Vlad Pop, group 936

https://github.com/Pop-Vlad/FLCD

Contents

Language description:

Lexic:

Alphabet:

Uppercase and lowercase characters of the english alphabet (A-Z and a-z)

Decimal digits (0-9)

Special characters: ( ) { } [ ] ; = : - + * / % " space \n \t

Special symbols:

operators:

:= + - * / % == != < > <= >=

separators:

space { } ; ( ) \n \t [ ]

reserved words:

int char if while read write

Identifiers:

A sequence of letters and digits that starts with a letter.

IDENTIFIER = LETTER ({LETTER | DIGIT})

LETTER = "A" | "B" | ... | "Z" | "a" | "b" | ... | "z"

DIGIT = "0" | "1" | ... | "9"

Constants:

integer: A sequence of digits that does not start with the digit "0" and may have the symbol "+" or "-" before it.

INT = "0" | ["+" | "-"] NON_ZERO_DIGIT {DIGIT}

DIGIT = "0" | "1" | ... "9"

NON_ZERO_DIGIT = "1" | "2" | ... "9"

character:

CHARACTER = "LETTER" | "DIGIT"

LETTER = "A" | "B" | ... | "Z" | "a" | "b" | ... | "z"

DIGIT = "0" | "1" | ... | "9"

Array elements:

ARRAY_ELEM = IDENTIFIER "[" UNSIGNED_NUMBER "]"

UNSIGNED_NUMBER = "0" | NON_ZERO_DIGIT {DIGIT}

Token.in:

:= + - * / % < > <= >= == !=

space { } ; ( ) \n \t [ ]

int char if while read write

Syntax:

program ::= stmt_list

stmt_list ::= stmt | stmt ";" | stmt ";" stmt_list

stmt ::= simple_stmt | struct_stmt

simple_stmt ::= assign_stmt | decl_stmt | io_stmt

assign_stmt ::= identifier ":=" expression | array_elem ":=" expression

expression ::= expression "+" term| expression "-" term | term

term ::= term "*" factor | term "/" factor | factor

factor ::= arithmetic_operand | "(" expression ")"

arithmetic_operand ::= identifier | constant | array_elem

decl_stmt ::= type identifier

type ::= simple_type | comp_type

simple_type ::= "int" | "char"

comp_type ::= simple_type "[]"

io_stmt ::= "read(" read_operand ")" | "write(" write_operand ")"

read_operand ::= identifier | array_elem

write_operand ::= identifier | array_elem | constant

struct_stmt ::= "{" stmt_list "}" | if_stmt | while_stmt

if_stmt ::= "if" condition stmt | "if" condition stmt "else" stmt

while_stmt ::= "while" condition "{" stmt "}"

condition ::= expression relation expression

relation ::= "==" | "<" | ">" | "<=" | ">=" | "!="

Symbol table:

The symbol table uses a hash table. Each position of the table is a reference to an element of type Node. Collisions resolution: Store tokens that hash to the same position in a double linked list of nodes.

Symbol table structure:

- Table: Node[] – The table in which the list of nodes are stored. Each node contains a token.
- Size: int – The size of the hash table

Node structure:

- Next: Node – Reference to the next node in the linked list
- Previous: Node – Reference to the previous node int the linked list
- Token: String – The token

Dispersion: The hash table uses the default java hash() function for the tokens.

Program internal form:

The PIF uses a list of pairs. Each pair has a string representing the token and an integer representing the position. If the token is an identifier "0" is used for the token. If it is a constant "1" is used for the token. If the token is not an identifier or constant -1 is used for the position.

PIF structure:

- Pairs: Pair[]

Pair:

- First: string
- Second: int

Scanner:

Description:

The source code is stored in a string called "program" which is analyzed character by character.

The function detect() detects 1 token:

- The scanner starts from a position and builds a token by adding 1 character at a time.
- The scanner looks ahead if the token it builds can become a valid token. This is done by checking if an element from token.in starts with the current string or if the current string is a valid identifier or constant.

