

i116: Basic of Programming

11. Programming language processor: syntax & parse trees

Kazuhiro Ogata, Canh Minh Do

Roadmap

- A mini-programming language: Minila
- Parse trees for Minila
- Scanner and parser for Minila

A mini-programming language: Minila

- **Minila** stands for a mini-programming language whose syntax is as follows:

Prog ::= Stm Empty Statement

Stm ::=

↓
|

Var := Exp ; |

if Exp then Stm else Stm fi |

while Exp do Stm od |

Stm Stm

*Var ::= [a-zA-Z][a-zA-Z0-9]**

Exp ::= ... (Please see lecture note 6, where *E* is used instead of *Exp*)

A mini-programming language: Minila

The program checks if 119 is a prime number.

```
x := 119;  
y := 2;  
result := 1;  
flag := 1;  
while flag do  
  if x % y = 0  
  then flag := 0;  
    result := 0;  
  else y := y+1; fi  
  if x = y then flag := 0; else fi  
od
```

The empty statement is used there.

A mini-programming language: Minila

The program computes 10!.

```
x := 1;  
y := 1;  
while y < 10 || y = 10  
do  
    x := x * y;  
    y := y + 1;  
od
```

A mini-programming language: Minila

The program computes the greatest common divisor of the following integers: 19110 and 17850.

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

A mini-programming language: Minila

The program computes the positive integral part of the following integer:

20000000000000000000

```
v0 := 20000000000000000000;  
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
```

Parse trees for Minila

- The three new statements have been introduced:
 - Empty statement
 - Conditional (**if**) statement
 - Loop (**while**) statement
- Then, we need to have a new class for parse trees of each statement.

Parse trees for Minila

```
class EmptyParseTree(StmParseTree):  
    def __str__(self):  
        return '(Empty Statement)'
```

EmptyStm

Parse trees for Minila

```
class IfParseTree(StmParseTree):
```

```
    exp = ExpParseTree()
```

```
    stm1 = StmParseTree()
```

```
    stm2 = StmParseTree()
```

```
def __init__(self, e, s1, s2):
```

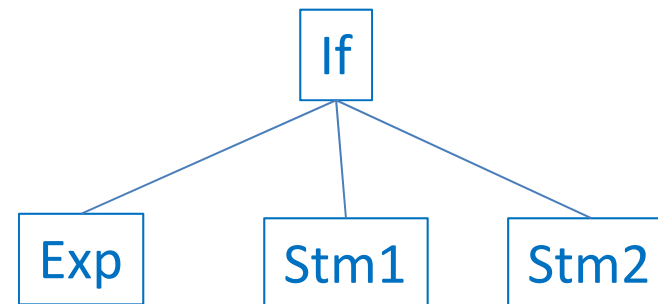
```
    self.exp = e
```

```
    self.stm1 = s1
```

```
    self.stm2 = s2
```

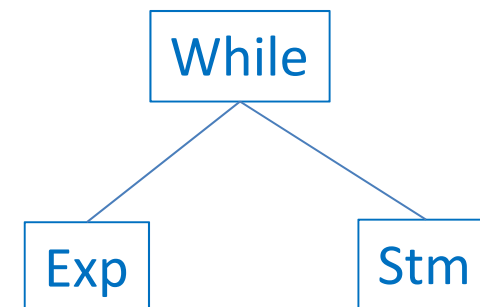
```
def __str__(self):
```

```
    return '(if ' + str(self.exp) + ' then ' + str(self.stm1) + ' else ' + str(self.stm2) + ' fi)'
```



