Shared Packed Parse Forest (SPPF)

Fri, Jun 27, 2014 parsing, sppf

In the last decade there has been a lot of interest in generalized parsing techniques. These techniques can be used to generate a working parser for any context-free grammar. This means that we no longer have to massage our grammar to fit into restricted classes such as *LL(k)* or *LR(k)*. Supporting all context-free grammars means that grammars can be written in a natural way, and grammars can be combined, since the class of context-free grammars is closed under composition.

One of the consequences of supporting the whole class of context-free grammars is that also ambiguous grammars are supported. In an ambiguous grammar there are sentences in the language that can be derived in multiple ways. Each derivation results in a distinct parse tree. For each additional ambiguity in the input sentence, the number of derivations might grow exponentially. Therefore generalized parsers output a parse forest, rather than a set of the parse trees. In this parse forest, often sharing is used used to reduce the total space required to represent all derivation trees. Nodes which have the same subtree are shared, and nodes are combined which correspond to different derivations of the same substring. A parse forest where sharing is employed is called a shared packed parse forest (SPPF).

This article will describe the SPPF data structure in more detail. More information about the generation of the SPPF using the GLL algorithm can be found in the paper GLL parse-tree generation by E. Scott and A. Johnstone. Right Nulled GLR parsers can also be used to generate an SPPF, which is described in the paper Right Nulled GLR Parsers by E. Scott and A. Johnstone.

There are three types of nodes in an SPPF associated with a GLL parser: *symbol nodes*, *packed nodes*, and *intermediate nodes*. In the visualizations symbol nodes are shown as rectangles with rounded corners, packed nodes are shown as circles, or ovals when the label is visualized, and intermediate nodes are shown as rectangles.

Symbol nodes have labels of the form (x,j,i) where x is a terminal, nonterminal, or ε (i.e. $x \in T \cup N \cup \lbrace \varepsilon \rbrace$), and $0 \leq j \leq i \leq m$ with m being the length of the input sentence. The tuple (j,i) is called the *extent*, and denotes that the symbol x has been matched on the substring from position j up to position i. Here j is called the *left extent*, and i is called the *right extent*.

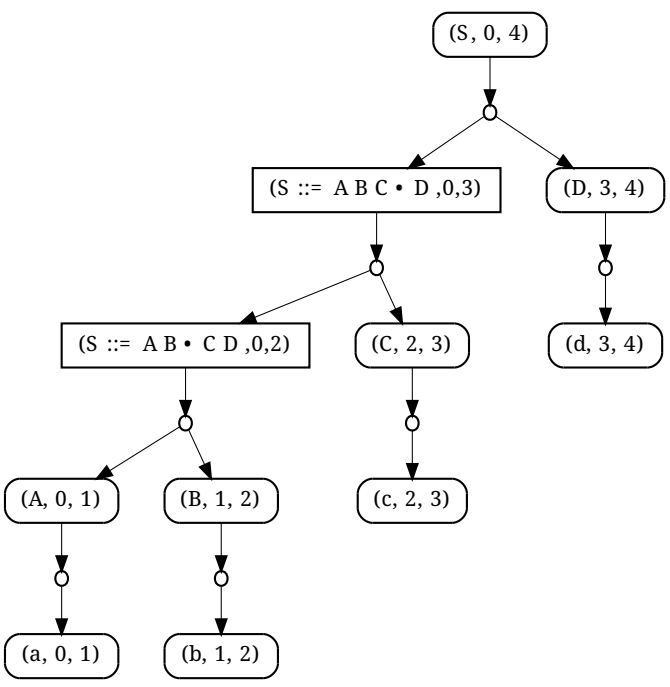
Packed nodes have labels of the form (t,k), where $0 \leq k \leq m$. Here k is called the *pivot*, and t is of the form $X ::= \alpha \cdot \beta$. The value of k represents that the last symbol of α ends at position k of the input string. Packed nodes are used to represent multiple derivation trees. When multiple derivations are possible with the same extent, starting from the same nonterminal symbol node, a separate packed node is added to the symbol node for each derivation.

Intermediate nodes are used to binarize the SPPF. They are introduced from the left, and group the children of packed nodes in pairs from the left. The binarization ensures that the size of the SPPF is worst-case cubic in the size of the input sentence. The fact that the SPPF is binarized does not mean that each node in the SPPF has at most two children. A symbol node or intermediate node can still have as many packed node children as there are ambiguities starting from it. Intermediate nodes have labels of the form (t,j,i) where t is a grammar slot, and (j,i) is the extent. There are no intermediate nodes of the shape $(A ::= \alpha \cdot, j,i)$, where the grammar pointer of the grammar slot is at the end of the alternate. These grammar slots are present in the form of symbol nodes.

