Finite-state Approximation of Constraint-based Grammars using Left-corner Grammar Transforms

Brown University

Mark Johnson

ACLT-COLING 1998
Partial evaluation
Composition/Erasure
Left-Corner (LC)

Summary
Right Linear Grammars

• LC parsers require only finite stack-depth to parse left linear or

  LC parsing applies directly to LCS

Why use LC approximation?

• LCS languages can be manipulated via PS calculus

• LC parsing has some psycholinguistic validity

  • can be used as oracle to guide LC parser

  • linear time recognition

  • FSM processing is faster

Why approximate LCS with FSMs?
<table>
<thead>
<tr>
<th>Remaining Input</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D \land D \vert$</td>
<td>$D \land D \vert$</td>
</tr>
<tr>
<td>$D \land D \vert D \vert N$</td>
<td>$D \land D \vert$</td>
</tr>
<tr>
<td>$D \land D \vert D \vert N$</td>
<td>$S$</td>
</tr>
</tbody>
</table>

### State Transition Function $\delta$:

- $\delta: \text{Stacks of nonterminals and terminals} \times \text{Stacks of nonterminals and terminals}$

### Non-deterministic Top-down Parsing
Finite values

In many UCs, the syntactically potential features range over
approximation accepts a subset of UC language

- Ignore categories whose complexity exceeds some bound

- Restriction (a.k.a. abstraction) (Sibber 1985)

ubounded UC categories

approximation accepts a subset of UC language

- Collapse all states sharing a common prefix

approximation accepts a subset of UC language

- Ignore state stacks larger than some fixed bound

ubounded state stack size

FS approximations to TD states
States of a TD parser
A left-corner (LC) parser exhibits finite state size on both left-linear and right-linear CFGs (*). A left-corner (LC) parser uses $\mathcal{L}(G)$. A LC parser for grammar $G$ acts isomorphically to a top-down parser using $\mathcal{L}(G)$.
\[ N \in \mathcal{V} : \epsilon \leftarrow \mathcal{V} \setminus \mathcal{A} \]
\[ P \ni \not\epsilon X \leftarrow \mathcal{B} \]
\[ N \in \mathcal{V} : \mathcal{B} \setminus \mathcal{V} \leftarrow \mathcal{A} \setminus \mathcal{V} \]
\[ X \ni \not\epsilon \mathcal{L} \]
\[ N \in \mathcal{V} : \mathcal{L} \setminus \mathcal{A} \leftarrow \mathcal{A} \]

\[
\pi_{\mathcal{L} \cap \mathcal{N}} \times \mathcal{N} \cap \mathcal{N} = \pi_{\mathcal{L} \setminus \mathcal{A}} \setminus \mathcal{A} \]

Non-terminals of each production is recognized bottom-up
Left-corner of each production is recognized top-down

Left-corner grammar transform
Parsing with IC(\mathfrak{C}) : start
Parsing with $\lambda C$ : shift DET
A - X → β A - B 
A - A → ε 
A ∈ N.

\[ A - X \rightarrow \beta A - B : B \rightarrow X \beta \epsilon P. \]

Parsing with \( L(G) \):

\[ S \]

[Diagram of a parsing tree with labeled nodes including 'the dog', 'ran fast', 'the', 'DET', 'NP', 'VP', 'S', 'S-DET', 'V', 'ADV', 'VP', 'S-S', 'VP-V', 'VP-VP', 'A']
Left-linear $C$ right-linear $\mathcal{IC}(C)$ finite states

States of an IC parser
States of an LC parser (cont.)

- Right-linear \( G \Rightarrow \) unbounded TD states in \( \mathcal{LC}(G) \)
fast ran
Vp
Vp-Np
the do not run
the det
ds

S

\textit{Epsilon-removal after LIC transform}

\textbf{Linear C right-linear LIC(C \cdot) finite TD states}
Partial evaluation/Composition

Converts binary branches into (almost) binary branches •
\[ P \in \sigma \quad X \leftrightarrow B \quad : (\sigma, \varphi X - \varphi) \in \mathcal{R} \]

Such productions can be implemented as FSM arcs. E.g.: 

Exactly one transformed rule per input item 

All but one schema are right-linear 

\[ P \in \sigma \quad X \leftrightarrow B \quad : P \in \sigma \quad \forall \alpha \in \mathcal{R} \]

\[ \forall \alpha \in \mathcal{R} \quad \forall \alpha \in \mathcal{R} \]

\[ \forall \alpha \in \mathcal{R} \quad B \in \mathcal{R} \]

\[ \forall \alpha \in \mathcal{R} \quad S \alpha \]

Special case of binary productions
Helps characterize the errors in a FS approximation

Geometry

Geometry of LC state complexity is associated with a specific tree size, LC state complexity increases the stack state

Because only one production schema
- Which guides TCG parser for C
- FS transducer emits rule scheme used at each transition.

- Obtaining parse trees from PSM transitions
- Identifying useless productions in \( \mathcal{L}(C) \) (link table)
- Classifying unification grammar categories

Odds and ends
Constructions for which the approximation is exact

A characterization of \( L \) state complexity identities

The approximation is exact for left linear and right linear CF grammars

A finite-state approximation can be directly constructed from

right recursion

Left-corner grammar transforms convert left recursion into

Conclusion