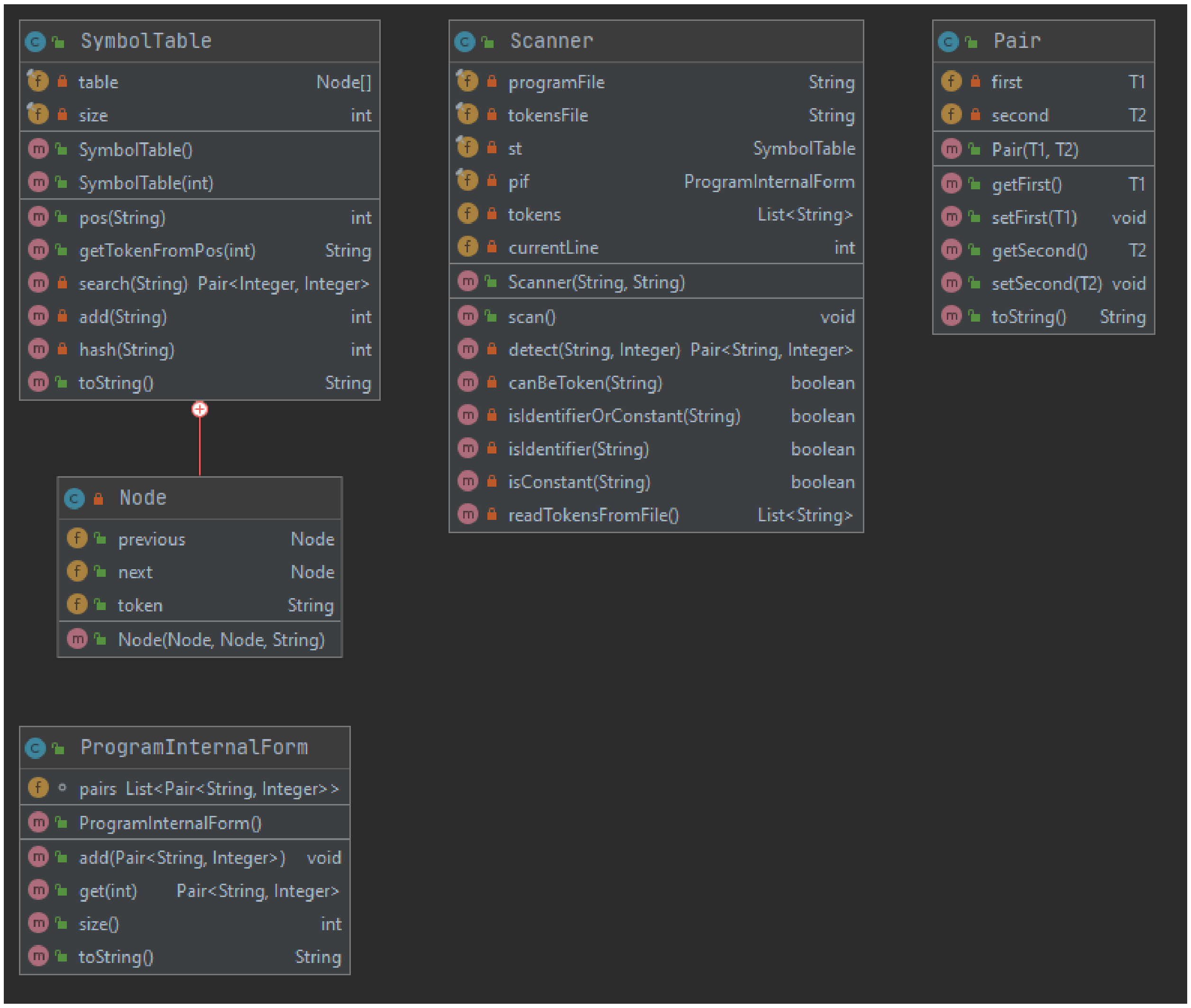
- If the token is "+" and "-", the scanner checks if it is an operator(e.g. 1 + 2) or is part of an integer constant (e.g. x := +3). This is done by checking if the last token in the PIF is ":=". If so, the token is part of a constant.

After detect() finds a token, the scanner tries to classify it as:

- Reserved word, operator or separator – The scanner searches for the token in a list of strings given by the file token.in
- Identifier or constant: - The scanner checks if the token matches a given regex.

The classified value is added to the PIF.

