19CSE401 - Compiler Design

Programming Language : Racket

Name	Roll Number
Raghul K B	CB.EN.U4CSE19346
O Mahanth	CB.EN.U4CSE19340
V Devakumar	CB.EN.U4CSE19358
V Nithin Krishna	CB.EN.U4CSE19360

CFG - Formal Definition :

Definition − A context-free grammar (CFG) consisting of a finite set of grammar rules is a quadruple (N, T, P, S) where

N - a set of non-terminal symbols.
T - a set of terminals where N ∩ T = NULL.
P - a set of production rules
S - the start symbol.

N = { program, defOrExpr, definition, expr, quoted, quasiQuoted, testCase, libraryRequire, pkg, name, symbol, number, bool, string, character, arthoperators, reloperators **}**

T = { BEGINN, BEGINN0, SETNQ, SET, DELAY, CAR, CDR, COMBINATIONS, LIST, REVERSE, IF, AND, OR, COND, ELSE, APPEND, LAMBDA, LAMBDASYM, LOCAL, LETREC, SHARED, LET, LETSTAR, RECUR, TSCHECKEXP, TSCHECKRAND, TSCHECKWITHIN, TSCHECKMEMBEROF, TSCHECKSATSIS, TSCHECKERROR, REQUIRE, DISPLAY, DEFINE, NEWLINE, EMPTY, DEFINESTRUCT, QSMARK, QUOTESX, QUOTEQUOTED, LEFTB, RIGHTB, LEFTSQB, RIGHTSQB, QUOTEQUASIQUOTED, CHARACTERQUOTED, ARTHOPERATORS, BOOLEAN, RELOPERATORS, SYMBOL, INT, DECIMAL, NAME, COMMA, COMMAAT, STRING, CHARACTER, LANG, COMMENT, WS
}

P = {

```
start :program;

program : defOrExpr program EOF
      |defOrExpr
      ;

defOrExpr : definition
      | expr
      | testCase
      | libraryRequire
      ;
nameplus : name nameplus
      | name
      ;
namestar : name namestar
               |
               ;
definitionstar : definition definitionstar
                        |
                        ;
definition : LEFTB DEFINE name expr RIGHTB
       | LEFTB DEFINE LEFTB name nameplus RIGHTB expr RIGHTB
       | LEFTB DEFINESTRUCT name LEFTB namestar RIGHTB RIGHTB
       ;
exprplus : expr exprplus
               | expr
               ;
lner : LEFTSQB name expr RIGHTSQB lner
        |
        ;
leerbplus : LEFTB expr expr RIGHTB leerbplus
                | LEFTB expr expr RIGHTB
                ;
leersqbplus : LEFTSQB expr expr RIGHTSQB leersqbplus
                      | LEFTSQB expr expr RIGHTSQB
                      ;
leersqbstar : LEFTSQB expr expr RIGHTSQB leersqbstar
                      |
                      ;
expr: LEFTB BEGINN exprplus RIGHTB
   | LEFTB BEGINN0 exprplus RIGHTB
   | LEFTB SETNQ NAME expr RIGHTB
```

```
    | LEFTB DELAY expr RIGHTB
    | LEFTB CAR expr RIGHTB
    | LEFTB CDR expr RIGHTB
    | LEFTB COMBINATIONS expr RIGHTB
    | LEFTB LIST expr RIGHTB
    | LEFTB REVERSE expr RIGHTB
    | LEFTB APPEND NAME expr RIGHTB
    | LEFTB LAMBDA LEFTB namestar RIGHTB expr RIGHTB
    | LEFTB LAMBDASYM LEFTB namestar RIGHTB expr RIGHTB
    | LEFTB LOCAL LEFTSQB definitionstar RIGHTSQB expr RIGHTB
    | LEFTB LETREC LEFTB lner RIGHTB expr RIGHTB
    | LEFTB SHARED LEFTB lner RIGHTB expr RIGHTB
    | LEFTB LET LEFTB lner RIGHTB expr RIGHTB
    | LEFTB LETSTAR LEFTB lner RIGHTB expr RIGHTB
    | LEFTB RECUR name LEFTB lner RIGHTB expr RIGHTB
    | LEFTB name exprplus RIGHTB
    | LEFTB COND leerbplus RIGHTB
    | LEFTB COND leersqbplus RIGHTB
    | LEFTB COND leersqbstar LEFTSQB ELSE expr RIGHTSQB RIGHTB
    | LEFTB IF  expr expr expr RIGHTB
    | LEFTB AND expr exprplus RIGHTB
    | LEFTB OR  expr exprplus RIGHTB
    | DISPLAY name
    | DISPLAY string
    | NEWLINE
    | EMPTY
    | QUOTESX
    | QSMARK
    | QUOTEQUOTED quoted
    | QUOTEQUASIQUOTED quasiQuoted
    | CHARACTERQUOTED
    | reloperators
    | arthoperators
    | name
    | number
    | symbol
    | bool
    | string
    | character
    ;
quotedstar : quoted quotedstar
                 |
                 ;
quoted
```

```
    : name
    | string
    | character
    | LEFTB quotedstar RIGHTB
    | QUOTEQUOTED quoted
    | QUOTEQUASIQUOTED quoted
    | COMMA quoted
    | COMMAAT quoted
    ;
quasiQuotedstar : quasiQuoted quasiQuotedstar
                            |
                            ;
quasiQuoted
    : name
    | number
    | string
    | character
    | LEFTB quasiQuotedstar RIGHTB
    | QUOTEQUOTED quasiQuoted
    | QUOTEQUASIQUOTED quasiQuoted
    | COMMA expr
    | COMMAAT expr
    ;
exprquestionmark : expr
                             |
                             ;
testCase
    : LEFTB TSCHECKEXP expr expr RIGHTB
    | LEFTB TSCHECKRAND expr expr RIGHTB
    | LEFTB TSCHECKWITHIN expr expr expr RIGHTB
    | LEFTB TSCHECKMEMBEROF expr exprplus RIGHTB
    | LEFTB TSCHECKSATSIS expr name RIGHTB
    | LEFTB TSCHECKERROR expr exprquestionmark RIGHTB
    ;
stringplus : string stringplus
                     | string
                     ;
lsprquestionmark : LEFTB stringplus RIGHTB
                            |
                            ;
libraryRequire
    : LEFTB REQUIRE STRING RIGHTB
    | LEFTB REQUIRE name RIGHTB
    | LEFTB REQUIRE LEFTB name STRING lsprquestionmark RIGHTB RIGHTB
```