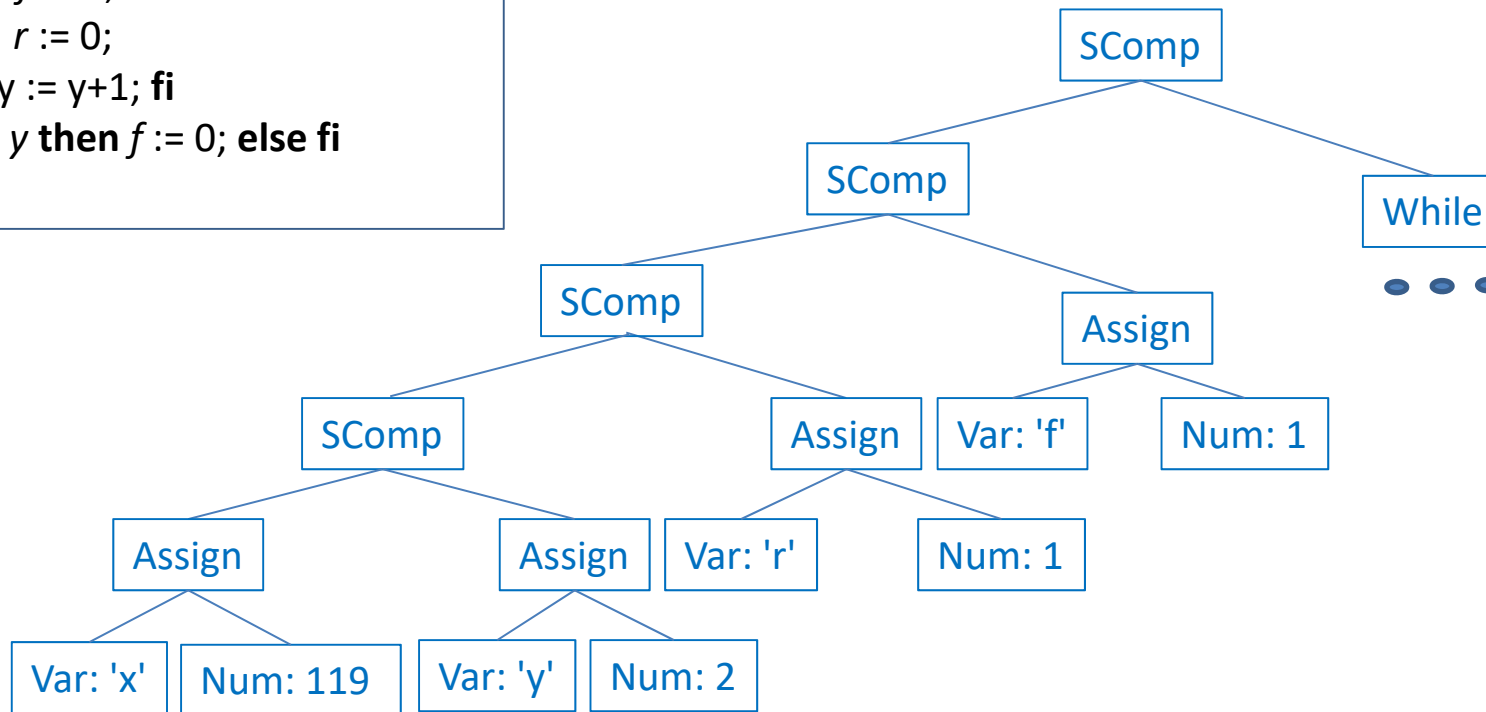
Parse trees for Minila

```
class WhileParseTree(StmParseTree):  
    exp = ExpParseTree()  
    stm = StmParseTree()  
    def __init__(self, e, s):  
        self.exp = e  
        self.stm = s  
    def __str__(self):  
        return '(while ' + str(self.exp) + ' do ' + str(self.stm) + ' od)'
```



