SER 502
Spring 2024
Project Team 1

SARS

Programming Language

Arizona State University

Contents

- **Introduction:** Overview of the project and team members
- **Tools Utilized:** Explanation of the tools used in development
- **Language Design:** Key aspects of the SARS language design
- **Grammar Overview:** Introduction to the grammar structure
- **Lannguage Features:** Distinctive elements of the SARS language
- **Execution Workflow:** Step-by-step guide for executing SARS programs
- **Demonstrations:** Showcase of execution artifacts and results

Project Team Members (SARS)

- **S**umeet Suryawanshi
- **A**kash Rana
- **R**ohan Mathur
- **S**adhanand Srinivasan

Language Design

SARS comes from the initials of all four project team members(Sumeet, Akash, Rohan, Sadhanand). The token is generated using python file.

Parse Tree generation is done using Prolog.

Language Design

- Input files have .sars extension.

- Lexer.py is used to generate a list of tokens which are parsed using Prolog to give the final output.

Tools Utilized

Tools employed in the project:

- **SWI-Prolog 9.2.4-1**: Compilation and Parsing
- **Python3.9^** : Token Generation
- **VS Code**: Running the prolog code

Source Code Structure

- SARS programs start with “begin” and end with “end” for clarity.
- Input files use the .sars extension for easy identification.
- Python-based Lexer generates tokens from SARS code.
- Prolog-based Parser constructs syntactically correct parse trees.
- Structured approach ensures clarity and ease of maintenance.

Grammar Overview

- **SARS language grammar is designed for clarity and simplicity.**
- **Grammar rules are created using Definite Clause Grammar (DCG).**
- **Clear rules define syntax and structure of SARS programs.**
- **Well-defined grammar facilitates accurate parsing and interpretation.**
- **DCG ensures consistency and readability in SARS code.**

SARS Features (Part-1)

- **Data types:**
  - **Integer:** 1,2,3…
  - **Boolean:** True/ False
  - **String:** “Hello”

- **Arithmetic operations include:**
  - **Addition ‘+’**
  - **Subtraction ‘-’**
  - **Multiplication ‘*’**
  - **Division ‘/’**

- **Ternary Operator:**
  - **x > y ‘?’ statement A;:Statement B;;**

- **Relational operators:**
  - **equal to ‘=’**
  - **not equal to ‘!=’**
  - **greater than ‘>’**
  - **lesser than ‘<’**
  - **Less than equal to ‘<=’**
  - **Greater than equal to ‘>=’**

- **Flexible data handling enhances versatility in SARS programming.**

- **Comprehensive support for data manipulation ensures robust functionality in SARS applications.**

SARS Features (Part-2)

- **SARS language offers distinctive loop constructs:**
  - **For**
  - **Forrange**
  - **While**

```
begin
{

for i in range(0:20)
{
print i;
}

}
end
```

```
int x = 10;
int y = 5;

if (x > y) {
    print "x is greater than y";
} elseif (x < y) {
    print "x is less than y";
} else {
    print "x is equal to y";
}
```

