Minor Research Report 2

25/07/2023

Student: TRAN, Thanh Cong
Supervisor: Nao Hirokawa

I. Overview

- Objectives:
  - Implement IMP Intepreter in Typescript
- Progress: Done
- Source code:

  https://github.com/thanhtcptit/typescript-mr/blob/main/imp_interpreter/run.ts

II. Implementation

1. Token parser

- Purpose: Parse a IMP-syntax file into tokens for interpretation
- Method:
  - Use regular expression to match tokens to pre-defined patterns and assign tag

Tag	Pattern
SPACE	[\n\t\r]+
SYNTAX	[:=, (,), ;, >=, >, <=, <, ==, !=, &&, ||, !, if, else, while, end, +, -]
VARIABLE	[A-Za-z][A-Za-z0-9_]*
NUMBER	-?[0-9]+
BOOLEAN	[true, false]

- Example:

IMP program	Parsed tokens

n := 10;
x := 0;
 	
while n > 0
  x := x + n;
  n := n - 1
end
	[
	['n', 'VARIABLE'], [':=', 'SYNTAX'],
	['10', 'NUMBER'], [';', 'SYNTAX'],
	['x', 'VARIABLE'], [':=', 'SYNTAX'],
	['0', 'NUMBER'], [';', 'SYNTAX'],
	['while', 'SYNTAX'], ['n', 'VARIABLE'],
	['>', 'SYNTAX'], ['0', 'NUMBER'],
	['x', 'VARIABLE'], [':=', 'SYNTAX'],
	['x', 'VARIABLE'], ['+', 'SYNTAX'],
	['n', 'VARIABLE'], [';', 'SYNTAX'],
	['n', 'VARIABLE'], [':=', 'SYNTAX'],
	['n', 'VARIABLE'], ['-', 'SYNTAX'],
	['1', 'NUMBER'], ['end', 'SYNTAX']
	]

2. Statement parser
    - Purpose: Parser the parsed tokens into code statements for evaluation
    - Method:
        - Define 3 type of statements and corresponding structures

Type	Structure	Example

Assign	<variable> := <variable> or <number> or <boolean>	x := 1;
		y := x

If	if <condition>
	    <statements>
	else (optional)
	    <statements>
	end
		if x > 0
		    y := 1
		else
		    y := 2
		end

While	while <condition>
	    <statements>
	end
		while x >= 0
		    y := y + x;
		    x := x - 1
		end

        - Implement operator, arithmetic, and logic expression parser based on tokens' tag

Parser	Tag pattern of token(s) - sorted by priority
Arithmetic operator	- SYNTAX "+" - SYNTAX "-"
Logic operator	- SYNTAX "&&" - SYNTAX "||"
Comparison operator	- SYNTAX "<=" - SYNTAX "<" - SYNTAX ">=" - SYNTAX ">" - SYNTAX "!=" - SYNTAX "=="
Arithmetic expression	- NUMBER - VARIABLE - SYNTAX "(" + Arithmetic expression + SYNTAX ")" - Arithmetic expression + Arithmetic operator + Arithmetic expression
Logic expression	- BOOLEAN - VARIABLE

	- Arithmetic expression + Comparison operator + Arithmetic expression - SYNTAX “!” + Logic term - SYNTAX “(“ + Logic expression + SYNTAX “)” - Logic expression + Logic operator + Logic expression

- Implement statement parser using the above parsers as building blocks

Parser	Structure
Assign statement	VARIABLE + SYNTAX “:=” + (Arithmetic expression | Logic expression)
If statement	SYNTAX “if” + Logic expression + Lazy(Block statement) + Optional(SYNTAX “else” + Lazy(Block statement)) + SYNTAX “end”
While statement	SYNTAX “while” + Logic expression + Lazy(Block statement) + SYNTAX “end”
Block statement	(Assign statement | If statement | While statement) + Repeat(SYNTAX “;” + (Assign statement | If statement | While statement))

3. Statement evaluation
   - Evalute the parsed statements in a top-down fashion, using a Map<string, number | boolean> as the interpreter’s environment
4. Test programs
   - Arithmetic and logic operator

IMP program	Interpreter’s environment

a := (1 + (2 - 6)) + 3;
b := (a + 1) - 2;
c := a >= 0 || b >= 0;
 	
if c
    d := !c
end
	a: 0
	b: -1
	c: true
	d: false

   - Greatest common divisor

IMP program	Interpreter’s environment

x := 128;
y := 72;
	x: 8
	y: 0

while y != 0
   if y > x
      tmp := x;
      x := y;
      y := tmp
   else
      x := x - y
  end
end
	tmp: 0

- sum([1, n])

IMP program	Interpreter’s environment

n := 10;
sum := 0;
 	
while n > 0
   sum := sum + n;
   n := n - 1
end
	n: 0
	sum: 55

- Find the pivot number x in [1, n] such that sum([1, x]) == sum([x, n])

IMP program	Interpreter’s environment

n := 8;
c := 1;
x := -1;
 	
while c <= n
   i := 0;
   s1 := 0;
   while i < c
      i := i + 1;
      s1 := s1 + i
   end;
 	
   s2 := 0;
   while i <= n
      s2 := s2 + i;
      i := i + 1
   end;
 	
   if s1 == s2
	n: 8
	c: 9
	x: 6
	i: 9
	s1: 36
	s2: 8

```
        x := c
    end;

    c := c + 1
end
```