UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Especificación sintáctica formal de Q y del lenguaje nADA

Compiladores

Mireles Suárez, Alberto Manuel
Morales Sáez, David Guillermo
Quesada Díaz, Eduardo
Sánchez Medrano, María del Carmen

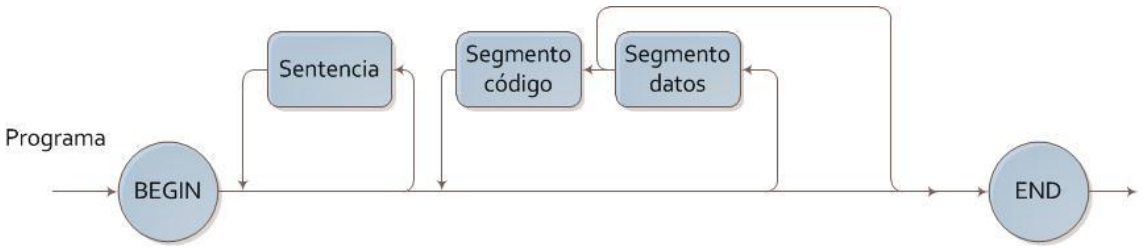
Índice

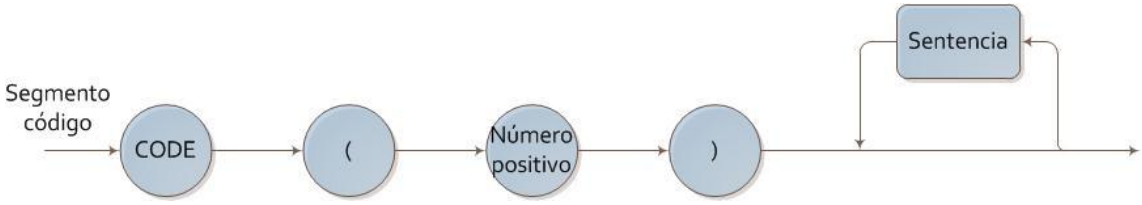
Introducción

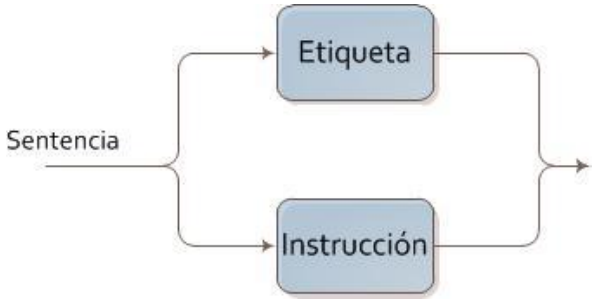
En este documento se presentan tanto los diagramas sintácticos reconocidos por la gramática Q, como la especificación del lenguaje escogido para realizar las prácticas durante el curso, llamado nADA. Seguidamente pasaremos a ver estos dos apartados por separado.

Diagramas sintácticos de Q

A continuación se detallan los diagramas sintácticos que son reconocidos por la máquina Q, identificando como círculos los símbolos terminales y como rectángulos los subdiagramas.