| *LEFTB REQUIRE LEFTB* name *STRING* pkg *RIGHTB RIGHTB*
;

pkg: *LEFTB* string string number number *RIGHTB*;

name: *NAME*;

symbol : *SYMBOL*;

number : *INT*|*DECIMAL*;

bool : *BOOLEAN*;

string : *STRING*;

character : *CHARACTER* ;

reloperators : *RELOPERATORS*;

arthoperators : *ARTHOPERATORS*;

}

S = program

CFG - Production Rules Table :

Non Terminals	General Format	Production
program	#lang racket (<expression-statements>) or (<definition-statements>) EOF	**->** defOrExpr program *EOF* |defOrExpr ;
defOrExpr	—	**->** definition | expr | testCase | libraryRequire ;

definition	(define <name> <expression-statements>) or (define (<name> <variable-name>) <expression-statements>) or (define-struct (<name> (<variables>))	**->** *LEFTB DEFINE* name expr *RIGHTB* | *LEFTB DEFINE LEFTB* name nameplus *RIGHTB* expr *RIGHTB* | *LEFTB DEFINESTRUCT* name *LEFTB* namestar *RIGHTB RIGHTB* ;
expr	(begin <expression-statements> ...+) or (begin0 <expression-statements> ...+) or (set! <id-name> <expression-statements>) or (delay <expression-statements>) or (car <expression-statements>) or (cdr <expression-statements>) or (list <expression-statements>)	**->** *LEFTB BEGINN* exprplus *RIGHTB* | *LEFTB BEGINN0* exprplus *RIGHTB* | *LEFTB SETNQ NAME* expr *RIGHTB* | *LEFTB DELAY* expr *RIGHTB* | *LEFTB CAR* expr *RIGHTB* | *LEFTB CDR* expr *RIGHTB* | *LEFTB COMBINATIONS* expr *RIGHTB* | *LEFTB LIST* expr *RIGHTB* | *LEFTB REVERSE* expr *RIGHTB* | *LEFTB APPEND NAME* expr *RIGHTB* | *LEFTB LAMBDA LEFTB* namestar *RIGHTB* expr *RIGHTB* | *LEFTB LAMBDASYM LEFTB* namestar *RIGHTB* expr *RIGHTB* | *LEFTB LOCAL LEFTSQB* definitionstar *RIGHTSQB* expr *RIGHTB* | *LEFTB LETREC LEFTB* Iner *RIGHTB* expr *RIGHTB*

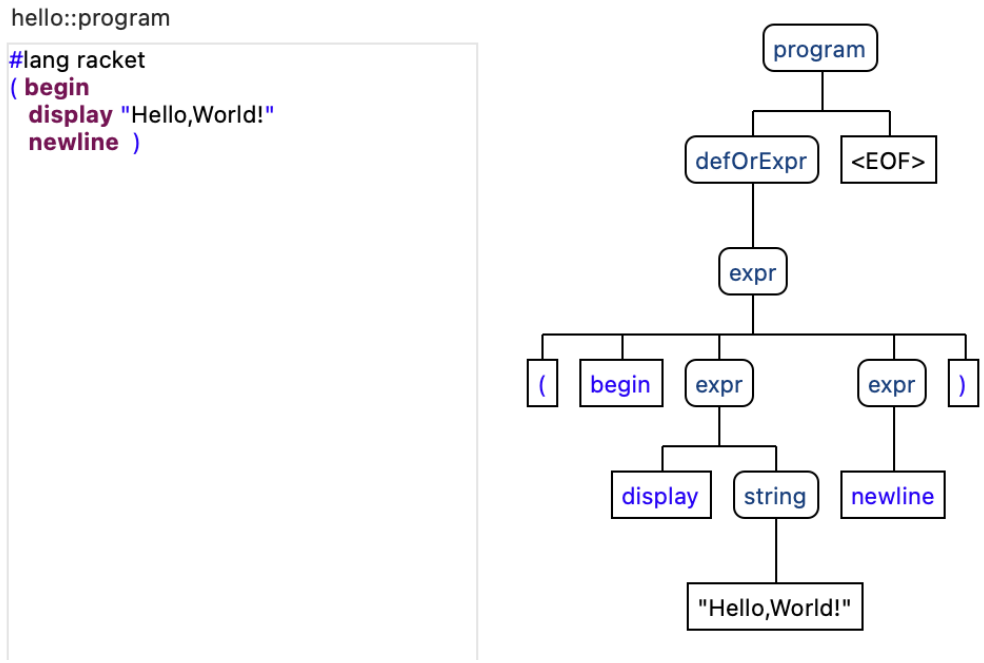
	or (reverse <expression-statements>) or (append <expression-statements>) or (lambda <id-name> <expression-statements>) or (local [<definition-statements>] <expression-statements>) or (let ([<id-name> <expression-statements>]) <expression-statements>) or (recur <id-name> ([<func-id-name> <expression-statements>] <expression-statements) or (cond <cond-clause>) <cond-clause>= [test-expr then <expression-statements>] | [else then <expression-statements>] | [test-expr => proc-expr] | [test-expr] or (if <test-expression> <then-expression> <else-expression>)	| *LEFTB SHARED LEFTB* lner *RIGHTB* expr *RIGHTB* | *LEFTB LET LEFTB* lner *RIGHTB* expr *RIGHTB* | *LEFTB LETSTAR LEFTB* lner *RIGHTB* expr *RIGHTB* | *LEFTB RECUR* name *LEFTB* lner *RIGHTB* expr *RIGHTB* | *LEFTB* name exprplus *RIGHTB* | *LEFTB COND* leerbplus *RIGHTB* | *LEFTB COND* leersqbplus *RIGHTB* | *LEFTB COND* leersqbstar *LEFTSQB ELSE* expr *RIGHTSQB RIGHTB* | *LEFTB IF* expr expr expr *RIGHTB* | *LEFTB AND* expr exprplus *RIGHTB* | *LEFTB OR* expr exprplus *RIGHTB* | *DISPLAY* name | *DISPLAY* string | *NEWLINE* | *EMPTY* | *QUOTESX* | *QSMARK* | *QUOTEQUOTED* quoted | *QUOTEQUASIQUOTED* quasiQuoted | *CHARACTERQUOTED* | reloperators | arthoperators | name | number | symbol | bool | string | character ;

