i219 Software Design Methodology
13. Case study 2
Mini programming language processor 1

Kazuhiro Ogata (JAIST)

Outline of lecture

- Minila
- Syntax of Minila
- Front end of Minila processor
- Minila interpreter

Minila (1)

- A mini programming language
- An imperative programming language
- Its processor consists of
  - An interpreter
  - A virtual machine
  - A compiler (code generator)

Minila (2)

A program in Minila that computes 2^{16}:

```
x := 2;
x := x*x;
x := x*x;
x := x*x;
x := x*x;
```

Minila (3)

A program in Minila that computes the 10^{th} factorial:

```
x := 1;
n := 1;
while (n = 10 || n < 10) do
  x := x*n;
  n := n+1;
od
```


Minila (4)

A program in Minila that computes the greatest common divisor of 19110 & 17850 with the Euclid's algorithm:

```
x := 19110;
y := 17850;
while y != 0 do
  tmp := x%y;
  x := y;
  y := tmp;
od
```

Minila (5)

A program in Minila that computes the positive integral part of the square root of 20000 with binary search:

```
v0 := 20000;  v1 := 0;  v2 := v0;
while v1 != v2 do
  if (v2 – v1)%2 = 0
    then v3 := v1+(v2 – v1)/2;
    else v3 := v1+(v2 – v1)/2+1;  fi
  if v3*v3 > v0
    then v2 := v3 – 1;
    else v1 := v3;  fi
od
```


Syntax of Minila (1)

P ::= S	(Program)
S ::= ε	(empty statement)
| var ':=' E ';'	(assign statement)
| 'if' E 'then' S 'else' S 'fi'	(if statement)
| 'while' E 'do' S 'od'	(while statement)
| S S	(sequential composition)

where var is a variable, E is an expression, and S is a statement.

Syntax of Minila (2)

```
E ::= num | '–' num | var | '–' var
    | '(' E ')' | '–' '(' E ')'
    | E '*' E | E '/' E | E '%' E
    | E '+' E | E '–' E
    | E '<' E | E '>' E | E '=' E | E '!=' E
    | E '&&' E | E '||' E
num ::= [0-9]+
var ::= [a-zA-Z][0-9a-zA-Z]*
```


Syntax of Minila (3)

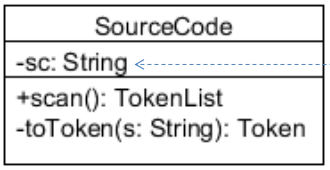
The precedence of the operators is as follows:

(unary) '–'	strongest
'*', '/', '%'	
'+', (bynary) '–'	
'<', '>', '=', '!='	
'&&', '||'	weakest

Every operator is left-associative, e.g. 3–2–1 is parsed as ((3–2) –1).

Front end of Minila processor (1)

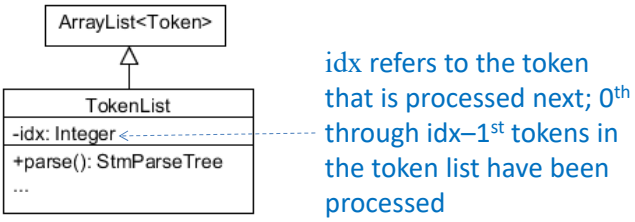
Feeding a program in Minila (a source code) into the Minila processor, the source code is stored as an object of class SourceCode.



The object makes a list of tokens from sc by receiving the message scan().