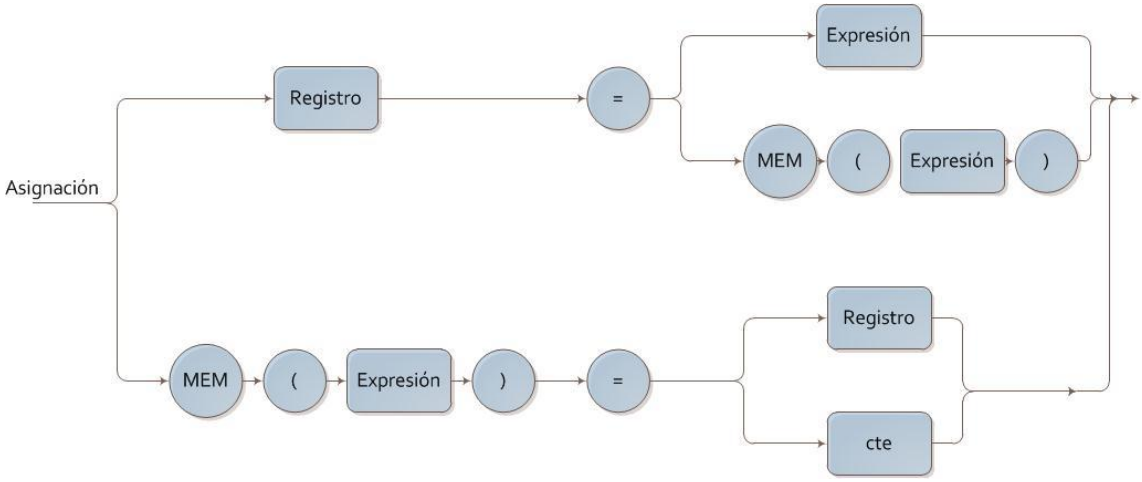


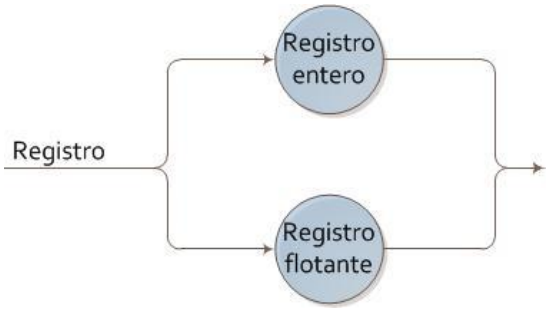


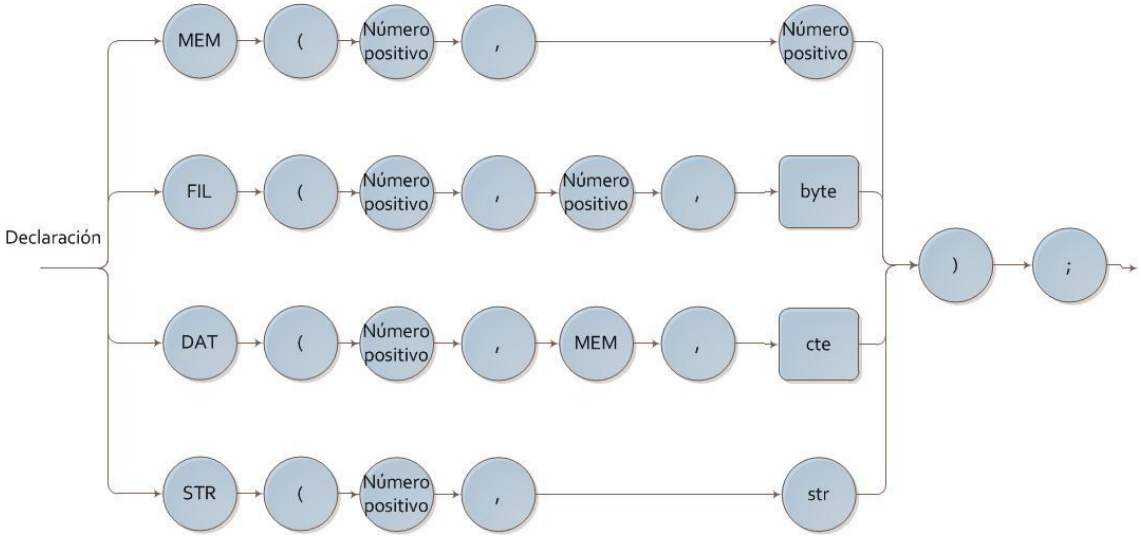

Sentencia
Etiqueta
Instrucción

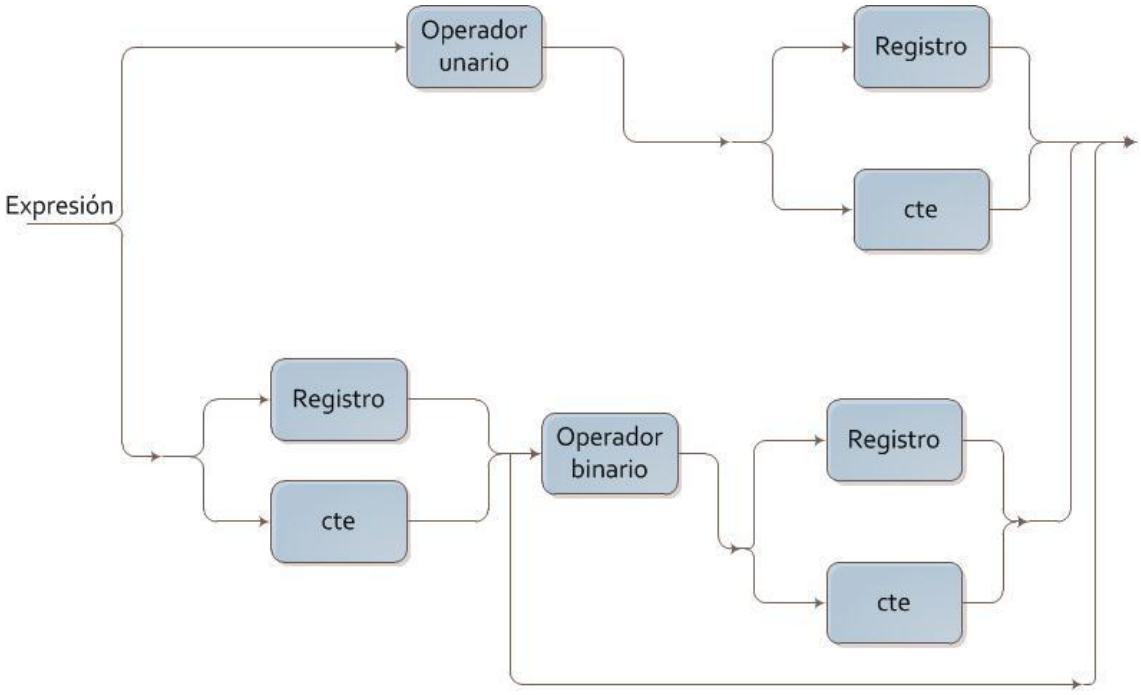

Etiqueta
L
Número positivo