	or (and <expression-statements>) or (or <expression-statements>)	
testCase	(check-expect <expression-statements> <expression-statements>) or (check-within <expression-statements> <expression-statements> <expression-statements>) or (check-member-of <expression-statements> <expression-statements> ...) or (check-range <expression-statements> <expression-statements> <expression-statements>) or (check-error <expression-statements>) or (check-random <expression-statements> <expression-statements>) or	-> *LEFTB TSCHECKEXP* expr expr *RIGHTB* | *LEFTB TSCHECKRAND* expr expr *RIGHTB* | *LEFTB TSCHECKWITHIN* expr expr expr *RIGHTB* | *LEFTB TSCHECKMEMBEROF* expr exprplus *RIGHTB* | *LEFTB TSCHECKSATSIS* expr name *RIGHTB* | *LEFTB TSCHECKERROR* expr exprquestionmark *RIGHTB* ;

	(check-satisfied <expression-statements> <name>)	
libraryRequire	(require <library-name>)	**->** *LEFTB REQUIRE STRING RIGHTB* | *LEFTB REQUIRE* name *RIGHTB* | *LEFTB REQUIRE LEFTB* name *STRING* lsprquestionmark *RIGHTB RIGHTB* | *LEFTB REQUIRE LEFTB* name *STRING* pkg *RIGHTB RIGHTB* ;
pkg	require(‘<expression_statements>’)	*LEFTB* string string number number *RIGHTB*;
name	—	**->** NAME ;
number	—	**->** NUMBER ;
bool	—	**->** BOOLEAN;
arthoperators	—	**->** ADD |SUB |MULT |DIV ;
reloperators	—	**->** LT |EQ |GT |LTE |GTE |NEQ ;

Parse Trees :

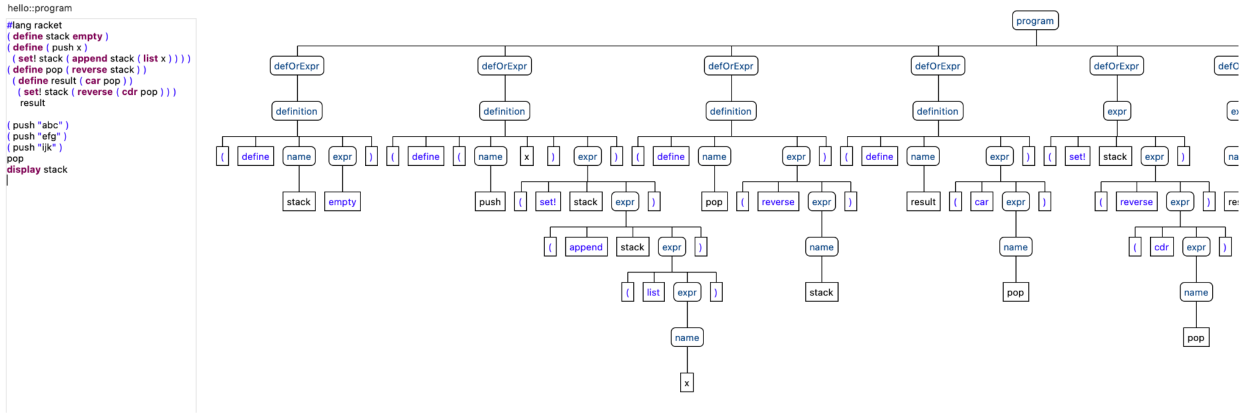
1) Hello World Program :



hello::program

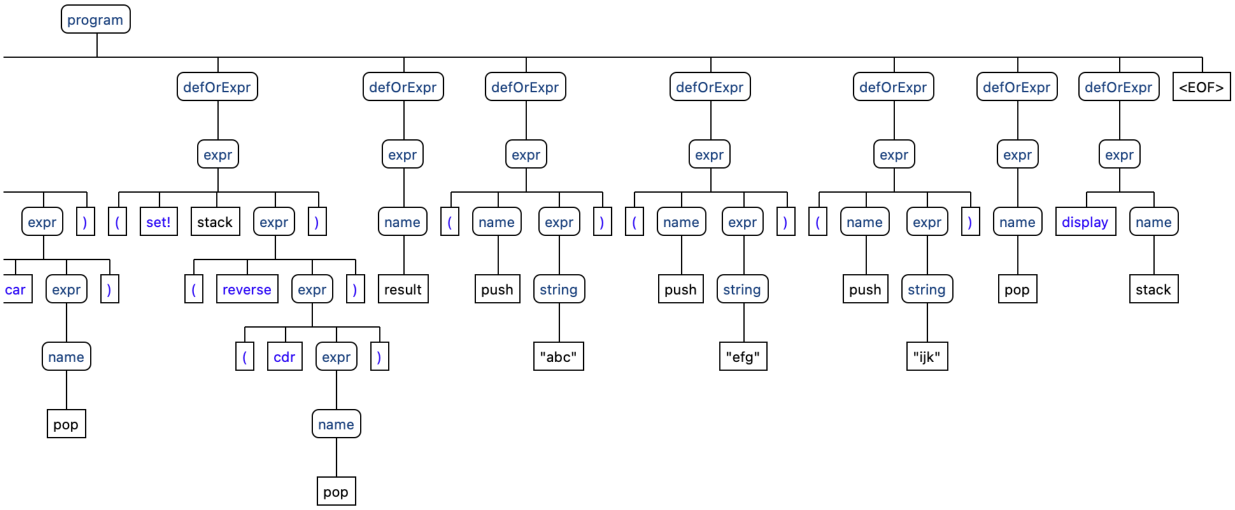
```
#lang racket
( begin
  display "Hello,World!"
  newline  )
```

