uc3m | Universidad Carlos III de Madrid

UNITS 7 AND 8: SEMANTIC ANALYSIS and ERROR HANDLING

We want to incorporate a repetitive sentence into a high-level language. The sentence can be represented by the following regular expression:

repeat (identifier | number) >> sentence⁺ <<

A program consists of at least one statement, where statements can be assignments, conditionals, and loops.

NOTE: The symbols "|" and "+" are part of the regular expressions, the others are part of the language.

It is required:

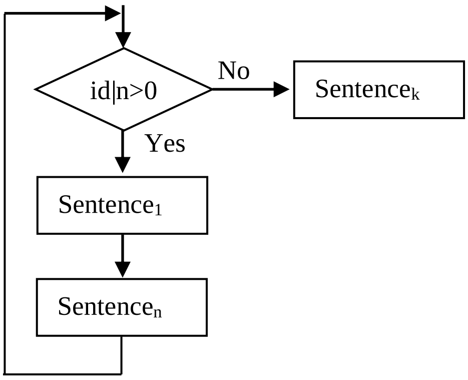
1. Define the grammar G that would generate valid programs of this programming language. Consider the assignment and conditional statements as terminal symbols of the grammar.

2. Describe the semantic routines of the grammar G that generate intermediate code in quartets with the following instructions, where *pos* are memory addresses, registers, or a number, and *reg*, *reg1* and *reg2* can be a record or a number. Write the semantic routines for the two interpretations that can be made about the execution flow of the loop:

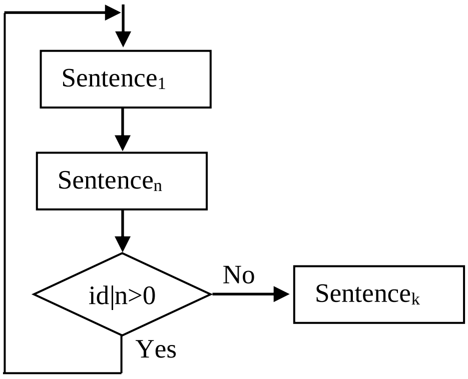
repeat (id | n) >>
$sentence_1$
…
$sentence_n$
<<
$sentence_k$

David Griol Barres, Antonio Berlanga de Jesús, Jesús García Herrero, Juan Manuel Alonso Weber

Mode A



Mode B



Statement	**Meaning**
(move, *pos_1*,, *pos_2*)	*pos_2* ← *pos_1*
(push, *pos_1*, ,)	includes the contents of *pos_1* into the stack
(pop, , , *pos_1*)	*pos_1* ← top of the stack
(label, , , *label*)	defines a label
(goto, , , *label*)	go to a label
(return, , , *reg*)	go to the address specified by *reg*
(if, *reg*, , *label*)	go to label *reg* es -1
(<, *reg*, , *label*)	go to label if the contents of *reg* is lower or equal to 0
(+, *reg_1*, *reg_2*, *reg*)	*reg* ← *reg_1* + *reg_2*
(-, *reg_1*, *reg_2*, *reg*)	*reg* ← *reg_1* - *reg_2*
(*, *reg_1*, *reg_2*, *reg*)	*reg* ← *reg_1* * *reg_2*
(/, *reg_1*, *reg_2*, *reg*)	*reg* ← *reg_1* / *reg_2*

SOLUTION:

A grammar to generate the language defined:

G = {assigment, condition, id, n, repeat, (,), <<, >>},{**S**, **S'**, **B**, **E**, **R**}, {**S**}
(1) **S**::= **E S'**
(2) **S'**::= **S**
(3) **S'**::= λ
(4) **E**::= assigment
(5) **E**::= condition
(6) **E**::= **B**
(7) **B**::= repeat (**R**
(8) **R**::= id) >> **S** <<
(9) **R**::= n) >> **S** <<

O::=Id
O.Value:= Id.Value;
O.Code:= "";

O::=Num
O.Value:= Id.Value;
O.Code:= "";

Z::=E
Z.Value:=E.Value;
Z.Code:=E.Code;

Z::=C
Z.Value:=C.Value;
Z.Code:=C.Code;

S::=λ
S.Code:=""

S_0::=CS_1
S_0.Code:=C.Code
S_1.Code

S_0::=CS_1
S_0.Code:=E.Code
S_1.Code

Use the stack to know which variables to assign the value of the expression, stack = -1 empty.

T::=λ
T.Code:=(push,-1,,)

T::=;V
T.Code:=V.Code

V::= Id T
V.Value:=newtemp;
V.Code:=(push,,,Id)

O'::=+
O'.Code=""
O'.Operation="+"

The Operation attribute is included to later know which operation to perform.

O'::=-
O'.Code=""
O'.Operation="-"

O'::=*
O'.Code=""
O'.Operation="*"

O'::=/
O'.Code=""
O'.Operation="/"

U::= λ
U.Code:=""

U::=O' E'
U.Value:=E'.Value;
U.Operation=O'.Operation
U.Code:=E'.Code

E'::=O U
E'.Value:=newtemp;
E'.Code:= O.Code
    if U.Code=""
    then (move, O.Value, , E'.Value)
    else U.Code
        Select case U.Operation
        case "+"
            (move, O.Value,,A)
            (move, U.Value,,B)
            (+, O.Value,,A)
        case "-"
            (move, O.Value,,A)
            (move, U.Value,,B)
            (-, O.Value,,A)
        case "*"
            (move, O.Value,,A)
            (move, U.Value,,B)
            (*, O.Value,,A)
        case "/"
            (move, O.Value,,A)
            (move, U.Value,,B)
            (/, O.Value,,A)
        end select
        (move, A,, E'.Value)

E::=E'->V
E.Start:=newlabel;
E.Stack_Empty:=newlabel;
E.Code:=(label,,,Start)
    (pop,,,A)
    (if,A,,Stack_Empty)
    (move,A,E'.Value,)
    (goto,,,Start)
    (label,,,Stack_Empty)

C::=¿(E')=>ZW
C.False:=newlabel;
C.Exit:=newlabel;
C.Code:=E'.Code
    (<,E'.Value,,C.False)
    Z.Code
    (goto,,,C.Exit)
    (label,,,C.False)
    W.Code
    (label,,,C.Exit)

W::=?
W.Code:=""

David Griol Barres, Antonio Berlanga de Jesús, Jesús García Herrero, Juan Manuel Alonso Weber

W::=/=>Z?
W.Code:= Z.Code