Scanner, PIF and ST class diagram:



```
SymbolTable
table                              Node[]
size                               int
SymbolTable()
SymbolTable(int)
pos(String)                        int
getTokenFromPos(int)               String
search(String)  Pair<Integer, Integer>
add(String)                        int
hash(String)                       int
toString()                         String
```

```
Node
previous                           Node
next                               Node
token                              String
Node(Node, Node, String)
```

```
ProgramInternalForm
pairs  List<Pair<String, Integer>>
ProgramInternalForm()
add(Pair<String, Integer>)         void
get(int)        Pair<String, Integer>
size()                             int
toString()                         String
```

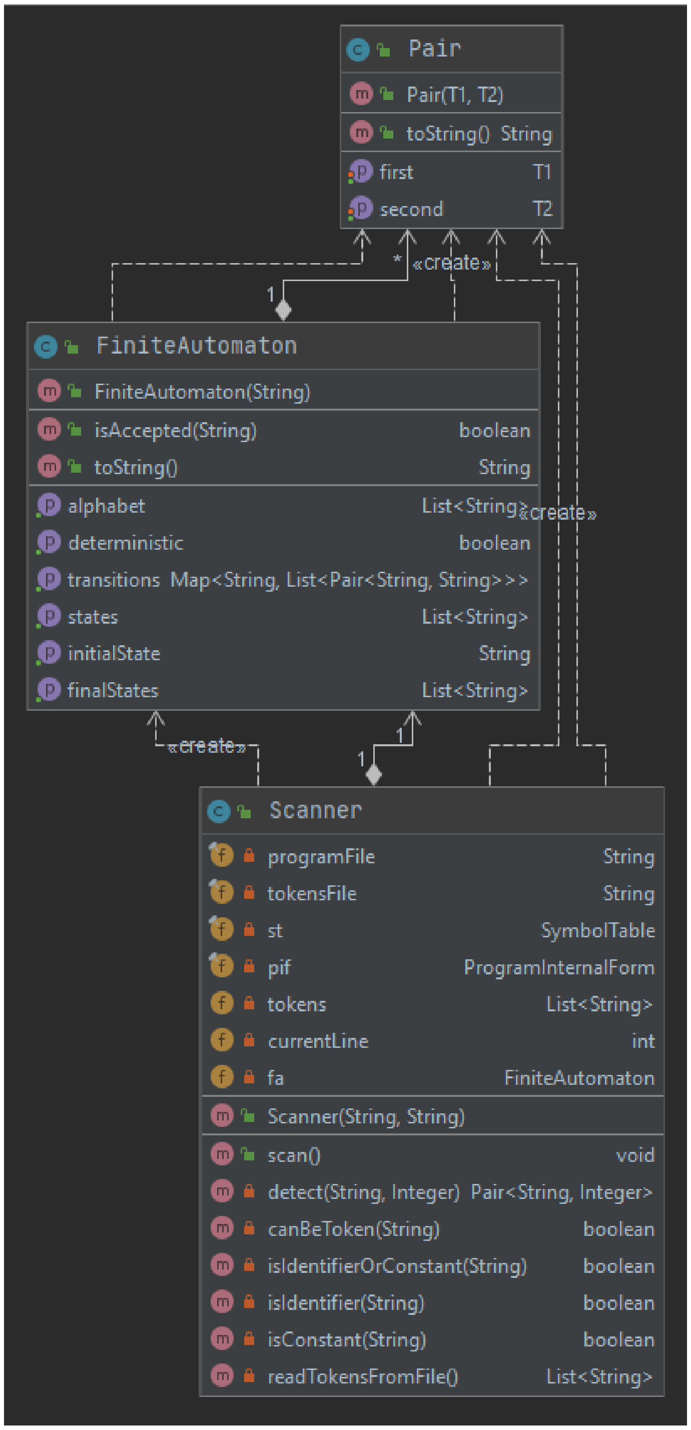
```
Scanner
programFile                        String
tokensFile                         String
st                                 SymbolTable
pif                                ProgramInternalForm
tokens                             List<String>
currentLine                        int
Scanner(String, String)
scan()                             void
detect(String, Integer)  Pair<String, Integer>
canBeToken(String)                 boolean
isIdentifierOrConstant(String)     boolean
isIdentifier(String)               boolean
isConstant(String)                 boolean
readTokensFromFile()               List<String>
```

```
Pair
first              T1
second             T2
Pair(T1, T2)
getFirst()         T1
setFirst(T1)       void
getSecond()        T2
setSecond(T2)      void
toString()         String
```

