dots ; \text{Exp} \epsilon$$

Cioè non è più necessario il ";" tra l'ultimo Exp e  $\epsilon$ .

## Versione LL(1)

Per trasformare la grammatica in LL(1) si è scelto di proseguire con la prima proposta, cioè di modificare la produzione 14.1 nel seguente modo:

$$\text{Seq\_Exp} ::= \text{Exp} ; \text{Seq\_Exp}$$

Utilizzando questa modifica non vengono alterati i First, rimane quindi valida la tabella dei First per la grammatica originale.

I Follow invece subiscono varie modifiche e viene quindi riproposta l'esecuzione dell'algoritmo

## Calcolo dei Follow

### Passo 0

$\text{Follow}_0(\text{Start}) = \$$

$\text{Follow}_0(X) = \text{vuoto}$ , per ogni X non terminale

### Passo 1

$\text{Follow}_i = \text{Follow}_{i-1}$

per ogni non terminale X e per ogni produzione della grammatica  $Y ::= w_1 X w_2$

si aggiunge a  $\text{Follow}_i(X)$   $\text{First}(w_2)$  (tranne  $\epsilon$ )

se  $\epsilon$  appartiene al  $\text{First}(w_2)$  allora si aggiunge a  $\text{Follow}_i(X)$ ,  $\text{Follow}_{i-1}(Y)$

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(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	\$, <b>Follow(Exp)</b>	\$, end, and, <b>Follow(X)</b> , then, else, ;, <b>Follow(Seq_Exp)</b>
Bind	in, <b>Follow(X)</b> ,	in, <b>Follow(Bind)</b>
X	<b>Follow(Bind)</b>	in, <b>Follow(X)</b>
Exp	end, and, <b>Follow(X)</b> , then, else, ;, <b>Follow(Seq_Exp)</b>	end, and, <b>Follow(Bind)</b> , then, else, ;, )
ExpA	<b>Follow(Exp)</b> , ),	end, and, <b>Follow(X)</b> , then, else, ;, <b>Follow(Seq_Exp)</b> , )
E1	<b>Follow(ExpA)</b>	<b>Follow(Exp)</b> , ),
T	+, -, <b>Follow(E1)</b>	+, -, <b>Follow(ExpA)</b>
T1	<b>Follow(T)</b>	+, -, <b>Follow(E1)</b>
F	*, /, <b>Follow(T1)</b>	*, /, <b>Follow(T)</b>
Y	<b>Follow(F)</b>	*, /, <b>Follow(T1)</b>
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, <b>Follow(X)</b> , then, else, ;, <b>Follow(Seq_Exp)</b>	\$, end, and, <b>in</b> , then, else, ;, )
Bind	in	-
X	in	-
Exp	end, and, <b>Follow(Bind)</b> , then, else, ;, )	end, and, <b>in</b> , then, else, ;, )
ExpA	end, and, <b>Follow(X)</b> , then, else, ;, <b>Follow(Seq_Exp)</b> , )	end, and, <b>in</b> , then, else, ;, )
E1	<b>Follow(Exp)</b> , )	<b>end, and, Follow(Bind), then, else, ;, )</b>
T	+, -, <b>Follow(ExpA)</b>	end, and, Follow(X), then, else, ;, Follow(Seq_Exp), )
T1	+, -, <b>Follow(E1)</b>	+, -, <b>Follow(Exp)</b> , )
F	*, /, <b>Follow(T)</b>	*, /, +, -, <b>Follow(ExpA)</b>
Y	*, /, <b>Follow(T1)</b>	*, /, +, -, <b>Follow(E1)</b>
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, 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, <b>Follow(Bind)</b> , then, else, ;, )	end, and, <b>in</b> , then, else, ;, )

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

### 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, <b>Follow(Bind)</b> , then, else, ;, )	*, /, +, -, end, and, <b>in</b> , then, else, ;, )
OPA	var, exp_const, (	-
OPM	var, exp_const, (	-
OPP	(	-

Seq_Exp	)	-
Seq_Var	)	-

### 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	)	-

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  $\epsilon$  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				

\	)	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									

\	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						

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