:

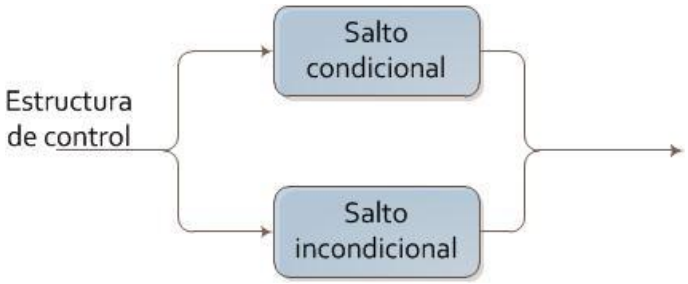

Instrucción
Asignación
Estructura de control
;

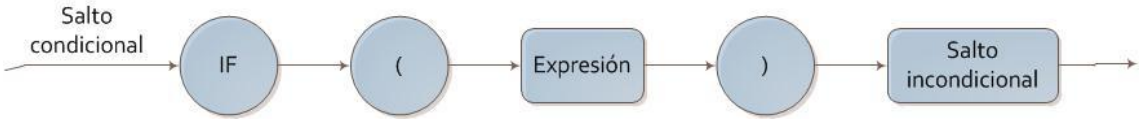

Asignación
Registro
=
Expresión
MEM
(
Expresión
)
MEM
(
Expresión
)
=
Registro
cte