Vlad Pop, group 936

Finite Automaton:

Finite automaton Class diagram:



Vlad Pop, group 936

Format of FA.in:

FA = STATES_LIST "\n" ALPHABET "\n" STATE "\n" STATES_LIST "\n" TRANSITIONS_LIST

STATES_LIST = STATE | STATE " " STATES_LIST

STATE = WORD

WORD = SYMBOL | SYMBOL WORD

ALPHABET = SYMBOL | SYMBOL " " ALPHABET

TRANSITIONS_LIST = TRANSITION | TRANSITION "\n" TRANSITIONS_LIST

TRANSITION = STATE " " LIST_OF_TRANSIT " " STATE

LIST_OF_TRANSIT = SYMBOL | SYMBOL " " LIST_OF_TRANSIT

SYMBOL = "a" | "b" | ... | "z" | "A" | "B" | ... | "Z" | "0" | "1" | ... | "9"

Finite automaton strucure:

FA has the following components:

- states: list of string
- alphabet: list of string
- transtions: map with source states (String) as keys and list of (symbol, destination state) pairs as values (symbol, destinationa are both strings).
- initialState: string
- finalStates: list of string
- isDeterministic(): Checks if the given FA from FA.in is deterministic. This is done by iterating through all the states and checking if there are any outgoing transitions with the same symbol. If there are such transitions, the FA is not deterministic.
- isAccepted(sequence): Checks if a given sequene is accepted by the FA. The sequence is analyzed character by character. The algorithm starts from the initial state and uses one sybol to perform a transition to another state. The algorithm finishes when a sybol can not be used for a transition or the sequence is consumed. The sequence is accepted if it is entirely consumed and the state reached is a final state.

Example:

- FA.in:

```
p q r
0 1
p
r
p 1 p
p 0 q
q 1 p
q 0 r
r 0 1 r
```

- Sequence: “101”

The FA is deterministic. (Tere are no transitons that have the same source state and the same symbol).

Current state = p, sequence = “101”

Iteration 1:

Current state = p, sequence = “01”

Iteration 2:

Current state = q, sequence = “1”

Iteration 2:

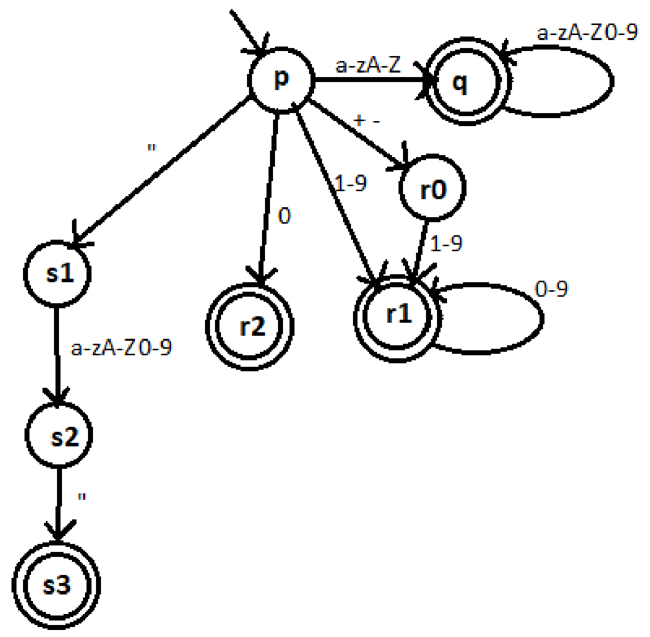
Current state = p, sequence = “”

P is not a final state, thus “101” is not accepted by the FA.

FA for identifiers or constants:

FA diagram:

Vlad Pop, group 936



FA input for identifiers or constants

```
p q r0 r1 r2 s1 s2 s3
a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 " + -
p
q r1 r2 r3 s3
p a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L
M N O P Q R S T U V W X Y Z q
q a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L
M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 q
p + - r0
r0 1 2 3 4 5 6 7 8 9 r1
p 1 2 3 4 5 6 7 8 9 r1
r1 0 1 2 3 4 5 6 7 8 9 r1
p 0 r2
p " s1
s1 a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K
L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 s2
s2 " s3
```