Simple Language

Team-9

Sai Kumar Chunchu

Vinesh Reddy Naga

Venkata Vamsikrishna Iytha

Yogaleena Mandalapu

Overview

- About Simple Language
- Grammar
- Lexer
- Parser/Syntax Tree
- Semantics
- Sample Outputs

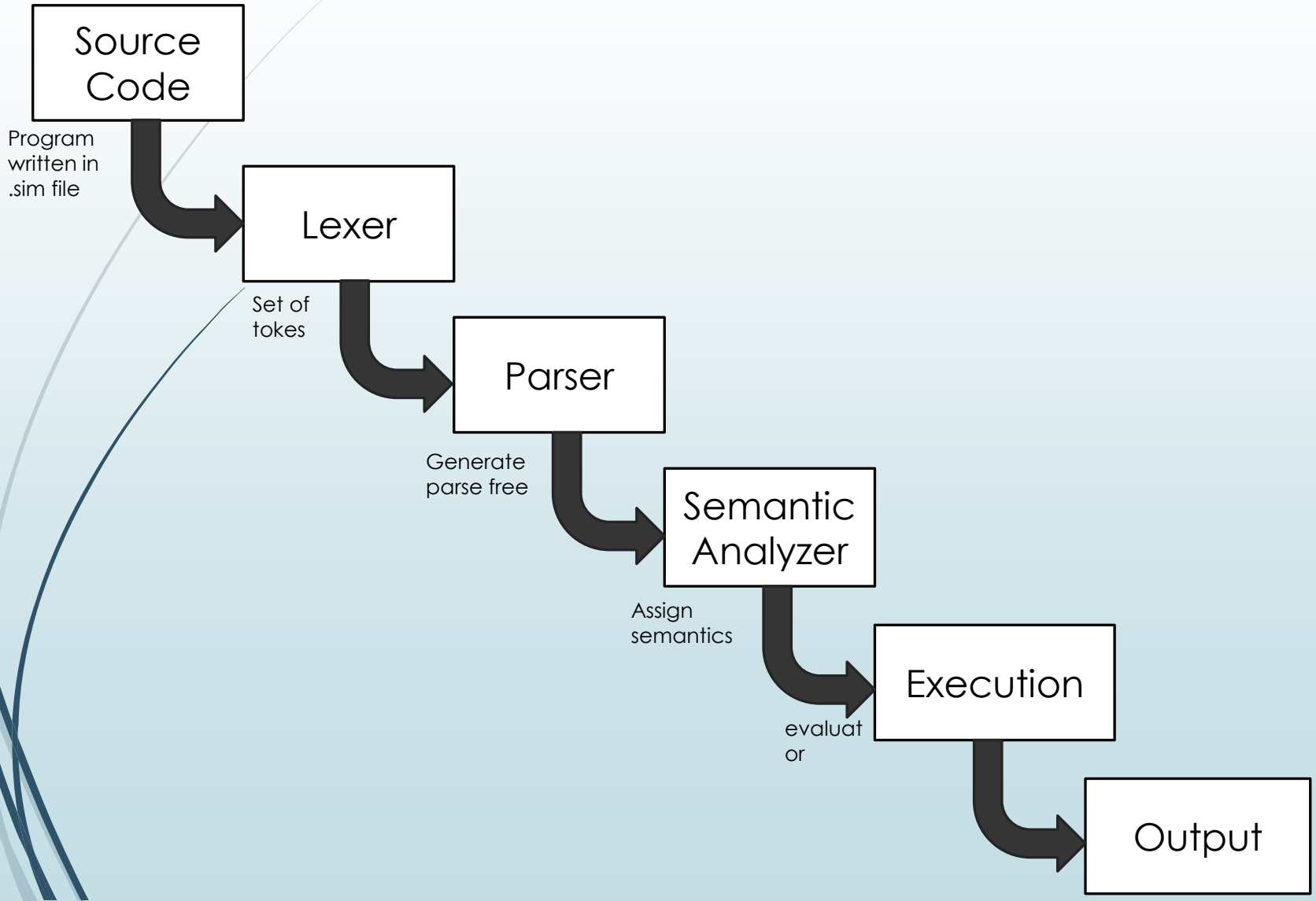
About Simple Language

- Our language is inspired from existing languages such as C++ and Python, so that the syntax is intuitive.
- Its super easy to code.
- An Interpreted language.
- Developed entirely in Prolog.

