Ordered Context-Free Grammars

Ziemowit Laski

Department of Information and Computer Science

University of California, Irvine

Irvine, CA 92697-3425, USA

laski@ics.uci.edu

April 30, 1999 (Updated: June 16, 1999)

Overview

- Parser Construction Blues
- Precedence in Context-Free Grammars
- Ordered Context-Free Grammars
- Derivation Properties of OCFGs
- Parsing: $\mathfrak{Bertha}^{\mathsf{TM}}$ to the Rescue
- Possible Future Work

Parser Construction Blues (1)

- In directional parsing, we are restricted to deterministic context-free languages (DCFLs)
- At first, things seem pretty rosy:
  - All DCFLs have an LR(1) grammar [Knu65]
  - Many languages are in LL(1), although...
  - For any finite k, there exists a DCFL that is not LL(k)
- But:
  - Finding an LR(1) grammar is very difficult, and its existence undecidable; things are even hairier for LALR(1) (of `yacc` fame);
  - LL(k) $\subset$ LR(1)

Parser Construction Blues (2)

- Even if is possible to find a suitable LL(k) or LALR(1) grammar for our DCFL,
  - "Language *translation* is a harder and more important problem than language *recognition*." [PQ96]
  - Semantic bindings (i.e., associating grammar rules with actions) become very tricky
  - Beneficial to leave the structure of the grammar intact, even at the expense of larger lookahead ($k > 1$)
- Our claim: Precedence is an alternative/complementary disambiguation tool to $k > 1$

Precedence in Context-Free Grammars (1)

- Enforcing Precedence Through Grammar Structure:

```
E : E '-' T
  | T ;
T : T '*' F
  | F ;
F : '-' num
  | num ;
```

```
E : T E' ;
E': '-' T E'
  | /* NULL */ ;
T : F T' ;
T': '*' F T'
  | /* NULL */ ;
F : '-' num
  | num ;
```

  - LR(k) version – Numerous unit rules
  - LL(k) version – Additional symbols

Precedence in Context-Free Grammars (2)

- "Deterministic Parsing of Ambiguous Grammars" in `yacc`:

```
%left '-'
%left '*'
%right UMINUS
    :
E : E '-' E
  | E '*' E
  | '-' E   %prec UMINUS
  | num ;
```

  - Precedence assigned to tokens (hence the `UMINUS`)
  - "Ambiguous context-free grammar together with disambiguating rules" [AJU75]

Precedence in Context-Free Grammars (3)

- Precedence/associativity assigned to *productions*

```
E : '-' E
  | E '*' E  %left (10)
  | E '-' E  %left (10)
  | num ;
```

  - Precedence implicit in production order
  - Associativity and precedence are viewed as an *integral* part of the grammar

Ordered Context-Free Grammars (1)

- Based on ordinary CFGs
- $G = (V_N, V_T, S, P)$, where
  (1) V_N, V_T and S are as in a CFG, and $V =_{def} V_N \cup V_T$
  (2) $P : V_N \longmapsto R$, where $R \subseteq \mathbb{P}(\leq_{\mathrm{P}}, V^* \longmapsto \mathbb{N})$, enumerates the productions in G; specifically,
    (a) P maps each nonterminal A to a *strict partial order* of possible *right sides* $\langle \omega, a \rangle \in R$
    (b) Each right side, in turn, maps the *right part* ω to a Boolean *associativity vector* $a \in \{0,1\}^{[0,\, rc(A,\omega)]}$, where
    (c) The *recursion count* $rc(A, \omega)$ denotes the number of occurrences of nonterminal A in the sentential form ω

Ordered Context-Free Grammars (2)

- If $p = \langle A, R \rangle \in P$ and $\langle \omega, a \rangle \in R$, then we may instead write $p = [A \longrightarrow \omega : a] \in P$.
- If $a = 0^{rc(A,\omega)}$, then p is *non-associative*; if $a = 1^{rc(A,\omega)-1}0$, *left-associative*, if $a = 01^{rc(A,\omega)-1}$, *right-associative*
  - A generalized definition of associativity
  - However, left-, right- full- and non-associativity suffice for directional parsing
- Let G be an OCFG; we may then define a *core* CFG, $G' = \mathrm{CORE}(G)$, obtained from G by discarding all precedence and associativity information
- The precedence relation is undefined for productions with different left parts

