Lecture 10 - Oct. 18

Syntactic Analysis

CFG: Case Studies
Semantic Analysis vs. Ambiguity

Announcements

Quiz 2:
1. no proofs
2. no essays.

- ANTLR tutorial
  + RE
  + CFG
  + OOP and Composite & visitor design patterns
- Project to be released by next Tuesday's class
- A possible alternative to ProgTest?
  14:30 to 16:00, Tuesday, November 1
- Programming Test date:
  + 2:00pm to 3:20pm on Saturday, October 29
  + Venue to be confirmed (LAS building)
  + Practice Test

to be finally confirmed on Thurs. class

- Quiz 2 on Thursday, October 19

Discussion: Compare Two CFGs

Expression	→	*IntegerConstant*
	|	*BooleanConstant*
	|	*BinaryOp*
	|	*UnaryOp*
	|	(*Expression*)
IntegerConstant	→	*Digit*
	|	*Digit IntegerConstant*
	|	−*IntegerConstant*
Digit	→	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
BooleanConstant	→	TRUE
	|	FALSE

1. V2 does semantic grouping of operators

? 1+2+3

2. V2 is less ambiguous

∵ it does not accept 2 => 8

↳ accepted by v1

↳ rejected by v2

only 1 parse tree by v1/v2 CFG

v1

BinaryOp	→	*Expression* + *Expression*
	|	*Expression* − *Expression*
	|	*Expression* * *Expression*
	|	*Expression* / *Expression*
	|	*Expression* && *Expression*
	|	*Expression* || *Expression*
	|	*Expression* => *Expression*
	|	*Expression* == *Expression*
	|	*Expression* /= *Expression*
	|	*Expression* > *Expression*
	|	*Expression* < *Expression*
UnaryOp	→	! *Expression*

v2

expected numerical exp.

ArithmeticOp	→	*ArithmeticOp* + *ArithmeticOp*
	|	*ArithmeticOp* − *ArithmeticOp*
	|	*ArithmeticOp* * *ArithmeticOp*
	|	*ArithmeticOp* / *ArithmeticOp*
	|	(*ArithmeticOp*)
	|	*IntegerConstant*
RelationalOp	→	*ArithmeticOp* == *ArithmeticOp*
	|	*ArithmeticOp* /= *ArithmeticOp*
	|	*ArithmeticOp* > *ArithmeticOp*
	|	*ArithmeticOp* < *ArithmeticOp*
LogicalOp	→	*LogicalOp* && *LogicalOp*
	|	*LogicalOp* || *LogicalOp*
	|	*LogicalOp* => *LogicalOp*
	|	! *LogicalOp*
	|	(*LogicalOp*)
	|	*RelationalOp*
	|	*BooleanConstant*

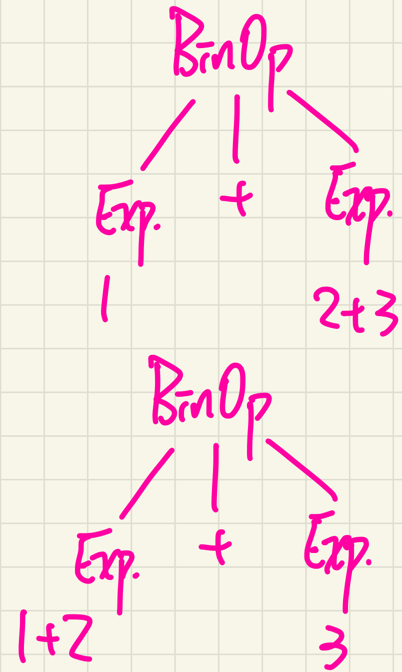
boolean exp.

1+2+3

BinaryOp → Expression + Expression
| Expression − Expression
| Expression * Expression
| Expression / Expression
| Expression && Expression
| Expression || Expression
| Expression => Expression
| Expression == Expression
| Expression /= Expression
| Expression > Expression
| Expression < Expression

UnaryOp → ! Expression

Ambiguity?