Parse trees for Minila

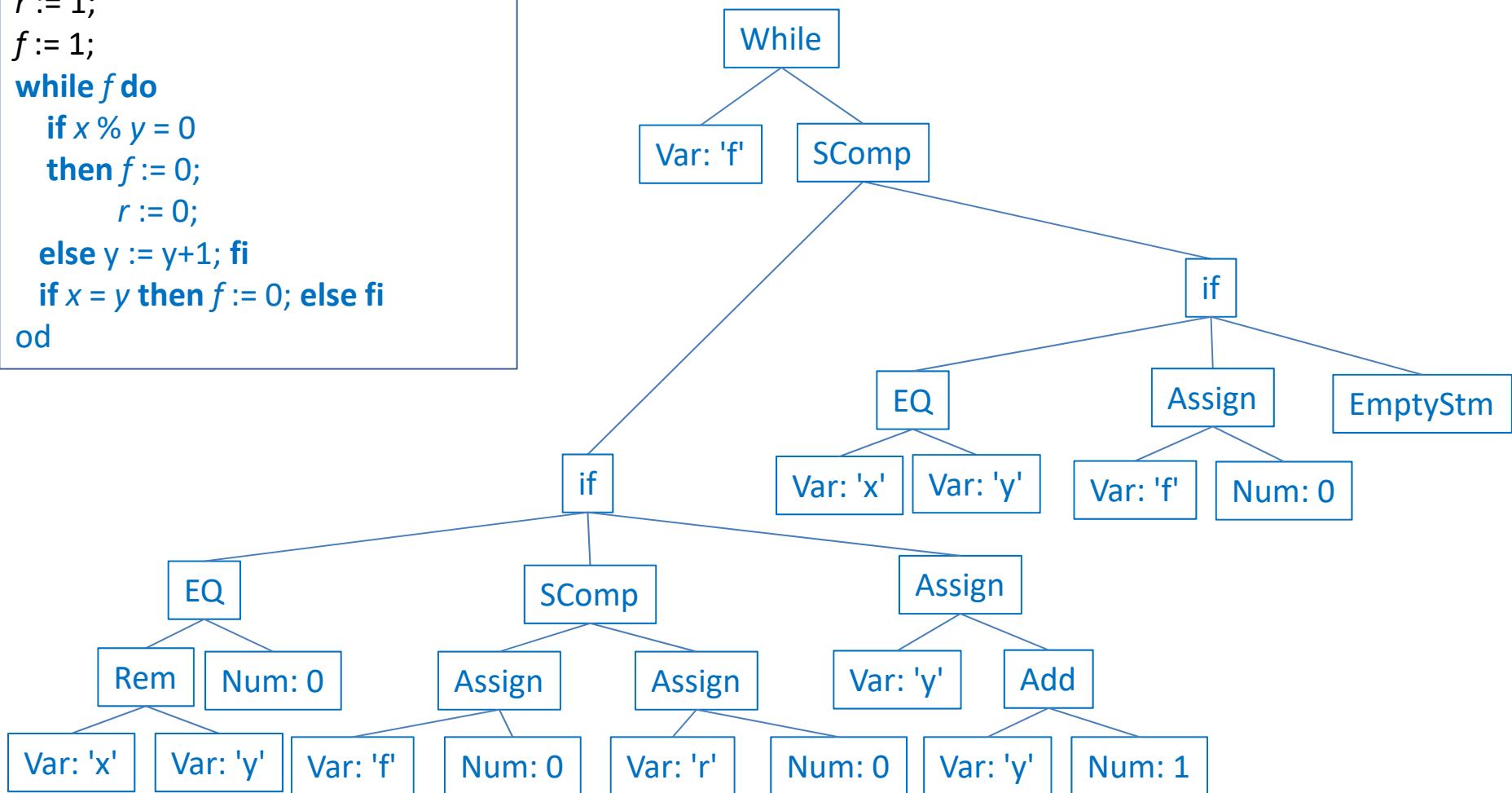
```
x := 119;  
y := 2;  
r := 1;  
f := 1;  
while f do  
  if x % y = 0  
  then f := 0;  
    r := 0;  
  else y := y+1; fi  
  if x = y then f := 0; else fi  
od
```