2) Stack Program :

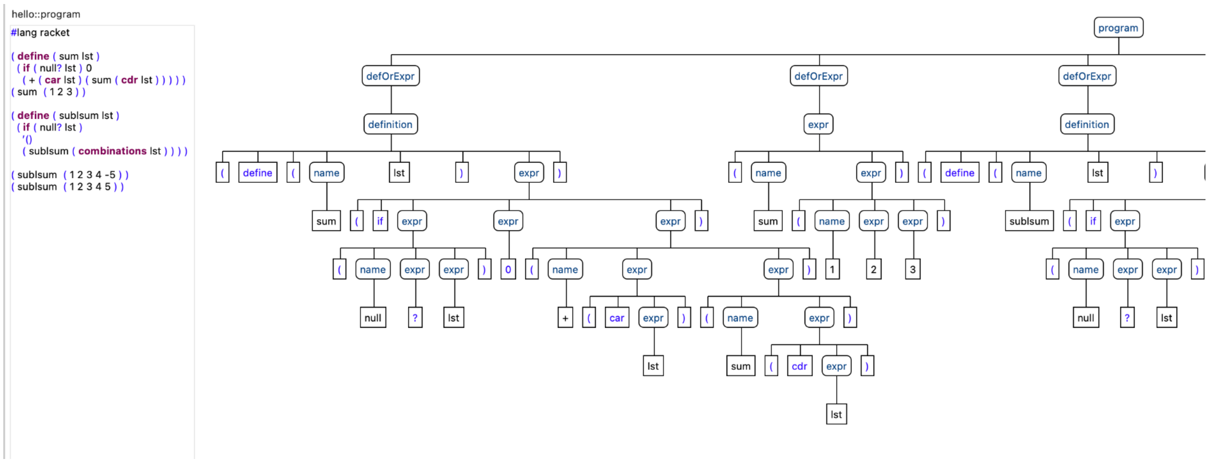


hello::program

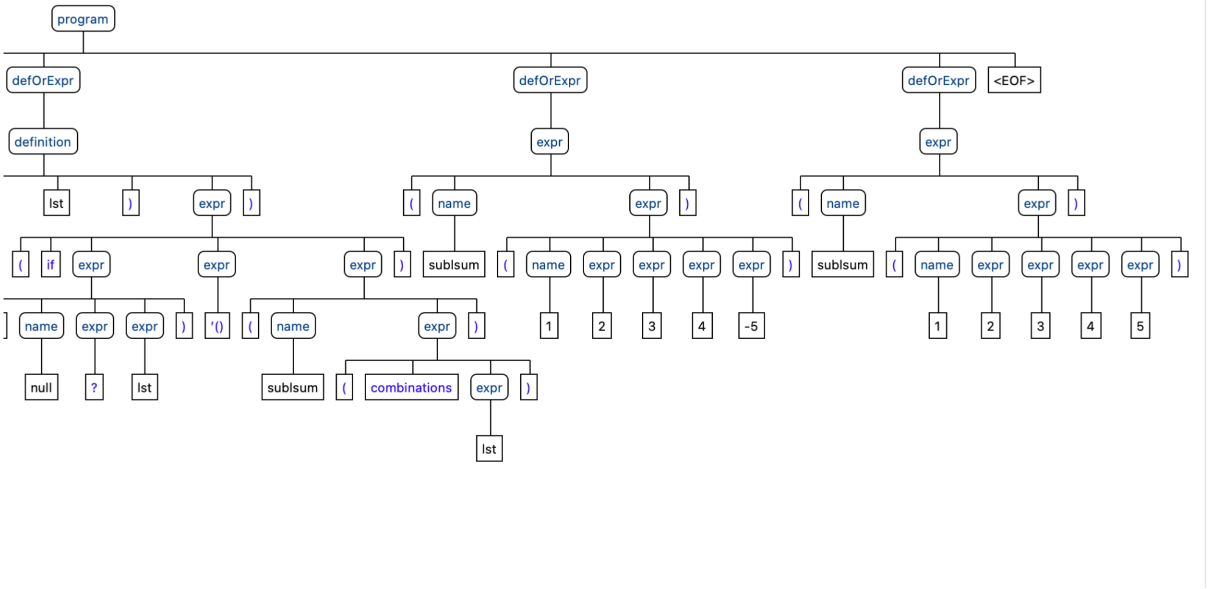
```
#lang racket
( define stack empty )
( define ( push x )
  ( set! stack ( append stack ( list x ) ) ) )
( define pop ( reverse stack ) )
  ( define result ( car pop ) )
    ( set! stack ( reverse ( cdr pop ) ) )
     result

( push "abc" )
( push "efg" )
( push "ijk" )
pop
display stack
```



