

# Parsing

## Tomita's Parser: Generalized LR Parsing

Laura Kallmeyer

Heinrich-Heine-Universität Düsseldorf

Summer 2021



# Table of contents

- 1 Motivation
- 2 Graph-structured stack
- 3 The parse forest
- 4 Conclusion

# Motivation

- LR-parsing with one lookahead is deterministic for LR(1) grammars. But there are CFLs that cannot be generated by LR(1)-grammars.
- If a grammar is not LR(1), we can construct a LR(1) parse table with more than one entry in some of the fields. This can be used for non-deterministic parsing.
- However, since we don't have tabulation, partial results get computed several times and the complexity is exponential.
- Tomita's idea: Use a graph-structured stack to avoid computing partial results more than once.

Tomita's parser is an **LR parser with tabulation**

# Graph-structured stack (1)

The stack is a directed acyclic graph (DAG) with the leaves being the topmost elements.

A directed acyclic graph consists of

- A set of **nodes** (or **vertices**)  $V$  (here finite), and
- a set of **edges**  $E \subset V \times V$ , such that
  - a) for all  $v \in V$ :  $\langle v, v \rangle \notin E$ , and
  - b) for every sequence  $v_1, \dots, v_k \in V$  with  $\langle v_1, v_2 \rangle, \dots, \langle v_{k-1}, v_k \rangle \in E$ :  $v_1 \neq v_k$ .

In our case, the vertices of the DAG are labelled with states, non-terminals or terminals.

## Graph-structured stack (2)

Our parsing is incremental, i.e., processes the input one by one from left to right.

For every word in the input, before processing that word, we have  $k$  possible states.

- We first do the possible reductions for each of the states while leaving the original stack if there is a shift possible. In case of a reduce/reduce or shift/reduce conflict, we branch. If several branches lead to the same states, we identify these. We repeat this until no more reductions are possible.
- We then do the possible shifts. Again, if several lead to the same states, we identify these.

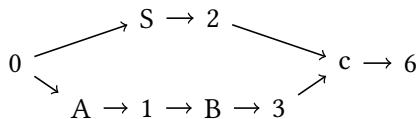
# Graph-structured stack (3)

Example: 1.  $S \rightarrow AB$ , 2.  $S \rightarrow SC$ , 3.  $B \rightarrow BC$ ,  
4.  $A \rightarrow a$ , 5.  $B \rightarrow b$ , 6.  $C \rightarrow c$

	$a$	$b$	$c$	$\$$	$A$	$B$	$C$	$S$
	s4				1			2
		s5				3		
			s6	acc			7	
Parse			r1, s6	r1			8	
table:		r4						
			r5	r5				
			r6	r6				
			r2	r2				
			r3	r3				

## Graph-structured stack (4)

For an input  $w = abcc$ , at some point (after shifting the first  $c$ ) the stack is the following:



## Graph-structured stack (5)

Problems (infinite loops) in generalized LR parsing can arise from

- Loops: Productions  $A \rightarrow B, B \rightarrow A$  would lead to an infinite reduce-loop.
- Hidden left-recursion:  $A \rightarrow \alpha A \beta$  with  $\alpha \xRightarrow{*} \epsilon$  would lead to an infinite loop of reducing  $\epsilon$  to  $\alpha$  since  $A \rightarrow \alpha \bullet A \beta$  and  $A \rightarrow \bullet \alpha A \beta$  would be in the same state.



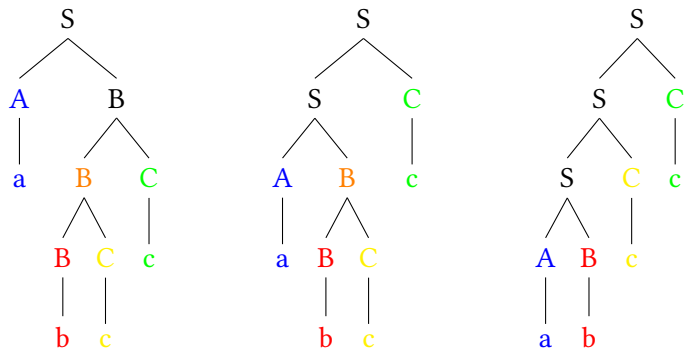
# The parse forest (1)

- The dag-structure avoids an explosion in the number of stacks.
- However, we can still have exponentially many parse trees for a given input.
- Therefore, a compact representation of parse forests is needed.
- Tomita uses two techniques: [sub-tree sharing](#) and [local ambiguity packing](#).