Parse trees for Minila

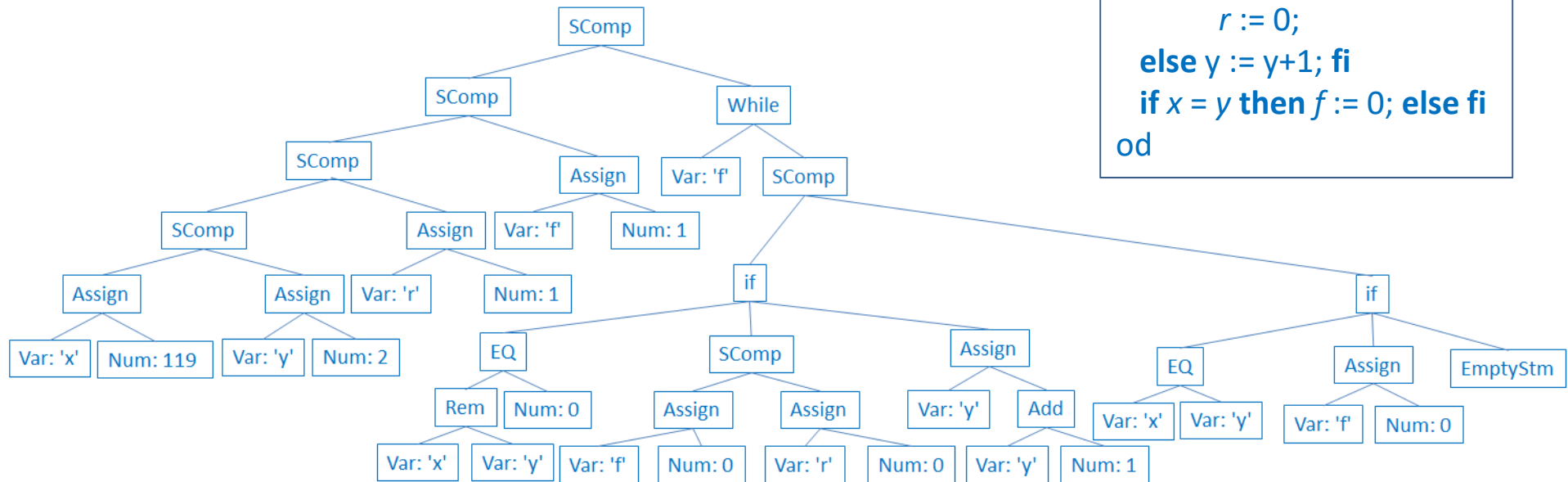
```

x := 119;
y := 2;
r := 1;
f := 1;
while f do
  if x % y = 0
  then f := 0;
    r := 0;
  else y := y+1; fi
  if x = y then f := 0; else fi
od
    
```



Parse trees for Minila

```
x := 119;  
y := 2;  
r := 1;  
f := 1;  
while f do  
  if x % y = 0  
  then f := 0;  
    r := 0;  
  else y := y+1; fi  
  if x = y then f := 0; else fi  
od
```



Parse trees for Minila

```
varX = VarParseTree('x')
varY = VarParseTree('y')
varR = VarParseTree('r')
varF = VarParseTree('f')
n119 = NumParseTree(119)
n2 = NumParseTree(2)
n1 = NumParseTree(1)
n0 = NumParseTree(0)
a1 = AssignParseTree(varX,n119)
a2 = AssignParseTree(varY,n2)
a3 = AssignParseTree(varR,n1)
a4 = AssignParseTree(varF,n1)
a5 = AssignParseTree(varR,n0)
a6 = AssignParseTree(varF,n0)
sc1 = SCompParseTree(a1,a2)
sc2 = SCompParseTree(sc1,a3)
sc3 = SCompParseTree(sc2,a4)
```

```
e1 = RemParseTree(varX,varY)
e2 = EQParseTree(e1,n0)
e3 = AddParseTree(varY,n1)
e4 = EQParseTree(varX,varY)
emps = EmptyParseTree()
if2 = IfParseTree(e4,a6,emps)
sc4 = SCompParseTree(a6,a5)
a7 = AssignParseTree(varY,e3)
if1 = IfParseTree(e2,sc4,a7)
sc5 = SCompParseTree(if1,if2)
while1 = WhileParseTree(varF,sc5)

pgm = SCompParseTree(sc3,while1)

print(sc3)
print(while1)
print(pgm)
```

Scanner and parser for Minila

- A **scanner** takes a program as a string, converting the string into a list of **tokens**.
- A **parser** takes a list of tokens, checking whether the list of tokens conforms to the Minila syntax and constructing the **parse tree** if it does so.