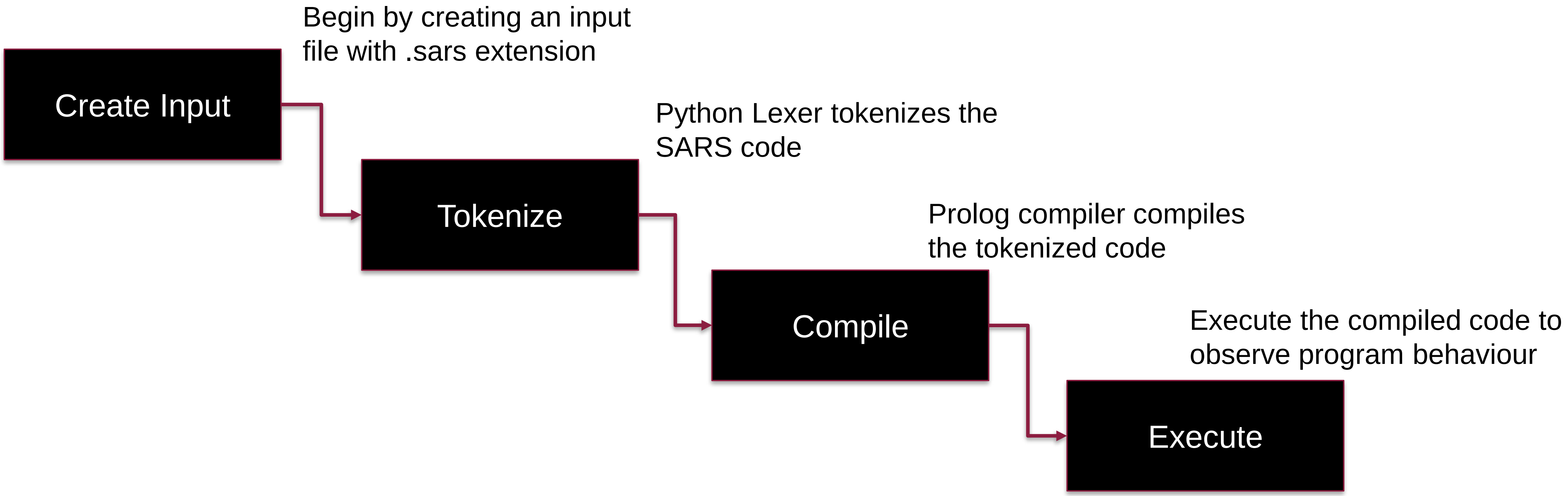
- **Conditional statements like If-else and Elseif enhance program logic.**

SARS Features (Part-3)

- SARS-specific operators enhance expressiveness and functionality.
- ‘Print’ Statement allows output of variable values during program execution.
- Unique operators contribute to the versatility and usability of SARS language.
- These features empower developers to create efficient and expressive SARS programs.

Execution Workflow

Execution of SARS program follows as systematic process:



Steps for Execution

Execution steps for SARS program.

1. **Prepare SARS Program: Write or obtain the SARS program code and save it with a .sars extension.**

1. **Create Input File: Create an input file containing the SARS program code with the .sars extension.**

2. **Use Lexer: Utilize the Lexer component by running the input file through a Python-based Lexer script.**

3. **Open SWI Prolog: Open SWI Prolog on the terminal or command prompt.**

4. **Load Compiler File: Load the SARS compiler file (SARS.pl) into SWI Prolog using the consult predicate.**

Steps for Execution

6. **Compile SARS Program: Compile the SARS program file using the sars_compiler predicate, specifying the path to the Lexer script and the input file.**

7. **Execute SARS Program: Run the compiled SARS program by calling the SARS predicate and providing the path to the Lexer script and the input file with the .sars extension.**

8. **Interact (if applicable): If the SARS program requires user input or interaction, provide the necessary input when prompted.**

9. **View Output: Once the program execution completes, view the output generated by the SARS program.**

10. **Exit SWI Prolog: Close the SWI Prolog interpreter when done with the execution.**

