Compilerbau für die Common Language Run-Time

Lexikalische Analyse

Lexik

- Lineare Zerlegung der Eingabezeichen in Tokens
  - üblicherweise ohne Berücksichtigung des Kontexts
- Tokenklassen
  - Definition oft in Form regulärer Ausdrücke

Reguläre Ausdrücke

- ε
  - $L(\varepsilon) = \{\varepsilon\}$
- a (aus Alphabet A)
  - $L(a) = \{ a \}$
- Seien r, s reguläre Ausdrücke
  - r | s
    - $L(r \mid s) = L(r) \cup L(s)$
  - rs
    - $L(rs) = \{ xy \mid x \in L(r), y \in L(s) \}$
  - r*
    - $L(r^*) = \{ x_1x_2\ldots x_n \mid xi \in L(r)\}$
  - (r)
    - $L((r)) = L(r)$

ANTLR

- *Another Tool For Language Recognition*
  - Terence Parr, seit 1989
  - Weiterentwicklung von PCCTS
- Kombination der meisten Compilerstufen
  - lexikalische Analyse
  - syntaktische Analyse
  - Repräsentation abstrakter Syntax
  - (Synthese:) *tree parser*
- Semantische Aktionen in Java, C++, C#, Python

Lexikalische Analyse mit ANTLR

- Syntax:

  class *Klassenname* extends Lexer;

  options

  tokens

  lexer rules

- options { *key* = *value; ...* }
  - Programm-Optionen: language=”CSharp”; ...
  - Lexer-Optionen:
    - charVocabulary = ‘\3’ .. ‘\177’;
    - caseSensitive = true;
    - ignore = Freiraum;
    - ...
  - pro lexikalischer Regel:
    - testLiterals = true;
    - ...

Lexikalische Analyse mit ANTLR

- gleicher Mechanismus für lexikalische und syntaktische Analyse
  - LL(k) mit zusätzlichen Prädikaten
- Regelnamen mit Großbuchstaben, Regelkörper EBNF-ähnlich
  - ID : (‘a’..’z’)+;
  - WS : (‘ ‘ | ‘\t’ | ‘\r’ | ‘\n’)+;
  - COMMENT: “//” (~(‘\n’ | ‘\r’))*;
- *protected rules*: keine Token generiert
  - protected DIGIT: ‘0’..’9’;
  - INTLIT: DIGIT+;
- generierte Klasse implementiert TokenStream:
  - public Token nextToken() throws TokenStreamException

Lexer-Aktionen

- eingebettet in Grammatikregeln, in {...}
- WS : (‘ ‘
  | ‘\t’
  | ‘\n’ { newline(); } // increase line number
) { $setType(Token.SKIP); }
  ;
-

Schlüsselwörter und reservierte Bezeichner

- Variante 1: Lexerregeln für jedes Schlüsselwort
- Variante 2: “automatische Schlüsselwortlisten”
  - Alle Strings in Grammatik werden in *literals table* aufgenommen
    - z.b. “while” LPAREN expr RPAREN body
  - alle Tokens werden gegen *literals table* getestet, falls testLiterals-Option für dieses Token wahr ist

Semantische Werte

- Wert assoziiert mit Token
  - standardmäßig Zeichenfolge des Tokens (String)
- Auslassung von Zeichen: !
  - CHAR_LITERAL: '\''! . '\''!
- Manipulation des Tokenwerts: $append, $getText, $setText
- Festlegung des Tokentyps: $setType
- Festlegung des Tokenobjekts: $setToken