Programming Languages: Lecture 10 Syntax Specifications of Programming Languages

Rishabh Dhiman

21 January 2022

1 Extended Backus-Naur Form

BNF is a CFG specification. Extended comes from its use of regex operators like Kleene Closure and optional specifier.

EBNF specification is a collection of rules which defines the CFG of a language.

PLs in general are Turing-complete. However, CFG cannot specify a Turing machine. So it tends to encode context-sensitive features too.

The syntax analysis of CSGs is very complex. However, we want parsing to be quick, low-order polynomial time, ideally linear.

Parsing CFGs can be made quick, the language *specification* gives rules for parsing, along with this there are language manual which explains the semantics of these languages. The *semantics* contain context-sensitive features.

The EBNF rules are as follows,

Start Symbol. The very first rule gives the productions of the start symbol of the grammar.

Non-terminals. Uses English words or phrases to denote non-terminal symbols. The words or phrases are chosen to be suggestive of the nature or meaning of the constructors.

Metasymbols.

- Sequences of constructs enclosed in '{' and '}' denote zero or more occurences of the construct (c.f. Kleene Clsoure on regex)
- Sequences of constructs enclosed in '[' and ']' are optional, ie, there only be zero or one occurence of the sequence
- Constructs are enclosed in '(' and ')' to group them together.
- '|' separates alternatives.
- '::=' defines the productions of each non-terminal symbol.
- '.' terminates the possibly many rewrite rules for a non-terminal.

The same ASCII alphabet is used as both alphabet and operator.

1.1 Balanced Paranthesis

Consider the CFG BP_3 given by

$$S \to \varepsilon \mid (S)S \mid [S]S \mid \{S\}S.$$

In EBNF, it will be given by

$\langle BracketSeq \rangle$ ::= $\{\langle Bracket \rangle\}$

$\langle Bracket \rangle$ ::= $\langle LeftParen \rangle\ \langle BracketSeq \rangle\ \langle RightParen \rangle$
| $\langle LeftSqbracket \rangle\ \langle BracketSeq \rangle\ \langle RightSqbracket \rangle$
| $\langle LeftBrace \rangle\ \langle BracketSeq \rangle\ \langle RightBrace \rangle$

$\langle LeftParen \rangle$ ::= '('

$\langle RightParen \rangle$ ::= ')'

$\langle LeftSqbracket \rangle$ ::= '['

$\langle RightSqbracket \rangle$ ::= ']'

$\langle RightBrace \rangle$ ::= '{'

$\langle LeftBrace \rangle$ ::= '{'

You can sepcify EBNF in EBNF, see the hypernotes.

In the olden days we used to enclose EBNF symbols in <>, it's called ASCII-EBNF.

The EBNF of prolog can be fit in just two slides, again see hypernotes.

2 Regex as a Language

Given a nonempty finite alphabet A,

- every regular language over A is also context-free
- the set of regular expression over A is not regular but is context-free

This is because of arbitrary nesting and they have associativity and precedence rules.

Both the alphabet of regex and operators of REGEXP(A) happen to be the same.

// There's a tangent about lexical generators here. Go over it when you have the time.