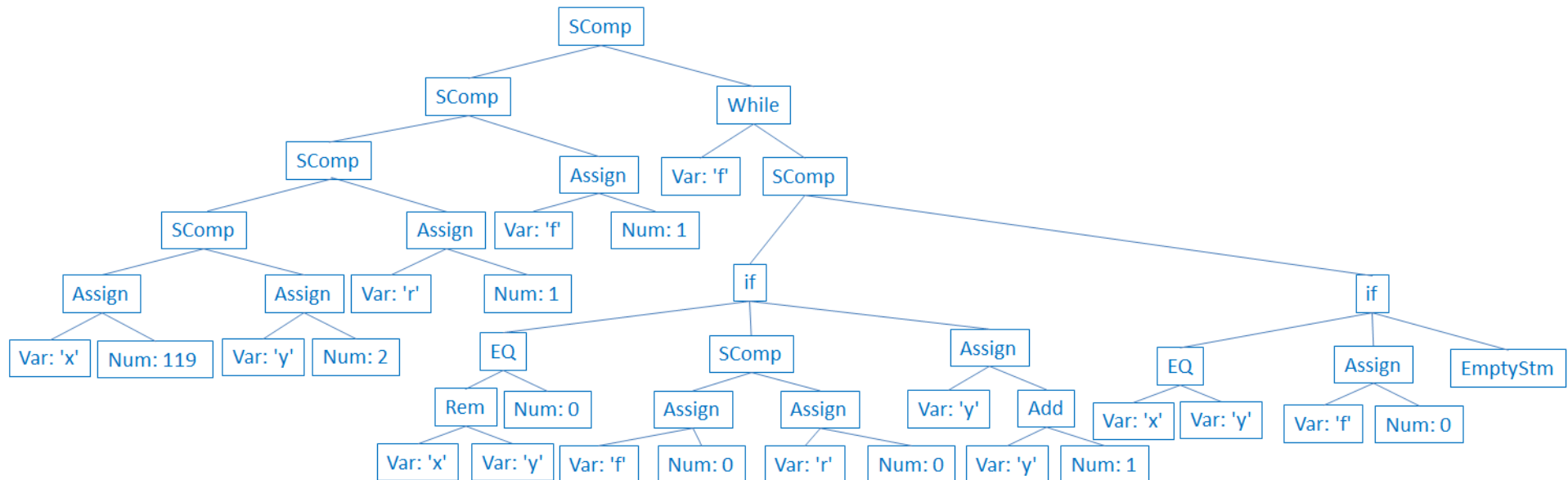
Scanner and parser for Minila

' x := 119; y := 2; r := 1; f := 1; while f do if x % y = 0 then f := 0; ...'

↓ scan

[var: x, :=, num: 119, ;; var: y, :=, num: 2, ;; var: r, :=, num: 1, ;; var: f, :=, num: 1, ;; while, var: f, do, if, var: x, %, var: y, =, num: 0, then, var: f, :=, num: 0, ;; ...]