Sample Program

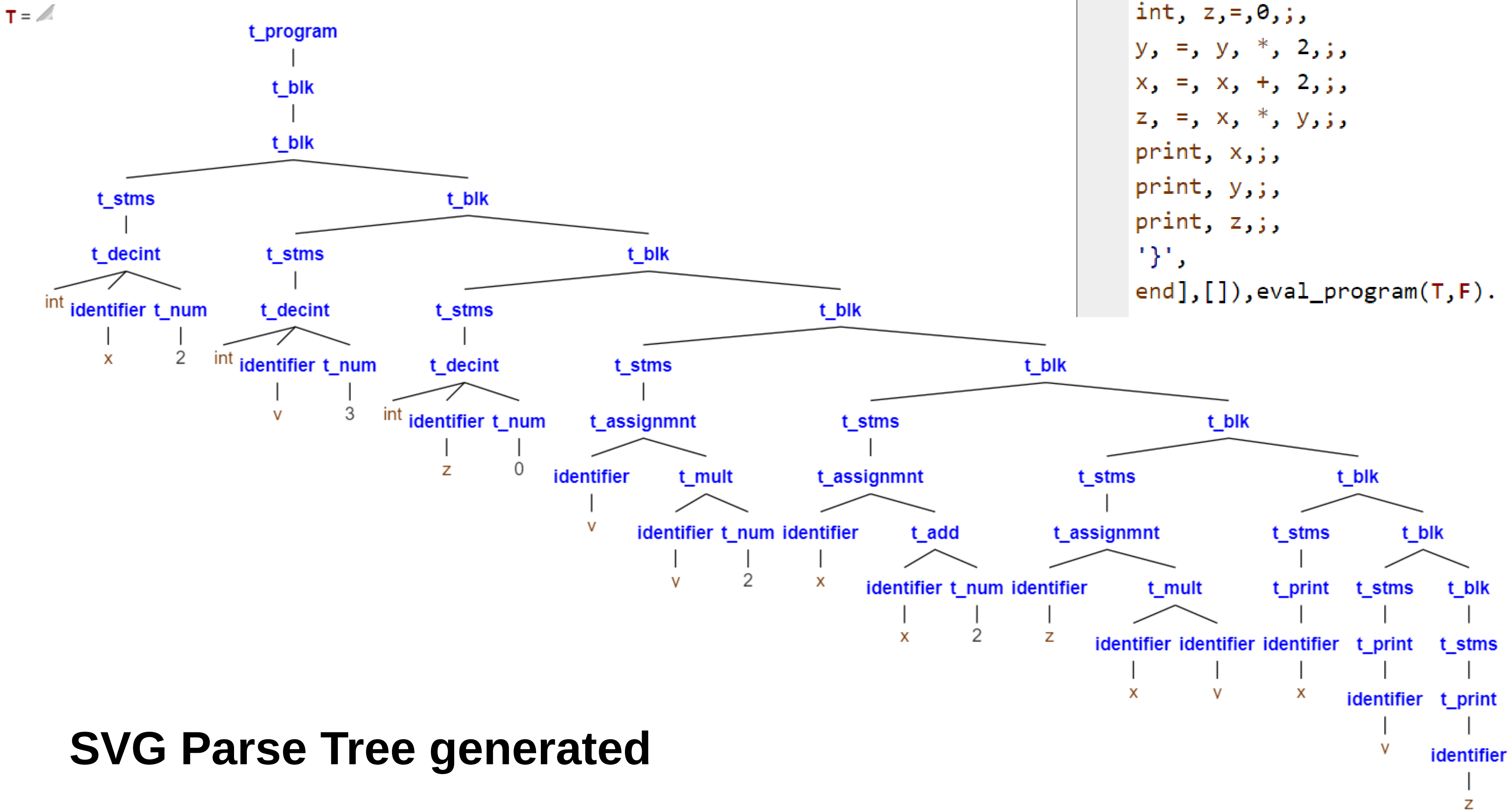
Here's a sample example of the SARS program.

```
begin
    int x = 5;
    int y = 10;
    int z = 0;

    for (int i = 0; i < 5; i++) {
        z = z + x;
    }

    if (z > 50) {
        print "Sum is greater than 50";
    } elseif (z < 10) {
        print "Sum is less than 10";
    } else {
        print "Sum is between 10 and 50";
    }
end
```

- **Initializes three integer variables x, y, and z with values 5, 10, and 0 respectively.**
- **Enters a for loop where x is added to z five times.**
- **Checks the value of z after the loop:**
  - **If z is greater than 50, it prints "Sum is greater than 50".**
  - **If z is less than 10, it prints "Sum is less than 10".**
  - **Otherwise, it prints "Sum is between 10 and 50".**

SVG Parse Tree

```
?-
program(T,[begin, '{', int, x,=,2,;,int, y,=,3,;,
int, z,=,0,;,
y, =, y, *, 2,;,
x, =, x, +, 2,;,
z, =, x, *, y,;,
print, x,;,
print, y,;,
print, z,;,
'}',
end],[]),eval_program(T,F).
```



SVG Parse Tree generated

Code Executions

Here's a sample example of the SARS program.

```
2 ?- ['E:/SER502-SARS-Team1/src/SARS.pl'].
true.
```

