Lecture 2
Describing Syntax
Instructor: Hossein Hojjat

January 19, 2018
Compiler Phases

Source Code (concrete syntax)

\[
\text{if (x == 0) x = x + 1;}
\]

Lexical Analysis

Token Stream

\[
\text{if (x == 0) x = x + 1;}
\]

Syntax Analysis (Parsing)

Abstract Syntax Tree (AST)

\[
\begin{array}{c}
\text{IF} \\
\text{==} \\
x \text{0} \\
\text{=} \\
x \text{+} \\
x \text{1}
\end{array}
\]

Semantic Analysis (Name Analysis, Type Analysis, ...)

Attributed AST

\[
\begin{array}{c}
\text{IF} \\
\text{==} \\
\text{boolean} \\
x \text{int} \\
0 \text{int} \\
\text{=} \\
x \text{int} \\
\text{+} \\
x \text{int} \\
1 \text{int}
\end{array}
\]

Code Generation

\[
\begin{array}{c}
16: \text{iload} \_2 \\
17: \text{ifne} \ 24 \\
20: \text{iload} \_2 \\
21: \text{iconst} \_1 \\
22: \text{iadd} \\
23: \text{istore} \_2 \\
24: \ldots
\end{array}
\]
How to describe a programming language?

- We need to provide:

1. **Syntax:** which strings of symbols are valid expressions in the language?
2. **Semantics:** what do valid expressions actually mean, or how do they behave?
Some Java syntax rules:

- Use a semicolon (";") to separate two statements
- Enclose the condition of an IF expression inside parentheses

Some semantics rules for valid Java expressions:

- \( x++ \) increment the value of variable \( x \) by 1
- \( x + 1 \) calculate the sum of \( x \) and 1
Describing Syntax

• Informal description using natural languages (English)

Pros.

• Explain high-level concepts to beginners

Cons.

• Imprecise, vague, tedious and repetitive
• Impossible to develop tools to analyze such descriptions

• List all valid programs

Cons:

• There exists arbitrarily long valid programs even for small languages
Describing Syntax

- **Formal languages and automata:**
  - Branch of CS that formalizes the properties of “languages” over strings and their syntax
  - Benefits of precise descriptions based on formal languages theory
    - Document what programs a compiler should accept or reject
    - Develop compiler phases (lexer, parser) using compiler generating tools

John Backus was the first to employ a formal technique for specifying the syntax of programming languages (Algol 60)
• While-Language is a small language we use to illustrate basic concepts
• “While” because it has \texttt{while} and \texttt{if} as the only control statements
  • no procedures, no exceptions
• All variables are of type integer
• Variables not declared, they are initially zero
• No objects, No pointers, No arrays
Convert if to while

• How to express conditional statement

```java
if (cond) {
    expr
}
```

• using a while statement?
Convert if to while

- While-language is Turing-complete! (although looks very simple)
- Does this program always terminate for any initial value of $x$?

```java
while (x > 1) {
    if (x % 2 == 0) {
        x = x / 2;
    } else {
        x = 3 * x + 1;
    }
}
```
While-language is Turing-complete! (although looks very simple)

Does this program always terminate for any initial value of x?

```java
while (x > 1) {
    if (x % 2 == 0) {
        x = x / 2;
    }
    else {
        x = 3 * x + 1;
    }
}
```

- Collatz Conjecture - open!
- Paul Erdős: “Mathematics may not be ready for such problems.”
Reasons for Unbounded Program Size

while (x < y) {
    x = y + x*(y + 3*(z + 12*(x - 7)));
    while (356436346 > x) {
        while (y < 100) {
            strangeVar67a = x + z;
            y = x + y + z;
            System.out.println("x" + x);
        }
    }
}
Tokens (Words) of the While Language

Ident ::= letter (letter | digit)*

integerConst ::= digit digit*

stringConst ::= "AnySymbolExceptQuote*"

keywords ::= if | else | while | println

special symbols ::= ( ) && < == + - * / % ! - { } ; ,

letter ::= a | b | c | ... | z | A | B | C | ... | Z

digit ::= 0 | 1 | ... | 8 | 9
while (x < y) {
    x = y + x*(y + 3*(z + 12*(x - 7)));
    while (356436346 > x) {
        while (y < 100) {
            strangeVar67a = x + z;
            y = x + y + z;
            System.out.println("x" + x);
        }
    }
}
Compiler Phases

Source Code (concrete syntax)
\[ \text{if} \ (x == 0) \ x = x + 1; \]

Lexical Analysis

Regular Expressions for Tokens

Token Stream
\[ \text{if} \ (x == 0) \ x = x + 1; \]

Syntax Analysis (Parsing)

Abstract Syntax Tree (AST)

Semantic Analysis (Name Analysis, Type Analysis, ...)

Attributed AST

Error

Code Generation

Machine Code

16: iload_2
17: ifne 24
20: iload_2
21: iconst_1
22: iadd
23: istore_2
24: ...
Reasons for Unbounded Program Size

constants of any length
variable names of any length
string constants of any length
words - tokens

while (x < y) {
    x = y + x*(y + 3*(z + 12*(x - 7)));
}

while (356436346 > x) {
    while (y < 100) {
        strangeVar67a = x + z;
        y = x + y + z;
        System.out.println("x"+x);
    }
}

nesting of expressions
nesting of statements

sentences
• Describe sentences using (possibly recursive) rules of a context-free grammar

program ::= statmt*
statmt ::= println( stringConst , ident )
    | ident = expr
    | if ( expr ) statmt (else statmt)?
    | while ( expr ) statmt
    | {statmt* }
expr ::= intLiteral | ident
    | expr (&& | < | == | + | - | * | / | % ) expr
    | ! expr | - expr
While Language without Nested Loops

\[
\text{statmt} ::= \text{println}(\text{stringConst}, \text{ident}) \\
| \text{ident} = \text{expr} \\
| \text{if (expr) statmt (else statmt)?} \\
| \text{while (expr) statmtww} \\
| \{\text{statmt}* \}
\]

\[
\text{statmtww} ::= \text{println}(\text{stringConst}, \text{ident}) \\
| \text{ident} = \text{expr} \\
| \text{if (expr) statmtww (else statmtww)?} \\
| \{\text{statmtww}* \}
\]
Compiler Phases

Source Code (concrete syntax)

Regular Expressions for Tokens

Token Stream

Context-Free Grammar

Abstract Syntax Tree (AST)

Attributed AST

Machine Code

16: iload_2
17: ifne 24
20: iload_2
21: iconst_1
22: iadd
23: istore_2
24: ...
To get abstract syntax trees:

- Start from context-free grammar for tokens
- Remove punctuation characters
- Interpret rules as tree descriptions, not string descriptions

\[
\text{statmt} ::= \text{println( stringConst , ident )} \quad \text{PRINT}(\text{string,ident}) \\
| \quad \text{id} = \text{expr} \quad \text{ASSIGN}(\text{id,expr}) \\
| \quad \text{if ( expr ) statmt (else statmt)?} \quad \text{IF}(\text{expr,statmt,statmt}) \\
| \quad \text{while ( expr ) statmt} \quad \text{WHILE}(\text{expr,statmt}) \\
| \quad \{\text{statmt* }\} \quad \text{BLOCK}(\text{List[statmt]})
\]