Front end of Minila processor (2)

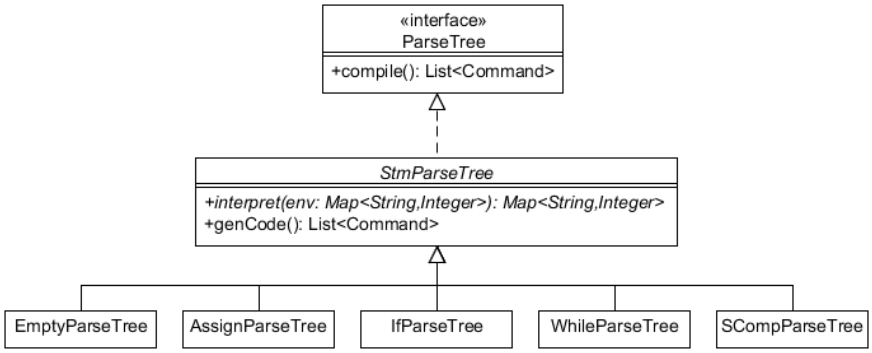
A list of tokens is represented as an object of class TokenList.



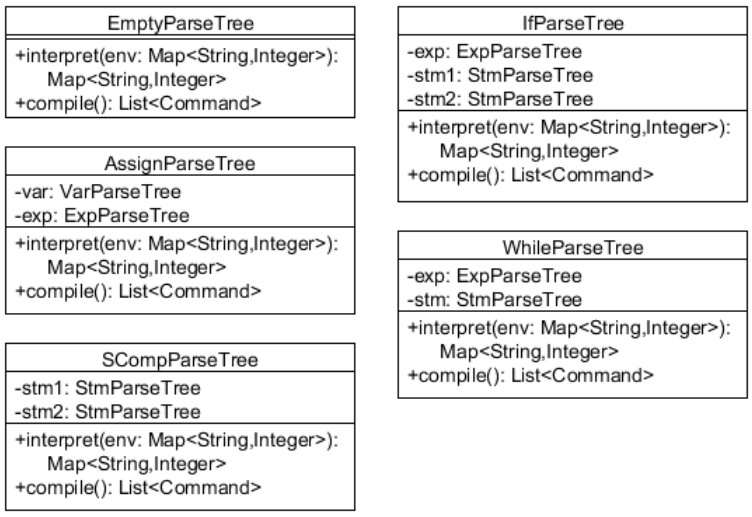
The object makes a parse tree from the list of tokens if the input program obeys the Minila syntax; displays a syntax error message otherwise.

Front end of Minila processor (3)

A parse tree is represented as an object of one of concrete classes that extends the abstract class StmParseTree.



Front end of Minila processor (4)



Front end of Minila processor (5)

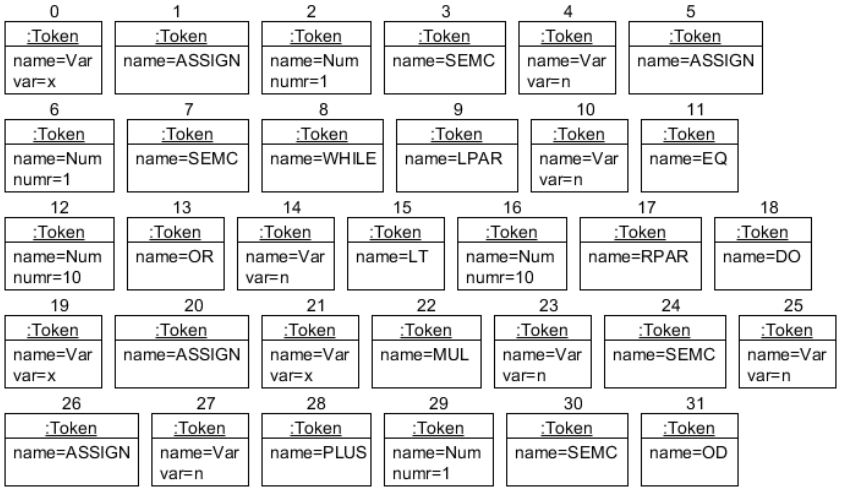
```
x := 1;  n := 1;
while (n = 10 || n < 10) do
  x := x*n;  n := n+1;
od
```

lexical analysis

[var: x, :=, num=1, ;, var: n, :=, num=1, ;, while, (, var: n, =, num=10, ||, var: n, <, num=10,), do, var: x, :=, var: x, *, var: n, ;, var: n, :=, var: n, +, num=1, ;, od]