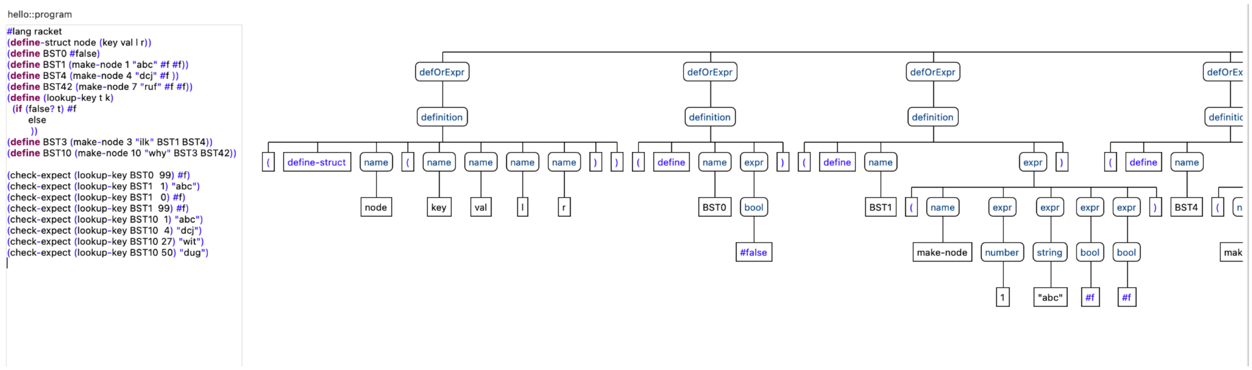
3) Sum of List Program :


hello::program

```
#lang racket

( define ( sum lst )
  ( if ( null? lst ) 0
    ( + ( car lst ) ( sum ( cdr lst ) ) ) ) ) )
( sum  ( 1 2 3 ) )

( define ( sublsum lst )
  ( if ( null? lst )
    '()
    ( sublsum ( combinations lst ) ) ) ) )

( sublsum  ( 1 2 3 4 -5 ) )
( sublsum  ( 1 2 3 4 5 ) )
```


program
defOrExpr defOrExpr defOrExpr <EOF>
definition expr expr
lst) expr) (name expr) (name expr)
(if expr expr expr) sublsum (name expr expr expr expr expr) sublsum (name expr expr expr expr expr)
name expr expr) '() (name expr) 1 2 3 4 -5 1 2 3 4 5
null ? lst sublsum (combinations expr)
lst

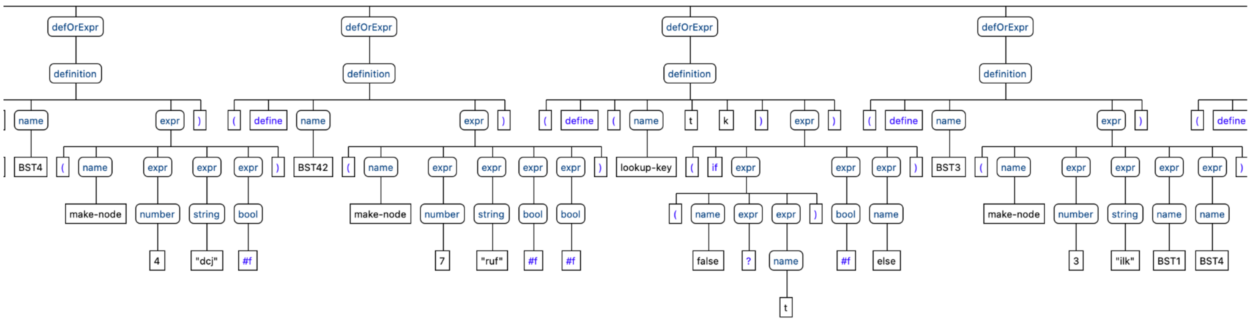
4) Lookup for key using test cases in BST Program :