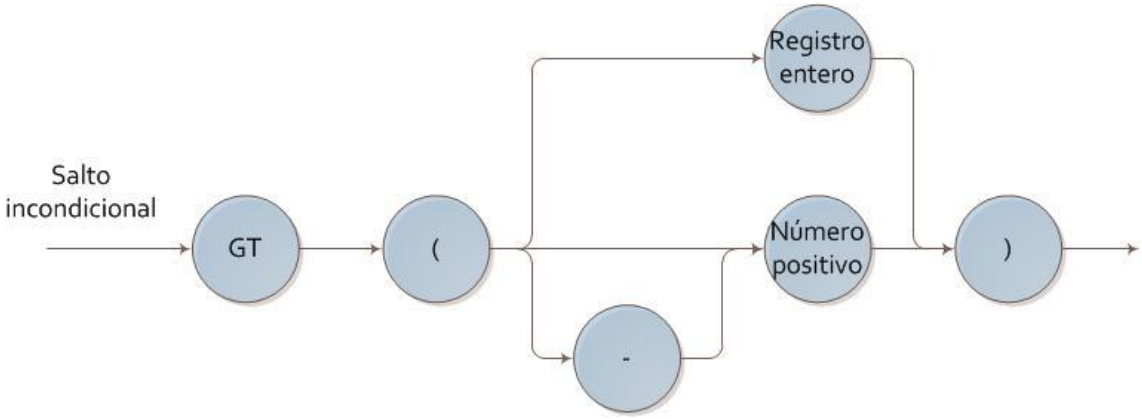

Registro
Registro entero
Registro flotante

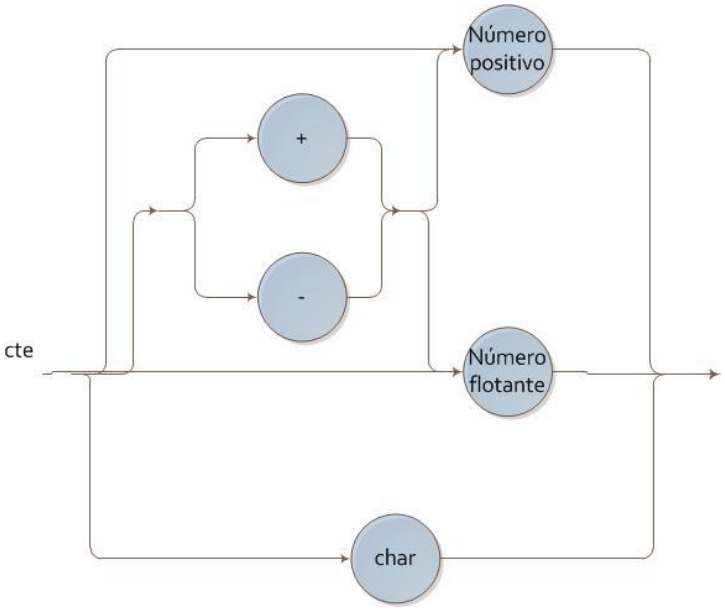

Declaración
MEM
(
Número positivo
,
Número positivo
FIL
(
Número positivo
,
Número positivo
,
byte
DAT
(
Número positivo
,
MEM
,
cte
STR
(
Número positivo
,
str
)
;

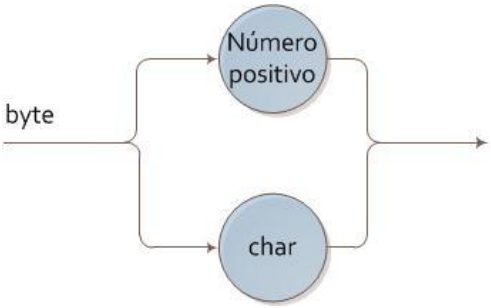

Expresión
Operador unario
Registro
cte
Registro
cte
Operador binario
Registro
cte

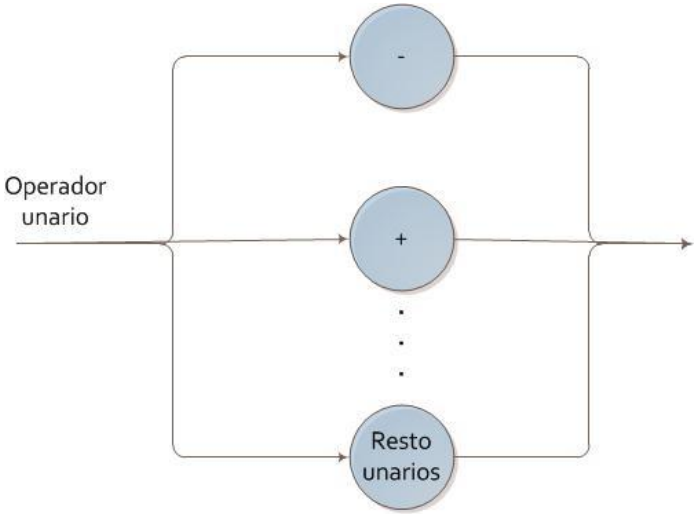

Estructura
de control
Salto
condicional
Salto
incondicional

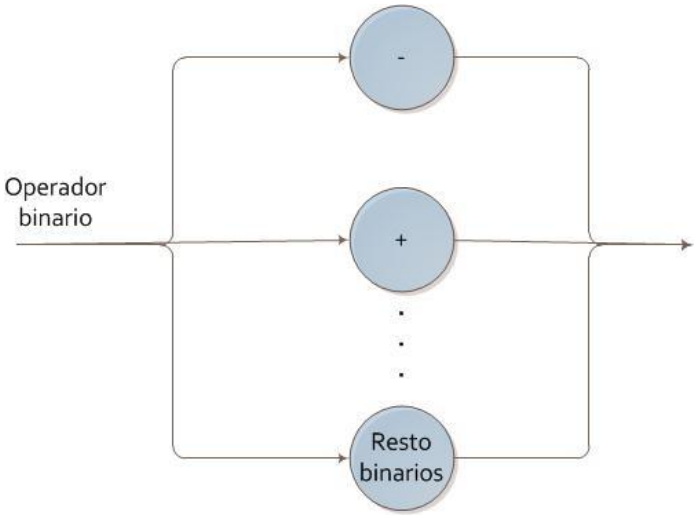

Salto
condicional
IF
(
Expresión
)
Salto
incondicional


Salto
incondicional
GT
(
Registro
entero
Número
positivo
-
)