Front end of Minila processor (6)

The object of TokenList looks like:



Front end of Minila processor (7)

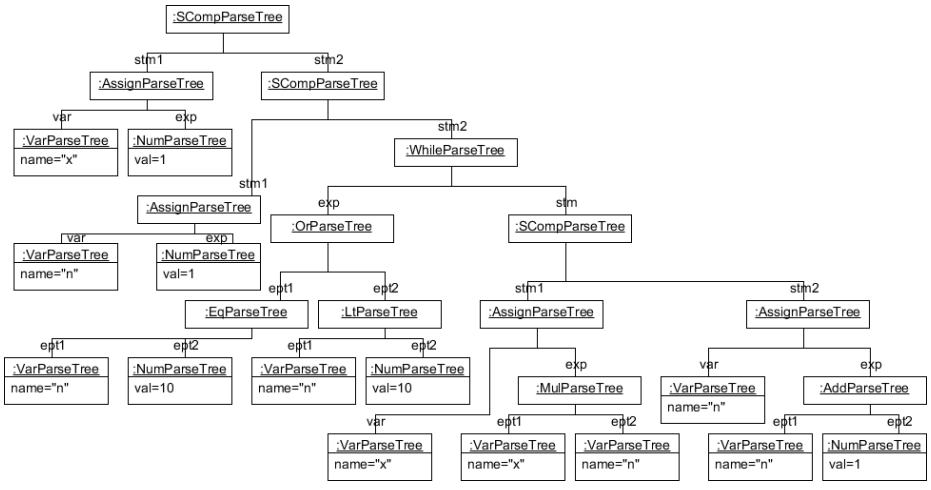
[var: x, :=, num=1, ;, var: n, :=, num=1, ;, while, (, var: n, =, num=10, ||, var: n, <, num=10,), do, var: x, :=, var: x, *, var: n, ;, var: n, :=, var: n, +, num=1, ;, od]

parsing

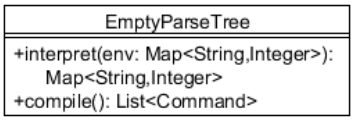
```
scomp(assign(x,1),
        scomp(assign(n,1),
                while(or(eq(n,10),lt(n,10)),
                        scomp(assign(x,mul(x,n)),
                                assign(n,add(n,1))))))
```


Front end of Minila processor (8)

The object diagram of the parse tree:

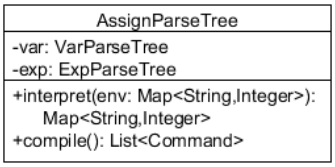


Minila interpreter (1)



When an object of EmptyParseTree receives interpret(env), it just returns env.

Minila interpreter (2)

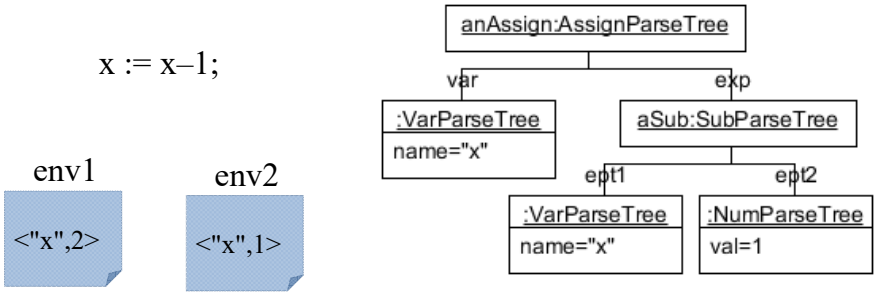


When an object of AssignParseTree receives interpret(env),

1. it sends interpret(env) to exp and obtains the result val,
2. modifies env such that val is associated with the name of var, making a new environment env', and
3. returns env'.