Program1: findFactorial.sars

```
2 ?- sars('E:/SER502-SARS-Team1/src/Lexer.py','E:/SER502-SARS-Team1/data/findFactorial.sars').
SARS Programming Language v1
SER 502 - Team 1
@Authors - Akash Rana, Sumeet Suryawanshi, Rohan Mathur, Sadanand Srinivasan

Parsing and Compiling in process......E:/SER502-SARS-Team1/data/findFactorial.sars

Tokens:
[begin,{,int,count,;,int,x,=,7,;,count,=,1,;,while,(,x,>,0,),{,count,=,count,*,x,;,x,=,x,-,1,;,},print,count,;,},end]

Parsed Tree:
t_program(t_blk(t_blk(t_stms(t_declare(int,identifier(count))),t_blk(t_stms(t_decint(int,identifier(x),t_num(7))),t_blk(t_stms(t_assi
gnmnt(identifier(count),t_num(1))),t_blk(t_stms(t_whileloop(t_condition(identifier(x),>,t_num(0)),t_blk(t_blk(t_stms(t_assignmnt(iden
tifier(count),t_mult(identifier(count),identifier(x)))),t_blk(t_stms(t_assignmnt(identifier(x),t_sub(identifier(x),t_num(1)))))))),t
_blk(t_stms(t_print(identifier(count))))))))))

result:
5040
true .
```

Code Execution

Program2: boolVar.sars

```
3 ?- sars('E:/SER502-SARS-Team1/src/Lexer.py','E:/SER502-SARS-Team1/data/boolVar.sars').
SARS Programming Language v1
SER 502 - Team 1
@Authors - Akash Rana, Sumeet Suryawanshi, Rohan Mathur, Sadanand Srinivasan

Parsing and Compiling in process......E:/SER502-SARS-Team1/data/boolVar.sars

Tokens:
[begin,{,bool,flag,=,true,;,int,x,=,28,;,if,(,x,>=,10,),{,x,=,x,+,2,;,},else,{,flag,=,false,;,},print,x,;,print,flag,;,},end]

Parsed Tree:
t_program(t_blk(t_blk(t_stms(t_decbool(bool,identifier(flag),true)),t_blk(t_stms(t_decint(int,identifier(x),t_num(28))),t_blk(t_stms(
t_if_condition(t_condition(identifier(x),>=,t_num(10)),t_blk(t_blk(t_stms(t_assignmnt(identifier(x),t_add(identifier(x),t_num(2))))))
,t_blk(t_blk(t_stms(t_assignmnt(identifier(flag),identifier(false))))))),t_blk(t_stms(t_print(identifier(x))),t_blk(t_stms(t_print(id
entifier(flag)))))))))))

result:
30
true
true .
```

Code Execution

Program2:findFibonacci.sars

```
4 ?- sars('E:/SER502-SARS-Team1/src/Lexer.py','E:/SER502-SARS-Team1/data/findFibonacci.sars').
SARS Programming Language v1
SER 502 - Team 1
@Authors - Akash Rana, Sumeet Suryawanshi, Rohan Mathur, Sadanand Srinivasan

Parsing and Compiling in process......E:/SER502-SARS-Team1/data/findFibonacci.sars

Tokens:
[begin,{,int,a,=,0,;,int,b,=,1,;,int,itr,=,0,;,while,(,itr,<,8,),{,int,curr,=,a,;,a,=,b,;,b,=,curr,+,b,;,print,a,;,itr,=,itr,+,1,;,},
},end]

Parsed Tree:
t_program(t_blk(t_blk(t_stms(t_decint(int,identifier(a),t_num(0))),t_blk(t_stms(t_decint(int,identifier(b),t_num(1))),t_blk(t_stms(t_
decint(int,identifier(itr),t_num(0))),t_blk(t_stms(t_whileloop(t_condition(identifier(itr),<,t_num(8)),t_blk(t_blk(t_stms(t_decint(in
t,identifier(curr),identifier(a))),t_blk(t_stms(t_assignmnt(identifier(a),identifier(b))),t_blk(t_stms(t_assignmnt(identifier(b),t_ad
d(identifier(curr),identifier(b)))),t_blk(t_stms(t_print(identifier(a))),t_blk(t_stms(t_assignmnt(identifier(itr),t_add(identifier(it
r),t_num(1))))))))))))))))))

result:
1
1
2
3
5
8
13
21
true
```

Thank you!

Github repository: https://github.com/ssuryaw5/SER502-SARS-Team1

Arizona State University