Formal definition of the modified LispKit grammar

G = (NT, T, R, S) where:
- NT = {Prog Bind X Exp ExpA E1 T T1 F Y OPA OPM OPP Seq_Exp Lst_Exp Seq_Var}
- T = {let letrec in end = and lambda if then else exp_const ( ) + - * / cons car cdr eq leq atom , var}
- S ∈ NT, S = Prog
- R ∈ ℘(NT × (NT ∪ T ∪ {ε})) is identified by:

Prog ::= let Bind in Exp end | letrec Bind in Exp | ε
Bind ::= var = Exp X
X ::= and Bind | ε
Exp ::= Prog | lambda (Seq_Var) Exp | ExpA | OPP (Seq_Exp) | if Exp then Exp else Exp
ExpA ::= T E_1
E_1 ::= OPA T E_1 | ε
T ::= F T_1
T_1 ::= OPM F T_1 | ε
F ::= var Y | exp_const | (ExpA)
Y ::= (Seq_Exp) | ε
OPA ::= + | -
OPM ::= * | /
OPP ::= cons | car | cdr | eq | leq | atom
Seq_Exp ::= Exp Lst_Exp | ε
Lst_Exp ::= , Seq_Exp | ε
Seq_Var ::= var Seq_Var | ε

First and Follow for the modified LispKit grammar

Non Terminal	First Set	Follow Set
Prog	let letrec	) end and then else , in
Bind	var	in
X	and ε	in
Exp	lambda if let letrec cons car cdr eq leq atom var exp_const (	) end and then else , in
ExpA	var exp_const (	) end and then else , in
E1	+ - ε	) end and then else , in
T	var exp_const (	+ -) end and then else , in
T1	* / ε	+ -) end and then else , in
F	var exp_const (	* / + -) end and then else , in
Y	(ε	* / + -) end and then else , in
OPA	+ -	var exp_const (
OPM	* /	var exp_const (
OPP	cons car cdr eq leq atom	(
Seq_Exp	lambda if let letrec cons car cdr eq leq atom var exp_const (ε	)
Lst_Exp	, ε	)
Seq_Var	var ε	)