Everything You Always Wanted to Know About Parsing

Part VI : Parse Forests

Giorgio Satta
University of Padua, Italy

ESSLLI, August 2013
In natural language processing applications, grammars are massively ambiguous.

For a given input string, extracting parse trees one by one has two main disadvantages:

- Number of parse trees is exponential in the input sentence length.
- Downstream processing of each tree, e.g. semantic interpretation, translation, etc., is done independently of the other trees, failing to recognize common subparts and resulting in redundant processing.
A **shared and packed parse forest** is a **compact** representation for set $T(G, w)$

Phrases in $T(G, w)$ can be **packed**
Phrases in $T(G, w)$ can be **shared**
A common representation for a shared and packed parse forest is a CFG generating only the parse trees in $T(G, w)$, up to tree homomorphism.

As we will see, this approach produces parse forest representations of size polynomial in $|G|$ and $|w|$. Such parse forests can be derived from the parse table $T$ resulting from the run of our tabulation algorithm.

This establishes a strong connection between the recognition and the parsing problem.
Several **notational variants** of the CFG representation of a **shared and packed parse forest** have been investigated in the literature:

- **and-or graphs**
- **hyperedge graphs**

The algorithms for the construction of all these representations are related to an algorithm proposed in [Bar-Hillel et al., 1964] for **intersecting** a context-free language and a regular language.
We consider here the Cocke-Kasami-Younger (CKY) algorithm

Let $G$ be a CFG in Chomsky normal-form and let $w = a_1 \cdots a_n$; we construct the \textit{shared and packed parse forest} $G_w$ from the table $T$ produced in a run of the algorithm

The \textbf{nonterminals} of $G_w$ are the items $[i, A, j]$ used by the tabular algorithm
CKY Parsing

The **productions** of $G_w$ are obtained as follows:

- for each deduction rule

  $$
  (k, B, j) \quad (j, C, i) \\
  \frac{}{\quad (k, A, i)}
  $$

  applied in the construction of $\mathcal{T}$, add to $G_w$ the production

  $$
  [k, A, i] \rightarrow [k, B, j] [j, C, i]
  $$

- for each deduction rule

  $$
  (i - 1, A, i) \\
  \frac{}{\quad \{a = a_i\}}
  $$

  applied in the construction of $\mathcal{T}$, add to $G_w$ the production

  $$
  [i - 1, A, i] \rightarrow a
  $$

Grammar $G_w$ needs to be **reduced**
Example:
Assume \( P = \{ S \rightarrow S \ S, \ S \rightarrow a \} \) and \( w = aaa \)

The algorithm constructs the following productions for \( G_w \):

\[
\begin{align*}
[0, S, 3] & \rightarrow [0, S, 1] [1, S, 3] \\
[0, S, 3] & \rightarrow [0, S, 2] [2, S, 3] \\
[0, S, 2] & \rightarrow [0, S, 1] [1, S, 2] \\
[1, S, 3] & \rightarrow [1, S, 2] [2, S, 3] \\
[0, S, 1] & \rightarrow a \\
[1, S, 2] & \rightarrow a \\
[2, S, 3] & \rightarrow a
\end{align*}
\]
Example (cont’d) : graphical representation of $G_w$

```
[0, S, 3]
[0, S, 2]   [1, S, 3]
[0, S, 1]   [1, S, 2]   [2, S, 3]
           |   |   |
             a   a   a
```
Trees from $T(G, w)$ can be obtained by relabeling the trees generated by $G_w$, using the homomorphism $(A \in N)$:

$$h([j, A, i]) = A,$$

\[0, S, 3\]

\[0, S, 1\]

\[1, S, 2\]

\[1, S, 3\]

\[2, S, 3\]

\[0, S, 3\]

\[1, S, 3\]

\[1, S, 2\]

\[2, S, 3\]

\[a\]

\[a\]

\[h()\]
Grammar $G_w$ can be built *simultaneously* with the recognition process.

**Usual method**: for each item $I$ added to $T$, store the list $L(I)$ consisting of all pairs of items that have been used in the deduction of $I$ itself.
Example: consider production $A \rightarrow AS$

$$\mathcal{L}(A) = \langle (A \oplus S, 1), (A \oplus S, 2) \rangle$$
Grammar $G_w$ can be built in time and space $O(|G| \cdot |w|^3)$

Grammar $G_w$ can be reduced in time and space $O(|G_w|)$, using standard algorithms

Therefore the CKY parsing algorithm has the same asymptotic complexity as the CKY recognition algorithm
Extensions

For recognition algorithms other than CKY, the stack symbols are not associated with nonterminals of $G$.

In this general case, construction of $G_w$ is still possible and the set of trees generated by $G_w$ is a covering of the set $T(G, w)$.

This means that trees generated by $G_w$ can be mapped back to trees in $T(G, w)$ by a linear time process that collapses the intermediate nodes.
When **probabilistic or weighted extensions** of CFGs are used, the *K-best trees* in the parse forest $G_w$ can be individually extracted using dynamic programming.

One can also construct from $G_w$ a new parse forest, represented as a CFG $G_w^{(K)}$, that generates **the $K$-best trees** in $T(G, w)$. 

---

**Extensions**

When **probabilistic or weighted extensions** of CFGs are used, the *K-best trees* in the parse forest $G_w$ can be individually extracted using dynamic programming.

One can also construct from $G_w$ a new parse forest, represented as a CFG $G_w^{(K)}$, that generates **the $K$-best trees** in $T(G, w)$.
Y. Bar-Hillel, M. Perles, and E. Shamir.
1964.