Context-Free Grammar (CFG): Example Version 1

```
Expression        →  IntegerConstant
                  |  BooleanConstant
                  |  BinaryOp
                  |  UnaryOp
                  |  ( Expression )

IntegerConstant   →  Digit
                  |  Digit IntegerConstant
                  |  -IntegerConstant

Digit             →  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

BooleanConstant   →  TRUE
                  |  FALSE
```

```
BinaryOp  →  Expression + Expression
          |  Expression - Expression
          |  Expression * Expression
          |  Expression / Expression
          |  Expression && Expression
          |  Expression || Expression
          |  Expression => Expression
          |  Expression == Expression
          |  Expression /= Expression
          |  Expression > Expression
          |  Expression < Expression

UnaryOp   →  ! Expression
```

Example: (1 + 2) => (5 / 4)

Is this an AST/PT with valid meaning?

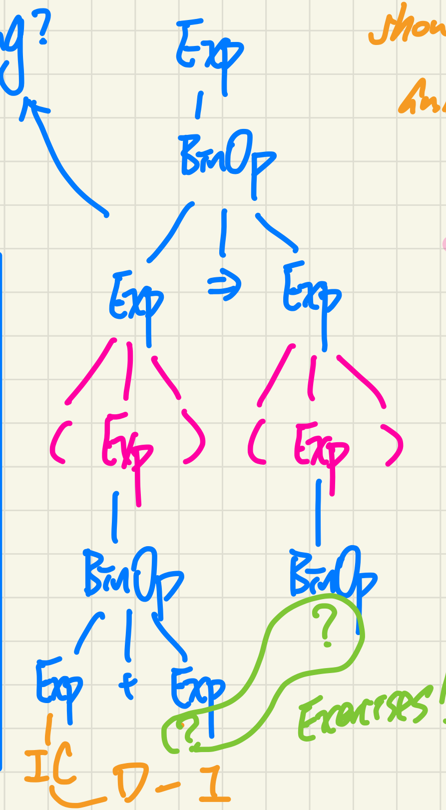
contains semantic error to be discovered

not appropriate witness for showing the language ambiguity

semantic analysis

5 - 6

↳ appropriate witness for proving ambiguity (exercise)



Context-Free Grammar (CFG): Example Version 1

1+2+3

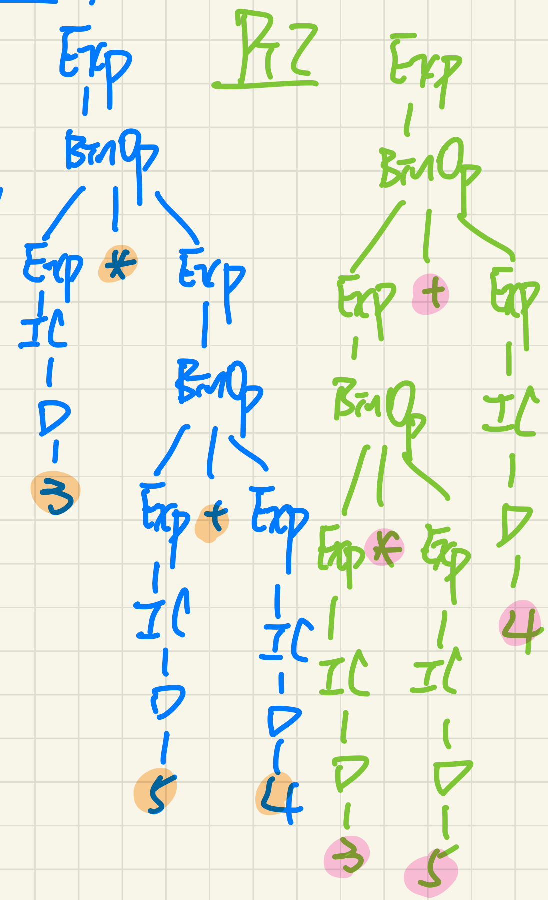
Example: 3 * 5 + 4

Expression	→	*IntegerConstant*
	|	*BooleanConstant*
	|	*BinaryOp*
	|	*UnaryOp*
	|	(*Expression*)
IntegerConstant	→	*Digit*
	|	*Digit IntegerConstant*
	|	−*IntegerConstant*
Digit	→	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
BooleanConstant	→	`TRUE`
	|	`FALSE`

BinaryOp	→	*Expression* + *Expression*
	|	*Expression* − *Expression*
	|	*Expression* * *Expression*
	|	*Expression* / *Expression*
	|	*Expression* && *Expression*
	|	*Expression* || *Expression*
	|	*Expression* => *Expression*
	|	*Expression* == *Expression*
	|	*Expression* /= *Expression*
	|	*Expression* > *Expression*
	|	*Expression* < *Expression*
UnaryOp	→	! *Expression*