## The parse forest (2)

Example: Take the preceding grammar,  $w = abcc$

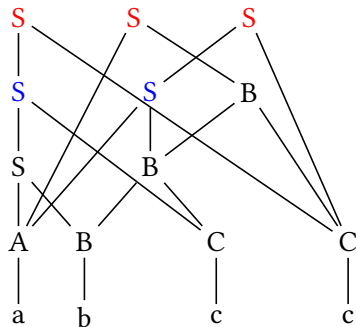
Three parse trees:



Sub-tree sharing: Common sub-trees are represented only once.

## The parse forest (3)

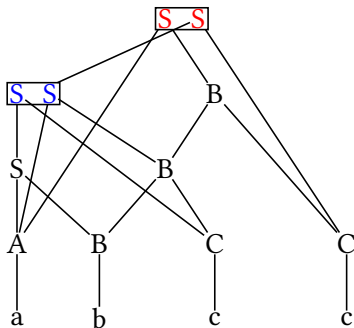
Result of sub-tree sharing:



Local ambiguity packing: whenever the same category spans the same input (possibly with different analyses), the corresponding nodes are put into one **packed node**.

# The parse forest (4)

Result of local ambiguity packing:



## The parse forest (5)

Packed parse forests are easy to construct within an LR-parser with graph-structured stack: Whenever a subtree is shared or different subtrees are packed into one node, there will be a corresponding shared node in the stack graph. More precisely,

- Whenever a node is shared, we create a shared sub-tree, and
- whenever two or more branches get identified into a single new branch, we create a packed node.

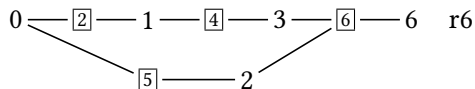
Instead of non-terminals or terminals we use pointers to/identifiers of parse trees as stack vertex labels. This way, in different places we can have pointers to the same parse tree.

# The parse forest (6)

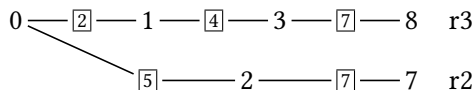
Example:  $w = abc$ .

Stack	analysis
0 s4	
0 — <span style="border: 1px solid black; padding: 0 2px;">1</span> — 4 r4	<span style="border: 1px solid black; padding: 0 2px;">1</span> : a
0 — <span style="border: 1px solid black; padding: 0 2px;">2</span> — 1 s5	<span style="border: 1px solid black; padding: 0 2px;">2</span> : A( <span style="border: 1px solid black; padding: 0 2px;">1</span> )
0 — <span style="border: 1px solid black; padding: 0 2px;">2</span> — 1 — <span style="border: 1px solid black; padding: 0 2px;">3</span> — 5 r5	<span style="border: 1px solid black; padding: 0 2px;">3</span> : b
0 — <span style="border: 1px solid black; padding: 0 2px;">2</span> — 1 — <span style="border: 1px solid black; padding: 0 2px;">4</span> — 3 r1, s6	<span style="border: 1px solid black; padding: 0 2px;">4</span> : B( <span style="border: 1px solid black; padding: 0 2px;">3</span> )
0 — <span style="border: 1px solid black; padding: 0 2px;">2</span> — 1 — <span style="border: 1px solid black; padding: 0 2px;">4</span> — 3 s6	
<span style="border: 1px solid black; padding: 0 2px;">5</span> — 2 s6	<span style="border: 1px solid black; padding: 0 2px;">5</span> : S( <span style="border: 1px solid black; padding: 0 2px;">2</span> , <span style="border: 1px solid black; padding: 0 2px;">4</span> )

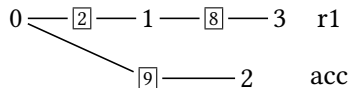
# The parse forest (7)



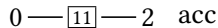
[6]: c



[7]: C([6])



[8]: B([4],[7]), [9]: S([5], [7])



[10]: S([2],[8]), [11]: [[10], [9]]

# Conclusion

## Tomita's algorithm

- is a general LR(1) parser that works for every CFG;
- uses a graph-structured stack to avoid the explosion otherwise linked to non-determinism;
- uses a compact parse forest representation to avoid the explosion arising from ambiguous grammars.

## Reference:

Masaru Tomita (1987) An Efficient Augmented-Context-Free Parsing Algorithm *Computational Linguistics* 13(1-2), 1987.