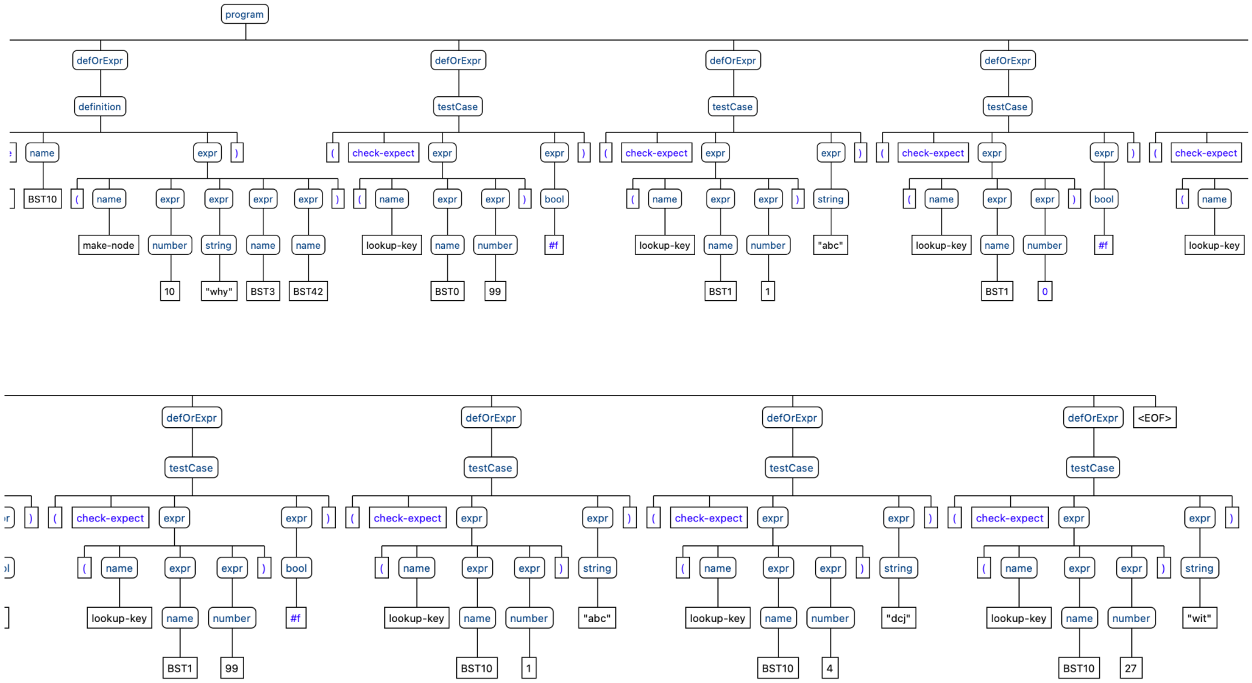


hello::program

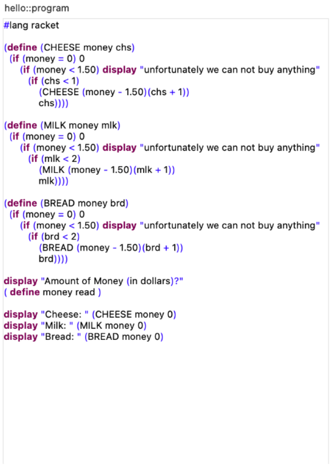
```
#lang racket
(define-struct node (key val l r))
(define BST0 #false)
(define BST1 (make-node 1 "abc" #f #f))
(define BST4 (make-node 4 "dcj" #f ))
(define BST42 (make-node 7 "ruf" #f #f))
(define (lookup-key t k)
  (if (false? t) #f
        else
         ))
(define BST3 (make-node 3 "ilk" BST1 BST4))
(define BST10 (make-node 10 "why" BST3 BST42))

(check-expect (lookup-key BST0  99) #f)
(check-expect (lookup-key BST1   1) "abc")
(check-expect (lookup-key BST1   0) #f)
(check-expect (lookup-key BST1  99) #f)
(check-expect (lookup-key BST10  1) "abc")
(check-expect (lookup-key BST10  4) "dcj")
(check-expect (lookup-key BST10 27) "wit")
(check-expect (lookup-key BST10 50) "dug")
```

defOrExpr defOrExpr defOrExpr defOrEx
definition definition definition definitic
(define-struct name (name name name name)) (define name expr) (define name expr) (define name
node key val l r BST0 bool BST1 (name expr expr expr expr) BST4 (n
#false make-node number string bool bool mak
1 "abc" #f #f


defOrExpr defOrExpr defOrExpr defOrExpr
definition definition definition definition
name expr) (define name expr) (define (name t k) expr) (define name expr) (define
BST4 (name expr expr expr) BST42 (name expr expr expr expr) lookup-key (if expr expr expr) BST3 (name expr expr expr expr)
make-node number string bool make-node number string bool bool (name expr expr) bool name make-node number string name name
4 "dcj" #f 7 "ruf" #f #f false ? name #f else 3 "ilk" BST1 BST4
t

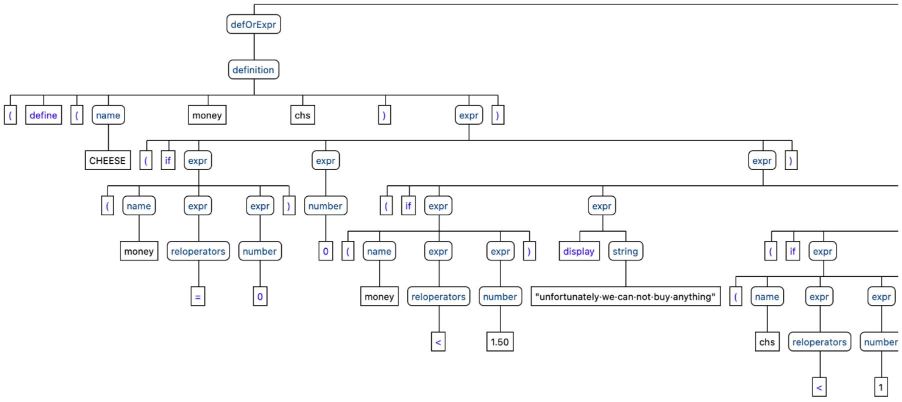

program
defOrExpr
definition
name
BST10
expr
)
(
name
make-node
expr
number
10
expr
string
"why"
expr
name
BST3
expr
name
BST42
)
defOrExpr
testCase
(
check-expect
expr
(
name
lookup-key
expr
name
BST0
expr
number
99
)
expr
bool
#f
)
defOrExpr
testCase
(
check-expect
expr
(
name
lookup-key
expr
name
BST1
expr
number
1
)
expr
string
"abc"
)
defOrExpr
testCase
(
check-expect
expr
(
name
lookup-key
expr
name
BST1
expr
number
0
)
expr
bool
#f
)
(
check-expect
(
name
lookup-key
defOrExpr
testCase
(
check-expect
expr
(
name
lookup-key
expr
name
BST1
expr
number
99
)
expr
bool
#f
)
defOrExpr
testCase
(
check-expect
expr
(
name
lookup-key
expr
name
BST10
expr
number
1
)
expr
string
"abc"
)
defOrExpr
testCase
(
check-expect
expr
(
name
lookup-key
expr
name
BST10
expr
number
4
)
expr
string
"dcj"
)
defOrExpr
testCase
(
check-expect
expr
(
name
lookup-key
expr
name
BST10
expr
number
27
)
expr
string
"wit"
)
<EOF>

5) Shopping Program :