PT1

witness of ambiguity



Context-Free Grammar (CFG): Example Version 2

```
Expression        →  ArithmeticOp
                  |  RelationalOp
                  |  LogicalOp
                  |  ( Expression )

IntegerConstant   →  Digit
                  |  Digit IntegerConstant
                  |  -IntegerConstant

Digit             →  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

BooleanConstant   →  TRUE
                  |  FALSE
```

```
ArithmeticOp  →  ArithmeticOp + ArithmeticOp
              |  ArithmeticOp - ArithmeticOp
              |  ArithmeticOp * ArithmeticOp
              |  ArithmeticOp / ArithmeticOp
              |  (ArithmeticOp)
              |  IntegerConstant

RelationalOp  →  ArithmeticOp == ArithmeticOp
              |  ArithmeticOp /= ArithmeticOp
              |  ArithmeticOp > ArithmeticOp
              |  ArithmeticOp < ArithmeticOp

LogicalOp     →  LogicalOp && LogicalOp
              |  LogicalOp || LogicalOp
              |  LogicalOp => LogicalOp
              |  ! LogicalOp
              |  (LogicalOp)
              |  RelationalOp
              |  BooleanConstant
```

Example: (1 + 2) => (5 / 4)

for V2, it's a parse error (no AST/PT can be built).

↳ not preferred as the user of compiler needs more feedback (e.g. Eclipse)

Context-Free Grammar (CFG): Example Version 2

```
Expression       →  ArithmeticOp
                 |  RelationalOp
                 |  LogicalOp
                 |  ( Expression )

IntegerConstant  →  Digit
                 |  Digit IntegerConstant
                 |  -IntegerConstant

Digit            →  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

BooleanConstant  →  TRUE
                 |  FALSE
```

```
ArithmeticOp  →  ArithmeticOp + ArithmeticOp
              |  ArithmeticOp - ArithmeticOp
              |  ArithmeticOp * ArithmeticOp
              |  ArithmeticOp / ArithmeticOp
              |  (ArithmeticOp)
              |  IntegerConstant

RelationalOp  →  ArithmeticOp == ArithmeticOp
              |  ArithmeticOp /= ArithmeticOp
              |  ArithmeticOp > ArithmeticOp
              |  ArithmeticOp < ArithmeticOp

LogicalOp     →  LogicalOp && LogicalOp
              |  LogicalOp || LogicalOp
              |  LogicalOp => LogicalOp
              |  ! LogicalOp
              |  (LogicalOp)
              |  RelationalOp
              |  BooleanConstant
```

Q: No semantic analysis at all for Version 2 grammar?

Example: ~~(1 + 2) => (5 - (2 + 3))~~

((1+2)>0) ⇒ (4/(5-(2+3)) > 0)

division by zero.

for simple cases, it might be worth checking it's not 0.

Person P;

P. setName("Jim");

Context-Free Grammar (CFG): Example Version 2

Expression	→	*ArithmeticOp*
	|	*RelationalOp*
	|	*LogicalOp*
	|	(*Expression*)
IntegerConstant	→	*Digit*
	|	*Digit IntegerConstant*
	|	−*IntegerConstant*
Digit	→	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
BooleanConstant	→	TRUE
	|	FALSE

ArithmeticOp	→	*ArithmeticOp* + *ArithmeticOp*
	|	*ArithmeticOp* − *ArithmeticOp*
	|	*ArithmeticOp* * *ArithmeticOp*
	|	*ArithmeticOp* / *ArithmeticOp*
	|	(*ArithmeticOp*)
	|	*IntegerConstant*
RelationalOp	→	*ArithmeticOp* == *ArithmeticOp*
	|	*ArithmeticOp* /= *ArithmeticOp*
	|	*ArithmeticOp* > *ArithmeticOp*
	|	*ArithmeticOp* < *ArithmeticOp*
LogicalOp	→	*LogicalOp* && *LogicalOp*
	|	*LogicalOp* || *LogicalOp*
	|	*LogicalOp* => *LogicalOp*
	|	! *LogicalOp*
	|	(*LogicalOp*)
	|	*RelationalOp*
	|	*BooleanConstant*

Example: 3 * 5 + 4.

Exercise: show the grammar is ambiguous.