Consider the following grammar:

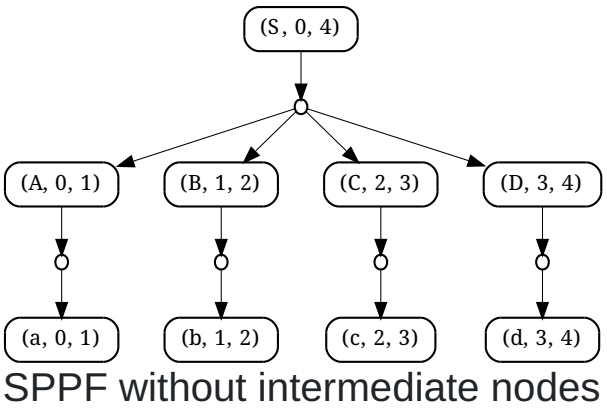
$\quad S ::= ABCD \quad A ::= a \quad B ::= b \quad C ::= c \quad D ::= d. $

Then given input sentence $abcd$, the the following SPPF will be the result:



SPPF with intermediate nodes

Suppose that the intermediate nodes had not been added to the SPPF. Then the nonterminal symbol nodes for A, B, C, and D would have been attached to the nonterminal symbol node S:



SPPF without intermediate nodes

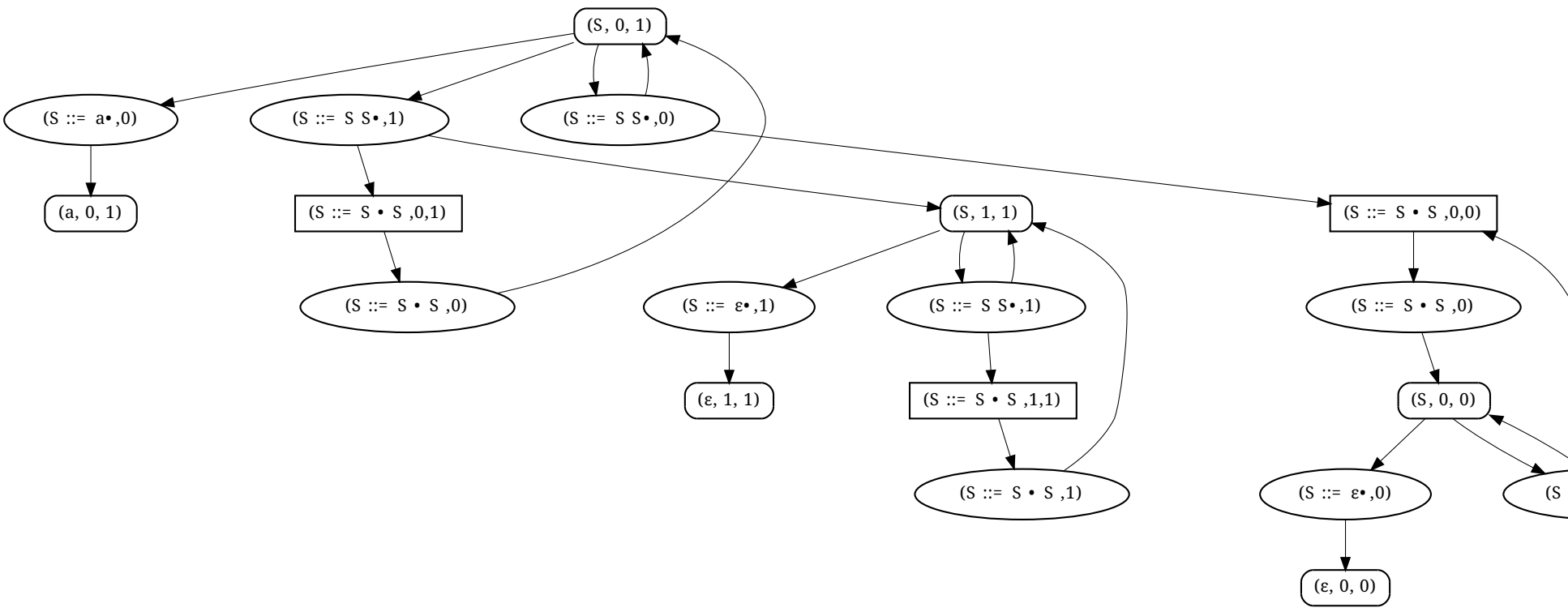
This example shows how intermediate nodes ensure that the tree is binarized.

Adding cycles

Grammars that contain cycles can generate sentences which have infinitely many derivation trees. A context-free grammar is cyclic if there exists a nonterminal $A \in N$ and a derivation $A \overset{+}\Rightarrow A$. Note that a cyclic context-free grammar implies that the context-free grammar is left-recursive, but the converse does not hold. The derivation trees for a cyclic grammar are represented in the finite SPPF by introducing cycles in the graph.

Consider the following cyclic grammar: $S ::= SS \mid a \mid \varepsilon$.