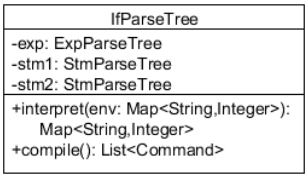
Minila interpreter (3)

x := x–1;



✓ When anAssign receives interpret(env1), it sends interpret(env1) to aSub, gets 1 and modifies env1 to get env2; it then returns env2.

Minila interpreter (4)



When an object of IfParseTree receives interpret(env),

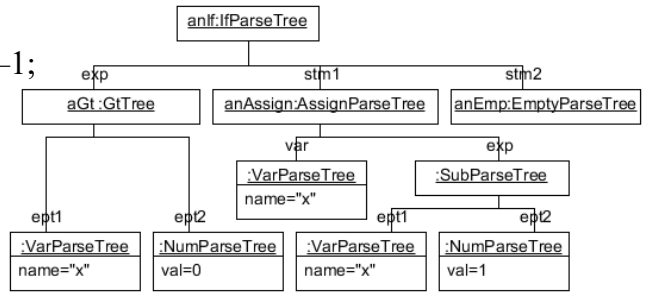
1. it sends interpret(env) to exp and obtains the result val, and
2. if val is not 0, it sends interpret(env) to stm1, obtains a new environment env' and returns env';
3. if val is 0, it sends interpret(env) to stm2, obtains a new environment env' and returns env'.

Minila interpreter (5)

env1

<"x",2>

if x > 0
then x := x–1;
else fi



env2

<"x",1>

env3

<"x",0>

✓ When anIf receives interpret(env1), it sends interpret(env1) to aGt and gets 1; This is why it sends interpret(env1) to anAssign and gets env2; it then returns env2.

✓ When anIf receives interpret(env3), it sends interpret(env3) to aGt and gets 0; This is why it sends interpret(env3) to anEmp and gets env3; it then returns env3.

Minila interpreter (6)

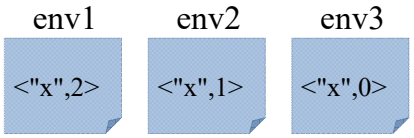
WhileParseTree
-exp: ExpParseTree -stm: StmParseTree
+interpret(env: Map<String,Integer>): Map<String,Integer> +compile(): List<Command>

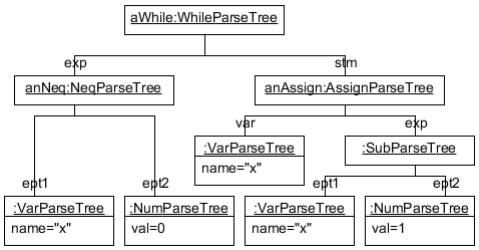
When an object of WhileParseTree receives interpret(env),

1. it sends interpret(env) to exp and obtains the result val, and
2. if val is 0, it returns env;
3. if val is not 0, it sends interpret(env) to stm, obtains a new environment env', sends interpret(env') to the object itself, obtains another new environment env'', and returns env''.

Minila interpreter (7)

```
while x != 0
  do x := x–1; od
```



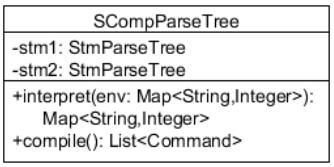


✓ When aWhile receives interpret(env1), it sends interpret(env1) to anNeq and gets 1; This is why it sends interpret(env1) to anAssign and gets env2; it then sends interpret(env2) to itself and gets env3; it finally returns env3.

✓ When aWhile receives interpret(env2), it sends interpret(env2) to anNeq and gets 1; This is why it sends interpret(env2) to anAssign and gets env3; it then sends interpret(env3) to itself and gets env3; it finally returns env3.

✓ When aWhile receives interpret(env3), it sends interpret(env3) to anNeq and gets 0; This is why it returns env3.

Minila interpreter (8)



When an object of SCompParseTree receives interpret(env),

1. it sends interpret(env) to stm1 and obtains a new environment env',
2. sends interpret(env') to stm2 and obtains another new environment env",
3. returns env".

Summary

- Minila
- Syntax of Minila
- Front end of Minila processor
- Minila interpreter