Features of Simple Language

- Data Types Supported: Integer, Boolean and String
- Boolean Operations included are AND, OR and NOT
- Decision Control Statements:- IF-Then-Else
- Looping Construct: For loop, For in Range loop and While loop.
- Arithmetic Operations: +,-,*,/
- Ternary Operator
- Comparison Operators: >, >=, <, <=, ==, !=
- String Concatenation
- Supports Comments for Code
- Takes care of redeclarations.
- Supports type checking.

Flow


Source Code
Program written in .sim file
Lexer
Set of tokes
Parser
Generate parse free
Semantic Analyzer
Assign semantics
Execution
evaluat or
Output

Tools and Platform

- SWI-Prolog version 8.2.4 for x64-win64
- SWI-Prolog pack tokenize
- MinGW
- Windows Batchfile
- Windows

Grammar

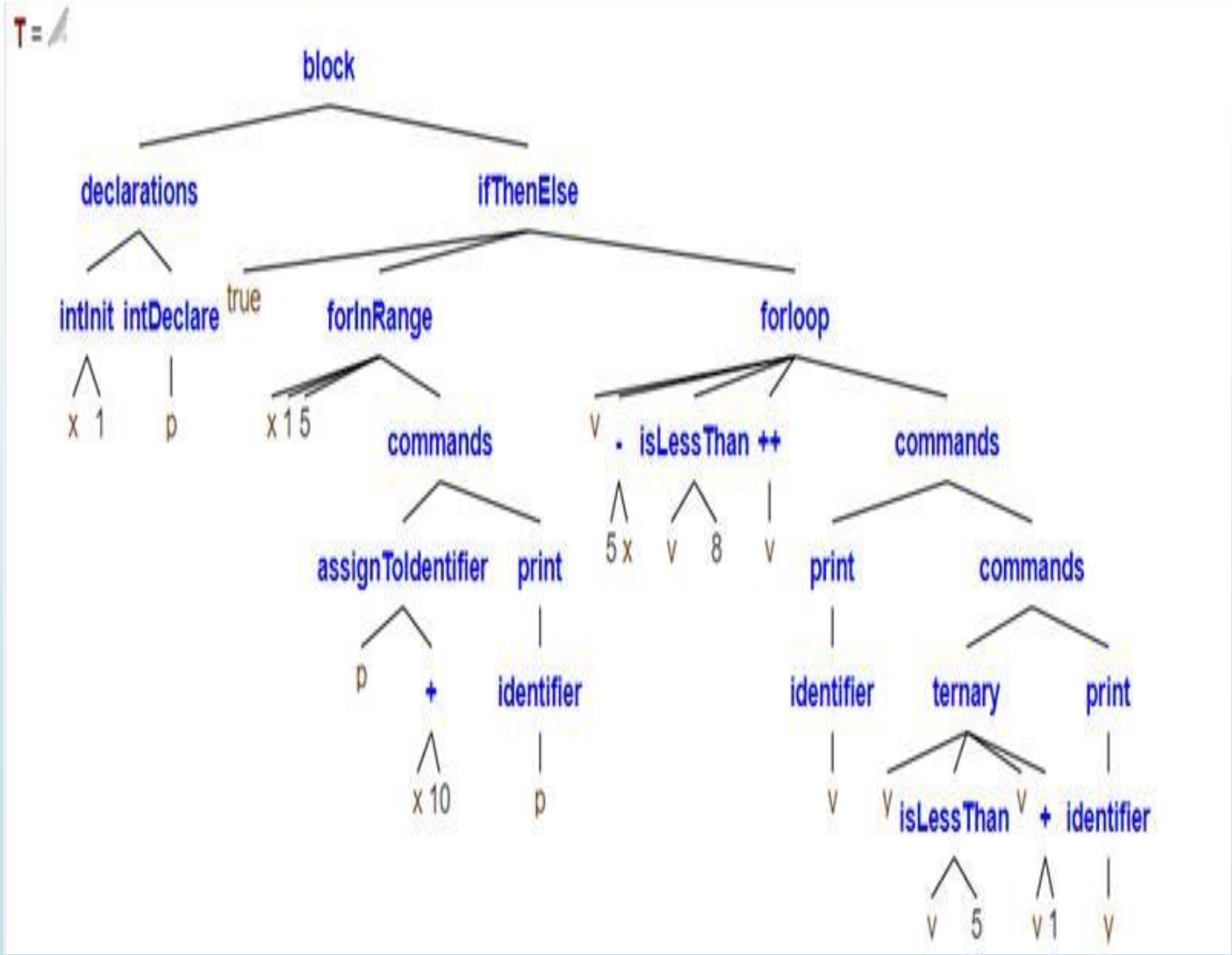
- P ::= begin K end | SC begin K end
- K ::= D;C;| C;
- D::= D; D | int I = N | string I = S | int I | string I | bool I | bool I = B
- C::= C; C | if (B) then {C} else {C}
  | for (int I = E; B; UO) {C}
  | for I in range(E,E) {C}
  | while (B) {C}
  | I = EXP
  | I = B ? E : E
  | D
  | print T
- B ::= true | false | E == E | E != E | E < E | E <= E | E > E | E >= E | BO | not B
- E ::= E + E | E - E | E * E | E / E | (E) | I | N | S
- UO ::= I++ | I--
- BO ::= B and B | B or B | (B)
- SC ::= /* S */
- N ::= Number
- S ::= “ String ”
- T::= I | N | S | B | S + S| I + I