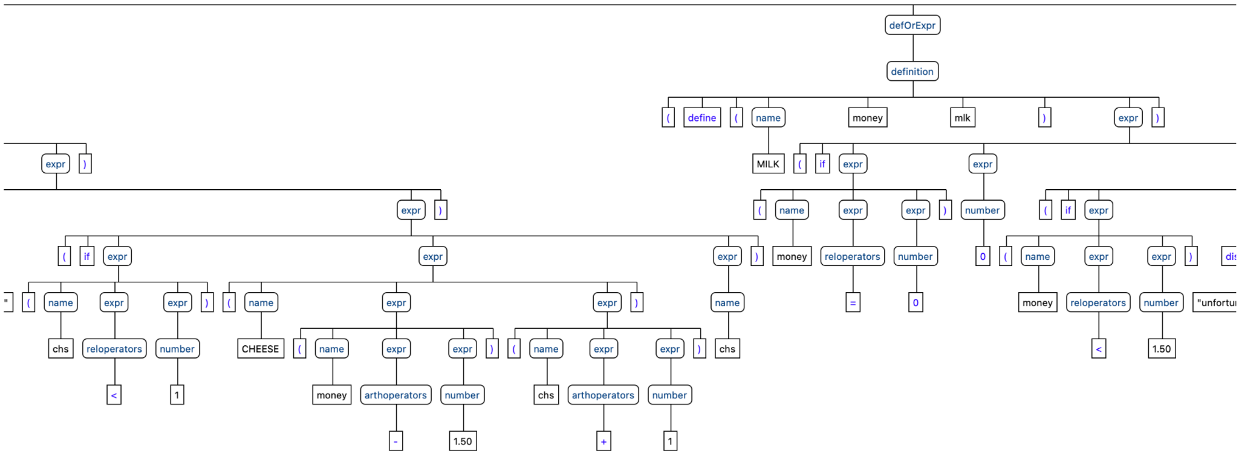


hello::program

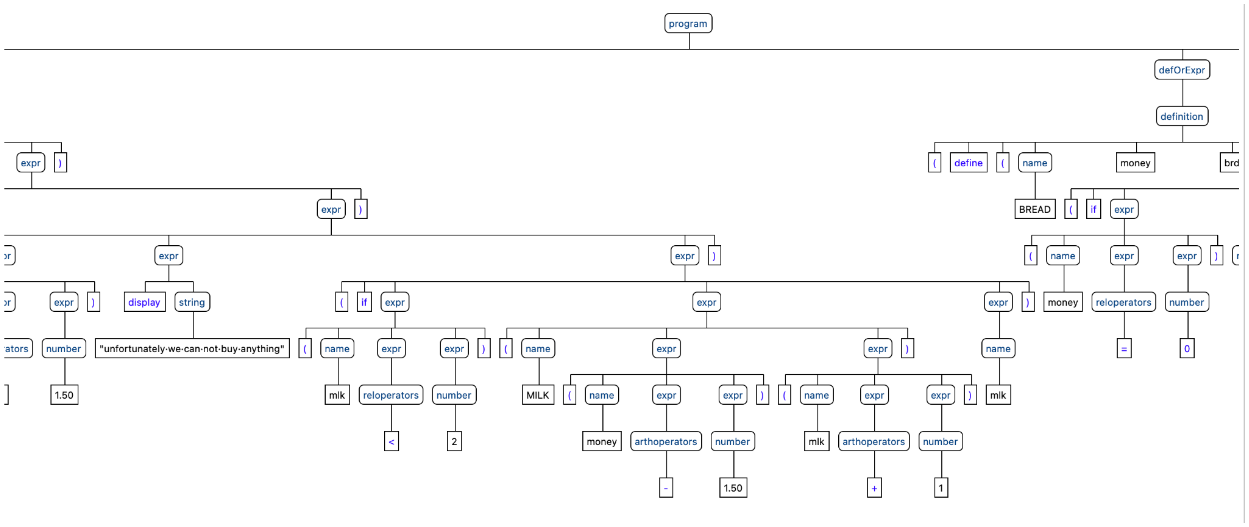
```
#lang racket

(define (CHEESE money chs)
  (if (money = 0) 0
     (if (money < 1.50) display "unfortunately we can not buy anything"
        (if (chs < 1)
           (CHEESE (money - 1.50)(chs + 1))
           chs))))

(define (MILK money mlk)
  (if (money = 0) 0
     (if (money < 1.50) display "unfortunately we can not buy anything"
        (if (mlk < 2)
           (MILK (money - 1.50)(mlk + 1))
           mlk))))

(define (BREAD money brd)
  (if (money = 0) 0
     (if (money < 1.50) display "unfortunately we can not buy anything"
        (if (brd < 2)
           (BREAD (money - 1.50)(brd + 1))
           brd))))

display "Amount of Money (in dollars)?"
( define money read )

display "Cheese: " (CHEESE money 0)
display "Milk: " (MILK money 0)
display "Bread: " (BREAD money 0)
```



