
BUMPY

SER502-Spring2019-Team11

Team 11 – bumpy

Bharat Goel

Madhukar Raj

Palak Chugh

Yuti Desai

Feature Of The Language

- Bumpy Language is developed in Prolog Completely
- We are using Top- down parsing technique
- Data structures used by the parser and interpreter: List
- Interpreter: Our interpreter is based on Reduction machine.
- It is an Imperative language and Statically typed
- It provides data types such as Integer and Boolean
- Looping construct – while loop
- Decision control statement – if-else statement
- inspired by C language

Operators and Constructs For BUMPY

- Operators: +,-,*,/,%,<,>,<=,>=, ~=, :=:, = ,and, or,
- Arithmetic Operator: +,-,*,/,%

  Addition : +
  Subtraction: -
  Multiplication: *
  Division: /
  Mod: %
- Comparison Operator: <,>,<=,>=, ~=, :=:

  Less than: <
  Greater than: >
  Less than equal to: <=
  Greater than equal to: >=
  Not equals to: ~=
  Equals to: :=:

Operators and Constructs For BUMPY

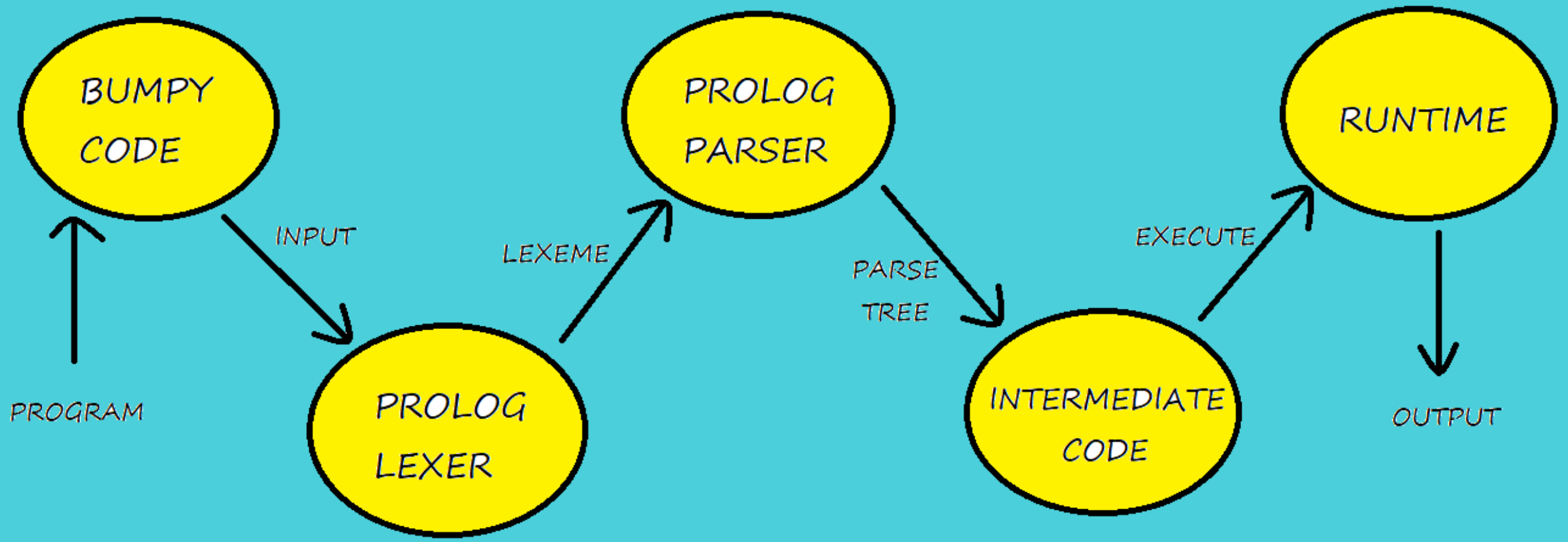
- Assignment Operator: =
- Boolean Operator : and, or ,not
- Primitive types: bool, var

  Bool : takes Boolean value

  Var: takes integer value
- Decision Constructs: incase do otherwise endcase

  Incase (condition) do (process) otherwise (process) endcase
- Iterative Constructs: when repeat endrepeat

  When (condition) repeat (process) endrepeat


BUMPY CODE
INPUT
PROLOG LEXER
LEXEME
PROLOG PARSER
PARSE TREE
INTERMEDIATE CODE
EXECUTE
RUNTIME
OUTPUT
PROGRAM

Design Flow

Lexical Analyzer and Parser

- Lexical Analyzer and Parser written in Prolog Definite Clause Grammar.
- Once the program is fed to the Lexer, it generates a list of lexemes for program by identifying keywords, eliminating spaces, tabs and new lines.
- After getting the tokens, tokens are passed through the parser which is a prolog DCG code with the grammar defined for the language using prolog predicates and it generates the parse tree for the given code.
- It Parses in a top-down fashion.

Intermediate Code

- Parse tree is itself the intermediate code as we are using Prolog for creating language.
- It gives Intermediate code as output to a separate file.
- Intermediate Code is generated using DCG as well.
- Used interpreter to evaluate the intermediate code.

Runtime/Interpreter

- Programmed in Prolog.
- The Interpreter is implemented by traversing node by node through the generated parse tree. Each node denotes an evaluation step that needs to be performed
- The environment at each node is looked up or updated via a list data structure in prolog.
- Used reduction rules to evaluate the code.

Language Grammar

- Parser → Program
- Program → Comment Block | Block
- Comment → @ Words @
- Block → start Declaration Process stop
- Words → Identifier Words | Numb Words| Identifier| Numb.
- Declaration → Datatype Identifier ; Declaration | Datatype Identifier ;
- Process → Assignvalue ; Process | Control Process | Iterate Process | Print Process |ReadValue Process | Assignvalue ; | Control | Iterate | Print | ReadValue

Language Grammar (Continued..)

- Datatype → var | bool
- Assignvalue → Identifier = Expression| Identifier is Boolexp
- Control → incase Condition do Process otherwise Process endcase
- Iterate → when Condition repeat Process endrepeat
- Print → show Expression ; | show * Value * ;
- ReadValue → input Identifier ;
- Condition → Boolexp and Boolexp| Boolexp or Boolexp | ~ Boolexp| Boolexp

Language Grammar (Continued..)

- Boolexp → Expression :=: Expression |Expression ~= Expression| Expression <= Expression| Expression >= Expression|Expression < Expression| Expression > Expression |Expression :=: Boolexp|Expression ~= Boolexp| yes | no
- Expression → Term + Expression |Term – Expression |Term
- Term → Identifier * Term| Numb * Term|Numbneg * Term | Identifier / Term|Numb /Term|Numbneg /Term | Identifier mod Term|Numb mod Term|Numbneg mod Term|Identifier|Numb|Numbneg
- Identifier → _[^a-z]alphanumeric | [^a-z]alphanumeric
- Numb → number
- Numbneg → - Numb

Demonstration: Executing Prolog Files