↓ parse



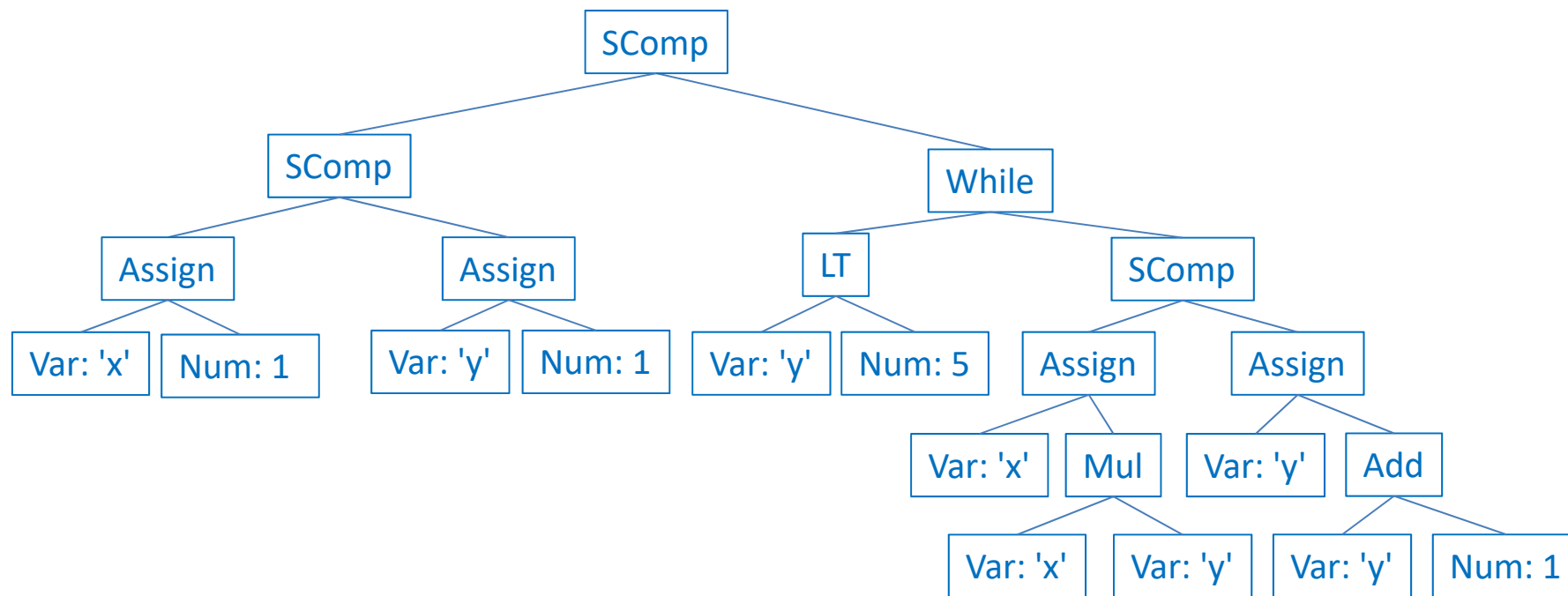
Scanner and parser for Minila

' x := 1; y := 1; while y < 5 do x := x * y; y := y + 1; od'

↓ scan

[var: x, :=, num: 1, ,, var: y, :=, num: 1, ,, while, var: y, <, num: 5, do, var: x, :=, var: x, *, var: y, ,, var: y, :=, var: y, +, num: 1, ,, od]

↓ parse



Scanner and parser for Minila

- The following files are available at the course website:
 - `parse.py`
 - `scan.py`
 - `token2.py`
 - `tname.py`
- `parse.py` imports `parseTree.py` in which the classes of parse trees are defined, and `scan.py` imports `misc.py` in which procedure `l2s` is defined.
- In `parse.py`, a class `TokenList` is defined and one of its methods is `parse()`:

```
tlo = TokenList(a list of tokens)  
pt = tlo.parse()
```

Scanner and parser for Minila

```
from scan import *
from parse import *
fact = '\
  x := 1;\
  y := 1;\
  while y < 5\
  do\
    x := x * y;\
    y := y + 1;\
  od'
tl = scan(fact)
tlo = TokenList(tl)
pt = tlo.parse()
print(fact)
print(l2s(tl))
print(pt)
```

The program that computes 4! is written as a string referred to as *fact*.

scan converts *fact* to a list of tokens referred to as *tl*.

A *TokenList* object referred to as *tlo* is made with *tl*.

parse checks whether *tl* conforms to the Minila syntax and constructs a parse tree referred to as *pt* if so.

Scanner and parser for Minila

```
from scan import *  
from parse import *  
fact = ' \  
  ' x := 1; \  
  ' y := 1; \  
  ' while y < 10 || y = 10' \  
  ' do' \  
    ' x := x * y; \  
    ' y := y + 1; \  
  ' od' \  
tl = scan(fact)  
tlo = TokenList(tl)  
pt = tlo.parse()  
print(fact)  
print(l2s(tl))  
print(pt)
```

The program computes 10!.

Scanner and parser for Minila

```
from scan import *
from parse import *
gcd = ' \
' x := 19110; \
' y := 17850; \
' while y != 0 do \
'   tmp := x%y; \
'   x := y; \
'   y := tmp; \
' od '
tl = scan(gcd)
tlo = TokenList(tl)
pt = tlo.parse()
print(gcd)
print(l2s(tl))
print(pt)
```

The program computes the greatest common divisor of the following two integers:

19110 and **17850**

Scanner and parser for Minila

```
from scan import *  
from parse import *  
isPrime = ' \  
' x := 119; \  
' y := 2; \  
' r := 1; \  
' f := 1; \  
' while f do \  
'   if x % y = 0 \  
'   then f := 0; \  
'     r := 0; \  
'   else y := y+1; fi \  
'   if x = y then f := 0; else fi \  
' od '
```

```
tl = scan(isPrime)  
tlo = TokenList(tl)  
pt = tlo.parse()  
print(isPrime)  
print(l2s(tl))  
print(pt)
```

The program checks whether **119** is a prime number.

Scanner and parser for Minila

```
from scan import *  
from parse import *  
sr = ' \  
' v0 := 2000000000000000000; \  
' 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 '
```

```
tl = scan(sr)  
tlo = TokenList(tl)  
pt = tlo.parse()  
print(sr)  
print(l2s(tl))  
print(pt)
```

The program calculates the positive integral part of the following integer:
2000000000000000000

Parse trees for Minila