Lexer and Parser

- Simple uses tokenize pack from SWI prolog to tokenize the code and chain it to the Parser.
- Parser receives a list of tokens and uses DCG notation to generate an abstract syntax tree out of the source code.
- This syntax tree has all the information needed for executing the code.
- At this stage we eliminate syntax errors from the code.

Parse Tree

```
begin
int x = 1 ;
int p;
if (true)then{
for x in range ( 1,5){
p = x + 10 ;
print p
}
}else{
for ( int v = 5-x ;v < 8 ; v++) {
print v ;
y = v < 5 ? v :v + 1;
print y
}
}
end
```

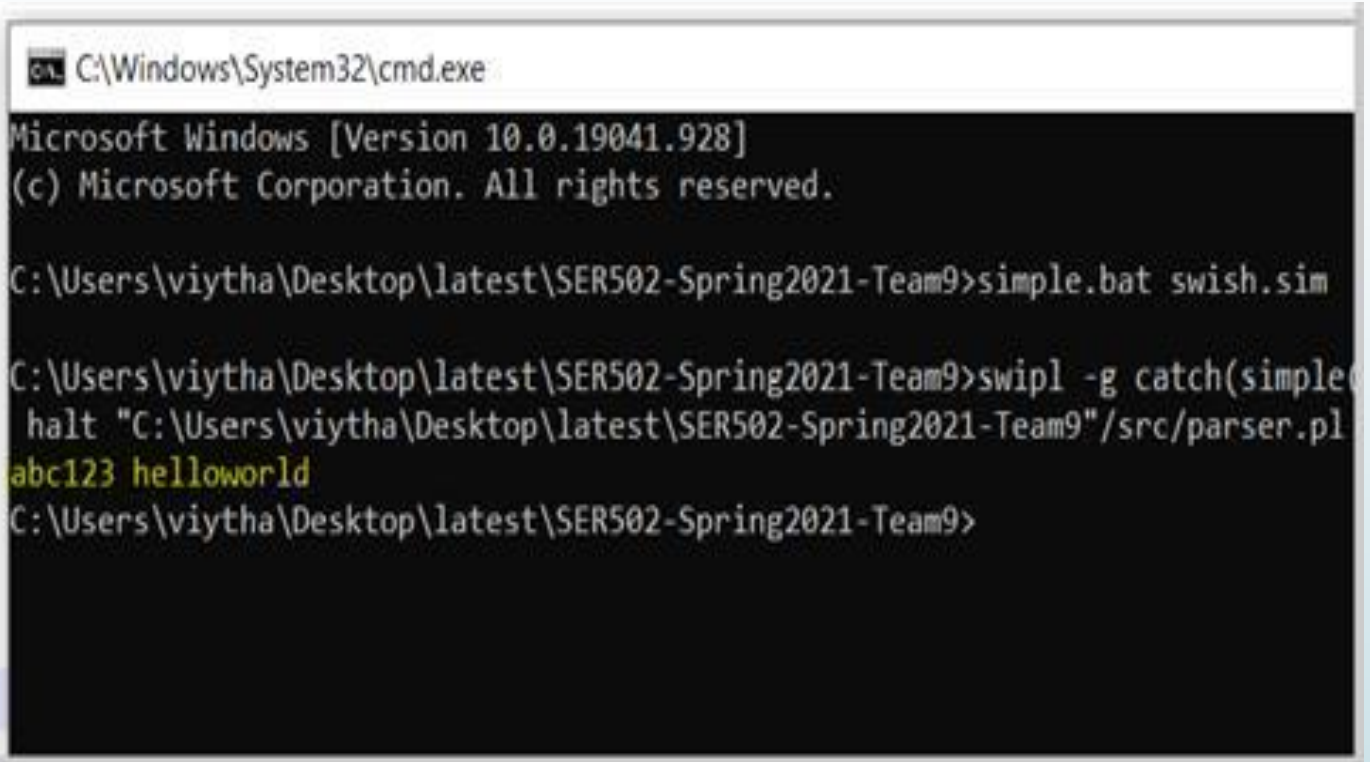


Semantics and execution

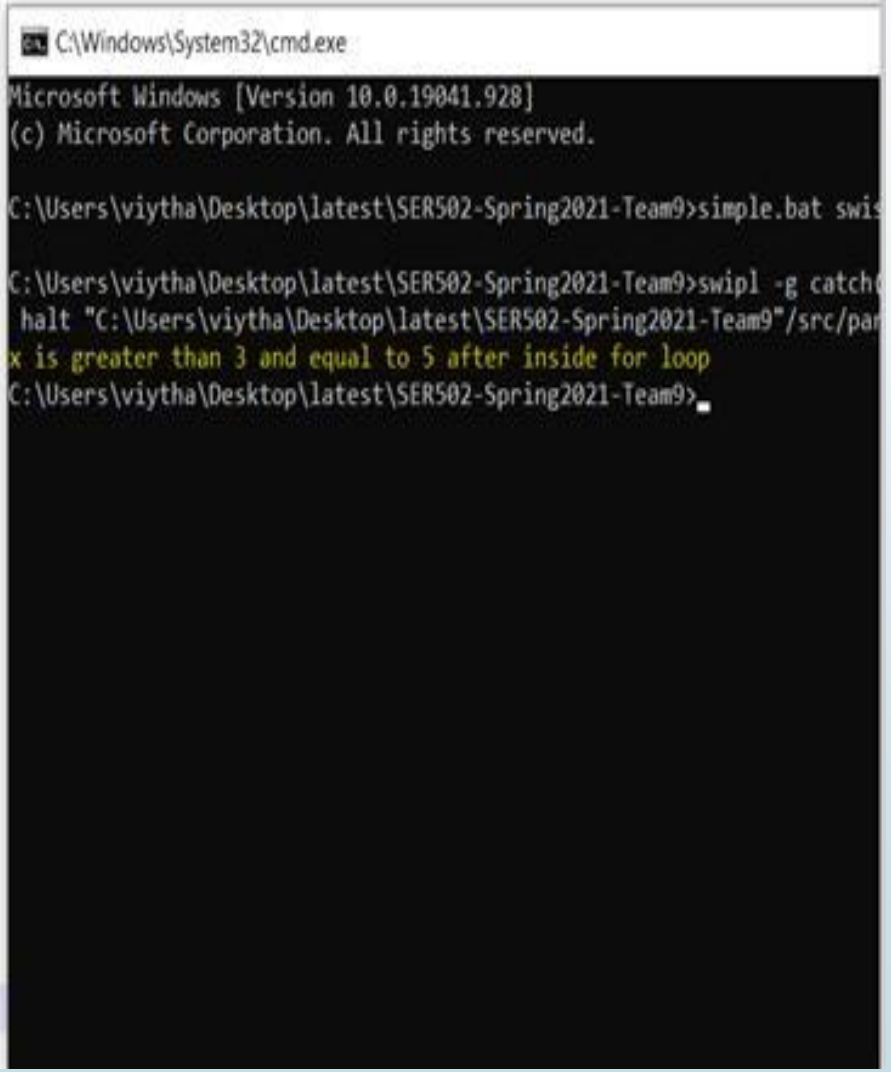
- We use predicates in prolog to assign semantics to the tree nodes generated in previous step.
- We maintain an Environment list to persist the state of variables while executing the program.
- Execution of the program is simply executing the top predicate i.e eval_program.

Output Screenshots

```
/* string concatenation  */

begin

string s1= "abc";
string s2 = "123";

print s1 + s2;
print " hello" + "world"
end
```



```
/* nested if-then-else loop */

begin
int x=5;
string s="x is greater than 3 and equal to 5";
string s1="x is equal to 5 and less than or equal to 3";
string s2 = " after inside for loop ";
string s3 = "not equal to 5";


if(x==5)then {
    if(x>3)then {
        print s
    }else{
        print s1
    };
    print s2
}else
    {
        print s3
    }
end
```



Thank you