Derivation Properties of OCFGs (1)

- An OCFG derivation $S \overset{*}{\underset{\mathrm{P}}{\Longrightarrow}} \gamma$ is a CFG derivation $S \Longrightarrow \gamma$ that *additionally* respects the following invariant:
  - If two productions with the same left part $A \in V_N$ are used in *immediate* succession to construct some path in the derivation tree, then either:
    (1) The production being used first has lower precedence, or
    (2) The two productions have incomparable precedence, or
    (3) They have equal precedence, in which case the associativity of the parent production $[A \longrightarrow \omega : a]$ must be obeyed:
      (a) The i-th occurrence of A in ω may be expanded via the child production only if $a[i] = 1$.
      (b) Productions with higher precedence may expand any A in ω.

Derivation Properties of OCFGs (2)

- An OCFG derivation/parse tree is "computationally correct" wrt a depth-first (bottom-up) traversal
- Given some word $w \in V_T^*$ in a DCFL,
  - An OCFG G will allow *at most* as many distinct parse trees for w as CORE(G), hence reducing/eliminating ambiguity
  - G may also invalidate *all* parse trees for w, rendering it syntactically incorrect
- In general, $L(G) \subseteq L(\text{CORE}(G))$

Derivation Properties of OCFGs (3)

- Given a partial order $\leq_{\mathrm{P}}$ among productions in G with the same left part, we can induce a partial order $\leq_{\mathrm{rm\,P}}$ among entire OCFG *derivations* $S \underset{\mathrm{rm\,P}}{\overset{*}{\Longrightarrow}} \gamma$ in G
  - Must choose consistent method of expanding nonterminals in intermediate sentential forms
  - Righttmost derivation seems the logical choice
    - Corresponds to a bottom-up, left-to-right scan of the input
    - Can be nicely mapped to closure computations within parser states, and to state transitions in a deterministic pushdown automaton (DPDA)

Derivation Properties of OCFGs (4)

- Given two rightmost derivations $S \overset{*}{\underset{\text{rm P}}{\Longrightarrow}} \gamma_1$ and $S \overset{*}{\underset{\text{rm P}}{\Longrightarrow}} \gamma_2$,
  - Their ordering is the same as the ordering of the productions used first, unless
  - The same production begins both derivations, in which case the succeeding productions are compared in a recursive manner
- Allows us to establish an ordering among items in each DPDA state, and use it to resolve many shift-reduce and reduce-reduce conflicts
- On the other hand, this may increase the total number of states – given a set of items, more than one distinct partial order may be possible

Parsing: Bertha™ to the Rescue (1)

- A Java™-based parser generator (syntax, semantics, implementation)
- Handles LALRP(1) grammars, which are OCFG analogues of LALR(1)

```
start symbol E { int val;
  reduce("-", E expr) { val = - expr.val; }
  assoc(left) reduce(E expr1, "*", E expr2) { ... }
  assoc(left) reduce(E expr1, "-", E expr2) { ... }
  reduce(num i) { ... }
  symbol num {
    :
  }
}
```

Parsing: Bertha™ to the Rescue (2)

- Terminal symbols are defined along with non-terminals, use `scan()` instead of `reduce(...)`
- Nested `start symbol`s are allowed, useful for processing whitespace and comments (e.g., `javadoc`)
- All `symbol`s may have attributes, as they are mere Java `class`es; the generated parser is type-safe

Possible Future Work

- Investigate the formal conditions under which the LRP(k) and LALRP(k) parser construction algorithms create additional states, beyond those created by the LR(k)/LALR(k) algorithms
  - Empirical observation: they practically never occur
- Apply Pager's [Pag77] or Spector's [Spe88] compaction algorithms to extend **Bertha** to handle LRP(1) grammars
- It is possible to nesting symbols within one another, so...
  - Within the scope of the outer symbol, it is possible to use inherited attributes
  - May lead to some precedence-enhanced variant of left-corner (LC(k)) parsing, a method which combines LL(k) and LR(k) techniques