cte
Número
positivo
+
-
Número
flotante
char


byte
Número positivo
char


Operador unario
-
+
Resto unarios


Operador binario
-
+
Resto binarios

Especificación del lenguaje nADA

Basada en la especificación indicada en: http://cuiwww.unige.ch/isi/bnf/Ada95/BNFindex.html

```
procedure_call_statement ::=  name  actual_parameter_part | ";"
subprogram_declaration ::= subprogram_specification ";"
subprogram_specification ::=
      "procedure" defining_program_unit_name subprog_spec2
subprog_spec2 ::= formal_part | ε


with_clause ::= "with" name with2 ";"
with2 ::= "," name with2 | ε
use_clause ::= "use" name with2 ";" | "use" "type" name with2 ";"
mode ::= "in" | "in" "out" |  "out" | ε


assignment_statement ::= name ":=" name ";"


case_statement ::=
      "case" expression "is" case_statement_alternative
      alternative2 "end" "case" ";"
case_statement_alternative ::=
      "when" discrete_choice_list "=>" sequence_of_statements
alternative2 ::= case_statement_alternative alternative2 | ε


loop_statement ::= loop2 "loop" sequence_of_statements
      "end" "loop" loop2 ";"
loop2 ::= identifier | string_literal ":" | ε
if_statement ::=
      "if" condition "then" sequence_of_statements
      if_stat2 "end" "if" ";"
if_stat2 ::= "else" sequence_of_statements | ε


enumeration_type_definition ::=
      "(" enumeration_literal_specification enum_def2 ")"
enumeration_literal_specification ::= identifier | "'"
graphic_character "'"
enum_def2 ::= "," enumeration_literal_specification enum_def2 | ε


string_literal ::= "quotation mark" string_literal2 "quotation mark"
string_literal2 ::= string_element string_literal2 | ε
string_element ::= "pair of quotation mark" | graphic_character
```

```
constrained_array_definition ::=
      "array" "(" range array_def2 ")" "of" component_definition
array_def2 ::= "," range array_def2 | ε
range ::= simple_expression ".." simple_expression


simple_expression ::= simple2 simple3
simple2 ::= "+"  term | "-"  term |  term
simple3 ::= "+" term simple3 | "-" term simple3 | "&" term simple3 | ε


term ::= factor term2
term2 ::=  "*" factor term2 | "/" factor term2 | "mod" factor term2 |
"rem" factor term2 | ε


factor ::= factor2 | "abs" numeric_literal  | "not" numeric_literal
factor2 ::= numeric_literal | numeric_literal "**" numeric_literal


actual_parameter_part ::=
      "(" parameter_association para_part2 ")"
para_part2 ::= "," parameter_association para_part2 | ε
parameter_association ::= selector2 expression | selector2 name


selector2 ::= selector_name "=>" | ε
selector_name ::= identifier | character_literal | string_literal

expression ::=
      relation expr_rel2 | relation expr_rel3 | relation expr_rel4
expr_rel2 ::= "and" relation expr_rel2 | ε
expr_rel3 ::= "or" relation  expr_rel3 | ε
expr_rel4 ::= "xor" relation expr_rel4
| ε


relation ::= simple_expression relation2 | relation3
relation2 ::= "=" simple_expression |
               "/=" simple_expression|
               "<" simple_expression |
               "<=" simple_expression|
               ">" simple_expression |
               ">="simple_expression | ε
relation3 ::= simple_expression relation4 "in" range
relation4 ::= "not" | ε
```

```
discrete_choice_list ::= discrete_choice choice2
choice2 ::= "|" discrete_choice choice2 | ε
discrete_choice ::= expression | range| "others"


sequence_of_statements ::= statement seq_statement2
seq_statement2 ::= statement seq_statement2 | ε
statement ::= statement2 simple_statement | statement2
compound_statement
statement2 ::= label statement2 | ε
compound_statement ::=
      if_statement
      | case_statement
      | loop_statement
simple_statement ::=
      null_statement
      | assignment_statement
      | exit_statement
      | procedure_call_statement
      | return_statement
      | abort_statement


express ::= expression | ε
cond ::= "when" condition  | ε


exit_statement ::= "exit" cond ";"
return_statement ::= "return" express ";"


defining_program_unit_name ::= name "."  identifier | identifier
formal_part ::=
      "(" parameter_specification ";" formal2 ")"
formal2 ::= ";" parameter_specification formal2 | ε

parameter_specification ::=
      defining_identifier_list ":" mode subtype_mark  par_spec2
    | defining_identifier_list ":"  access_definition par_spec2
par_spec2 ::= ":=" expression | ε


defining_identifier_list ::= identifier def_ident2
def_ident2 ::= "," identifier def_ident2 | ε
defining_designator ::= name | string_literal
```