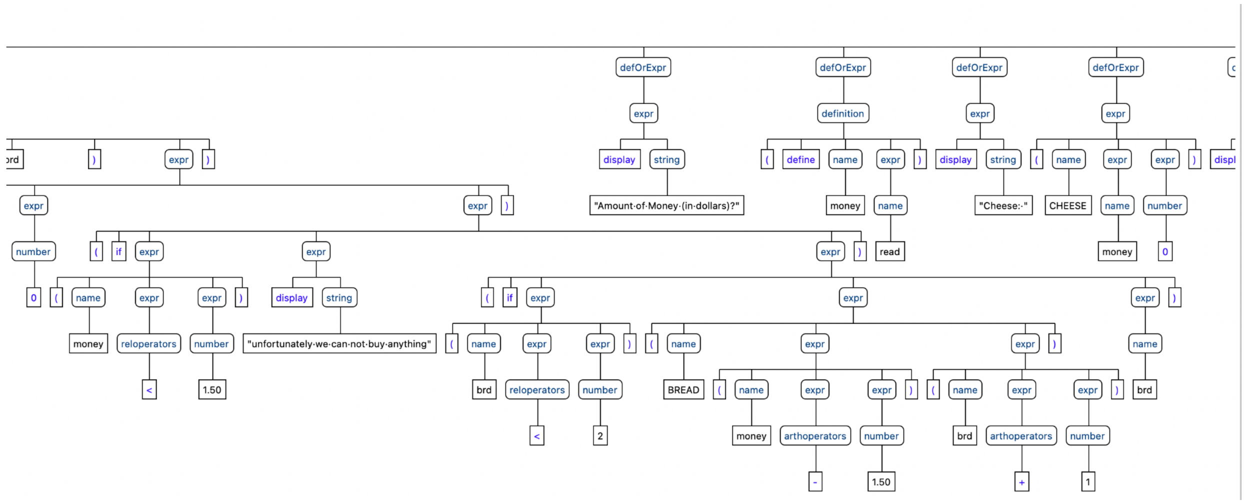
defOrExpr
definition
(define (name money chs) expr)
CHEESE (if expr expr expr)
(name expr expr) number (if expr expr (if expr
money reloperators number 0 (name expr expr) display string (name expr expr
= 0 money reloperators number "unfortunately-we-can-not-buy-anything" chs reloperators number
< 1.50 < 1

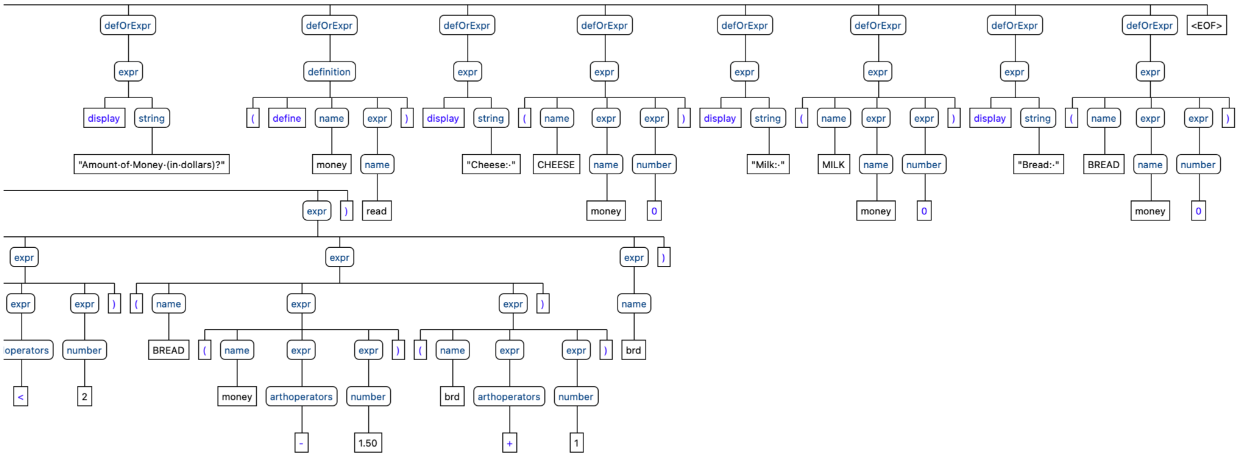


defOrExpr
definition
(define (name money mlk) expr)
expr) MILK (if expr expr
expr) (name expr expr) number (if expr
(if expr expr expr) money reloperators number 0 (name expr expr) dis
" (name expr expr) (name expr) name = 0 money reloperators number "unfortu
chs reloperators number CHEESE (name expr expr) (name expr expr) chs < 1.50
< 1 money arthoperators number chs arthoperators number
- 1.50 + 1



program
defOrExpr
definition
expr) (define (name money brd
expr) BREAD (if expr
pr expr expr) (name expr expr) (r
pr expr) display string (if expr expr expr) money reloperators number
ators number "unfortunately-we-can-not-buy-anything" (name expr expr) (name expr expr) name = 0
1.50 mlk reloperators number MILK (name expr expr) (name expr expr) mlk
< 2 money arthoperators number mlk arthoperators number
- 1.50 + 1


defOrExpr
defOrExpr
defOrExpr
defOrExpr
expr
definition
expr
expr
ord
)
expr
)
display
string
(
define
name
expr
)
display
string
(
name
expr
expr
)
displ
expr
expr
)
"Amount·of·Money·(in·dollars)?"
money
name
"Cheese:·"
CHEESE
name
number
number
(
if
expr
expr
expr
)
read
money
0
0
(
name
expr
expr
)
display
string
(
if
expr
expr
expr
)
money
reloperators
number
"unfortunately·we·can·not·buy·anything"
(
name
expr
expr
)
(
name
expr
expr
)
name
<
1.50
brd
reloperators
number
BREAD
(
name
expr
expr
)
(
name
expr
expr
)
brd
<
2
money
arthoperators
number
brd
arthoperators
number
-
1.50
+
1


defOrExpr
defOrExpr
defOrExpr
defOrExpr
defOrExpr
defOrExpr
defOrExpr
defOrExpr
<EOF>
expr
definition
expr
expr
expr
expr
expr
expr
display
string
(
define
name
expr
)
display
string
(
name
expr
expr
)
display
string
(
name
expr
expr
)
display
string
(
name
expr
expr
)
"Amount·of·Money·(in·dollars)?"
money
name
"Cheese:·"
CHEESE
name
number
"Milk:·"
MILK
name
number
"Bread:·"
BREAD
name
number
expr
)
read
money
0
money
0
money
0
expr
expr
expr
)
expr
expr
)
(
name
expr
expr
)
name
operators
number
BREAD
(
name
expr
expr
)
(
name
expr
expr
)
brd
<
2
money
arthoperators
number
brd
arthoperators
number
-
1.50
+
1