Given input sentence a, there are infinitely many derivations. All these derivations are present in the following SPPF:

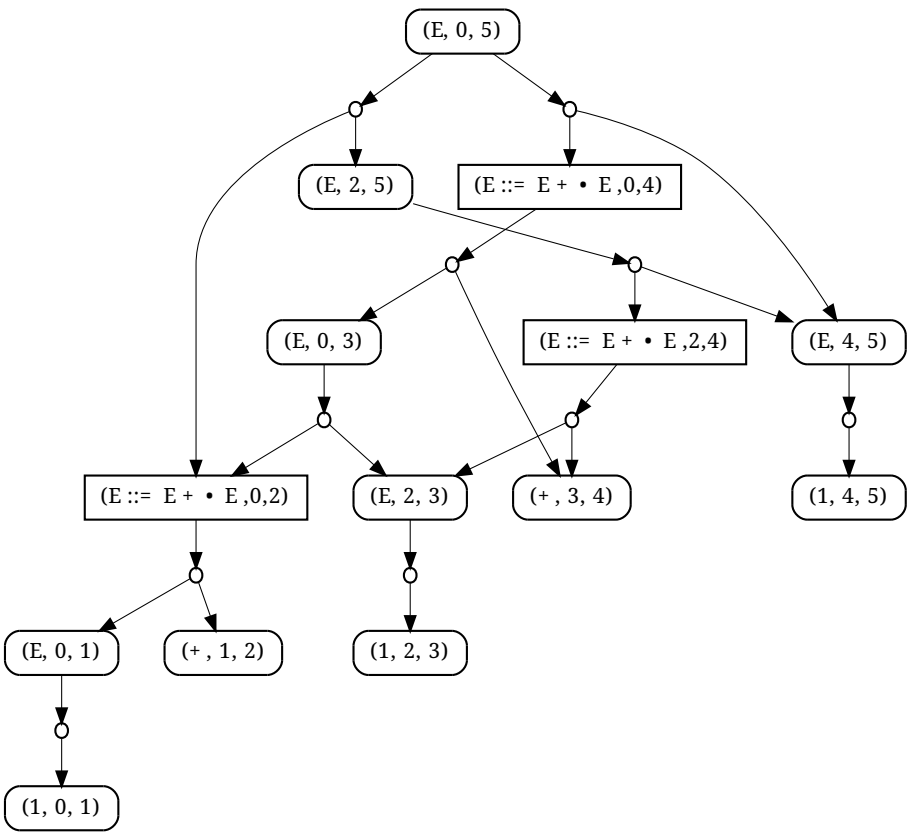


SPPF containing an infinite number of derivations

Ambiguities

A parse forest is *ambiguous* if and only if it contains at least one ambiguity. An ambiguity arises when a symbol node or intermediate node has at least two packed nodes as its children. Such nodes are called *ambiguous*. Consider for instance the following grammar with input sentence $1+1+1$: $ E ::= E + E \mid 1 $.

This gives the following SPPF:



SPPF containing an ambiguous root node

In this SPPF, symbol node $(E,0,5)$ has two packed nodes as children. This means that there are at least two different parse trees starting at this node, the parse trees representing derivations $(E+(E+E))$ and $((E+E)+E)$ respectively.

The set of all parse trees present in the SPPF is defined in the following way:

Start at the root node of the SPPF, and walk the tree by choosing one packed node below each visited node, and choosing all the children of a visited packed node in a recursive manner.

Structural Properties

There are various structural properties that are useful when reasoning about SPPFs in general. At first note that each symbol node (E,j,i) with $E \in T \cup N \cup \lbrace \varepsilon \rbrace$ is unique, so an SPPF does not contain two symbol nodes (A,k,l) and (B,m,n) with $A = B, k = m$, and $l=n$.

Terminal symbol nodes have no children. These nodes represent the leaves of the parse forest. Nonterminal symbol nodes (A,j,i) have packed node children of the form $(A ::= \gamma \cdot, k)$ with $j \leq k \leq i$, and the number of children is not limited to two.

Intermediate nodes (t,j,i) have packed node children with labels of the form (t,k), where $j \leq k \leq i$.

Packed nodes (t,k) have one or two children. The right child is a symbol node (x,k,i) and the left child (if it exists) is a symbol or intermediate node with label (s,j,k), where $j \leq k \leq i$. Packed nodes have always exactly one parent which is a symbol node or

intermediate node.

It is useful to observe that the SPPF is a bipartite graph, with on the one hand the set of intermediate and symbol nodes and on the other hand the set of packed nodes. Therefore edges always go from a node of the first type to a node of the second type, or the other way round. As a consequence, cyles in the SPPF are always of even length.

Transformation to an abstract syntax tree

In the end, we often want a single abstract syntax tree (AST) when parsing an input sentence. In order to arrive at this AST, we need disambiguation techniques to remove undesired parse trees from the SPPF or avoid the generation of undesired parse trees in the first place. {% cite sanden2014thesis %} describes several SPPF disambiguation filters that remove ambiguities arising in expression grammars. Furthermore a method is described to integrate parse-time filtering in GLL that tries to avoid embedding undesired parse trees in the SPPF.

Of course, other transformation might be needed such as the removal of whitespace and comments from the parse forest.

- Compositional Specification of Functionality and Timing of Manufacturing Systems →