Introduction to Syntax Analysis

Sebastian Hack

http://compilers.cs.uni-saarland.de

Compiler Construction Core Course 2017
Saarland University

Syntax Analysis in the Compiler Structure

Abstract Syntax vs. Concrete Syntax

Syntax is typically defined using context-free grammars

Abstract syntax describes the structure of a program:

$$
\begin{array}{lcl}
s & \to & \mathsf{While}(e, s) \\
 & | & \mathsf{If}(e, s, s) \\
 & | & \mathsf{ExprStmt}(e) \\
 & & \\
e & \to & \mathsf{Const}[v] \\
 & | & \mathsf{Id}[n] \\
 & | & \mathsf{Neg}(e) \\
 & | & \mathsf{Plus}(e, e) \\
 & | & \mathsf{Minus}(e, e) \\
 & \vdots &
\end{array}
$$

Concrete syntax describes how programs "look" like as text:

$$
\begin{array}{lcl}
s & \to & \texttt{while (}e\texttt{) }s \\
 & | & \texttt{if (}e\texttt{) }s\texttt{ else }s \\
 & | & e\texttt{;} \\
 & & \\
e & \to & \texttt{NUM} \\
 & | & \texttt{ID} \\
 & | & \texttt{- }e \\
 & | & e\texttt{ + }e \\
 & | & e\texttt{ - }e \\
 & | & \texttt{(}e\texttt{)} \\
 & \vdots &
\end{array}
$$

Lexing

- The terminals of the concrete syntax are so-called tokens that are produced by a lexer from the characters of the program text
- A token consists of
  - An ID that characterizes its type (identifier, number, semicolon, etc.)
  - Source code coordinates (for error reporting)
  - The corresponding program text (if necessary)
- Structure of tokens typically described by regular expressions
- Theory doesn't require lexing (context-free languages contain regular languages) but lexing makes the specification of the concrete syntax and the parser simpler

Lexing: Example

Program Text

```
q = 0;
r = x;
while (y <= r) {
  r = r - y;
  q = q + 1;
}
```

Tokens (coordinates omitted)

```
ID("q") ASSIGN INT_CONST("0") SEMI

ID("r") ASSIGN ID("x") SEMI

WHILE LPAREN VAR("y") LE VAR("r")
RPAREN LBRACE

ID("r") ASSIGN ID("r") MINUS ID("y")
SEMI

ID("q") ASSIGN ID("q") PLUS
INT_CONST("1") SEMI

RBRACE
```

Parsing

- The parser analyses the token stream and
  - either constructs the AST
  - or produces error messages on syntax errors
- Parsing requires an unambiguous grammar:
  Every syntactically correct input program has exactly one derivation
- Straight-forward grammars for common languages are ambiguous, common issues:
  - Precedence and associativity of operators
  - Dangling else
- We'll discuss different solutions to this problem in the parsing session

Parsing Example

Tokens

```
ID("q") ASSIGN INT_CONST("0") SEMI
ID("r") ASSIGN ID("x") SEMI
WHILE LPAREN VAR("y") LE VAR("r")
RPAREN LBRACE
ID("r") ASSIGN ID("r") MINUS ID("y")
SEMI
ID("q") ASSIGN ID("q") PLUS
INT_CONST("1") SEMI
RBRACE
```

Abstract Syntax Tree