

Grundlegende Parsingalgorithmen

Tomita-Parser

Kurt Eberle

k.eberle@lingenio.de

(Viele Folien, Teile von Folien, Materialien von **Helmut Schmid's**
Parsing-Kurs WS14 Tübingen, u.a.)

26. Februar, 2020

Überblick

Idee und Hintergrund

Daten-Strukturen

Tomita Erkennen

Tomita-Erkennen: Operationen

Parse-Wald

Überblick

Idee und Hintergrund

Daten-Strukturen

Tomita Erkennen

Tomita-Erkennen: Operationen

Parse-Wald

Tomita Parser

- ▶ *An efficient all-paths parsing algorithm for natural languages*
(Masaru Tomita, 1984)

Tomita Parser

- ▶ *An efficient all-paths parsing algorithm for natural languages* (Masaru Tomita, 1984)
- ▶ Other name: GLR parser (generalized LR)

Tomita Parser

- ▶ *An efficient all-paths parsing algorithm for natural languages* (Masaru Tomita, 1984)
- ▶ Other name: GLR parser (generalized LR)
- ▶ Problem: Many grammars are not in LR(k). How can we nevertheless use an LR parser?

Tomita Parser

- ▶ *An efficient all-paths parsing algorithm for natural languages* (Masaru Tomita, 1984)
- ▶ Other name: GLR parser (generalized LR)
- ▶ Problem: Many grammars are not in LR(k). How can we nevertheless use an LR parser?
- ▶ Idea 1: Backtracking

Tomita Parser

- ▶ *An efficient all-paths parsing algorithm for natural languages* (Masaru Tomita, 1984)
- ▶ Other name: GLR parser (generalized LR)
- ▶ Problem: Many grammars are not in LR(k). How can we nevertheless use an LR parser?
- ▶ Idea 1: Backtracking
- ▶ Idea 2: Parallel Simulation of multiple LR parsers
Non-determinism is handled by

Tomita Parser

- ▶ *An efficient all-paths parsing algorithm for natural languages* (Masaru Tomita, 1984)
- ▶ Other name: GLR parser (generalized LR)
- ▶ Problem: Many grammars are not in LR(k). How can we nevertheless use an LR parser?
- ▶ Idea 1: Backtracking
- ▶ Idea 2: Parallel Simulation of multiple LR parsers
Non-determinism is handled by
 - ▶ Reduplication of the parser

Tomita Parser

- ▶ *An efficient all-paths parsing algorithm for natural languages* (Masaru Tomita, 1984)
- ▶ Other name: GLR parser (generalized LR)
- ▶ Problem: Many grammars are not in LR(k). How can we nevertheless use an LR parser?
- ▶ Idea 1: Backtracking
- ▶ Idea 2: Parallel Simulation of multiple LR parsers
Non-determinism is handled by
 - ▶ Reduplication of the parser
 - ▶ Tree-structured stack (for left-hand side of the configuration)

Tomita Parser

- ▶ *An efficient all-paths parsing algorithm for natural languages* (Masaru Tomita, 1984)
- ▶ Other name: GLR parser (generalized LR)
- ▶ Problem: Many grammars are not in LR(k). How can we nevertheless use an LR parser?
- ▶ Idea 1: Backtracking
- ▶ Idea 2: Parallel Simulation of multiple LR parsers
Non-determinism is handled by
 - ▶ Reduplication of the parser
 - ▶ Tree-structured stack (for left-hand side of the configuration)
 - ▶ Graph-structured stack

Tomita Parser

- ▶ *An efficient all-paths parsing algorithm for natural languages* (Masaru Tomita, 1984)
- ▶ Other name: GLR parser (generalized LR)
- ▶ Problem: Many grammars are not in LR(k). How can we nevertheless use an LR parser?
- ▶ Idea 1: Backtracking
- ▶ Idea 2: Parallel Simulation of multiple LR parsers
Non-determinism is handled by
 - ▶ Reduplication of the parser
 - ▶ Tree-structured stack (for left-hand side of the configuration)
 - ▶ Graph-structured stack
- ▶ Graph-structured stack:

Tomita Parser

- ▶ *An efficient all-paths parsing algorithm for natural languages* (Masaru Tomita, 1984)
- ▶ Other name: GLR parser (generalized LR)
- ▶ Problem: Many grammars are not in LR(k). How can we nevertheless use an LR parser?
- ▶ Idea 1: Backtracking
- ▶ Idea 2: Parallel Simulation of multiple LR parsers
Non-determinism is handled by
 - ▶ Reduplication of the parser
 - ▶ Tree-structured stack (for left-hand side of the configuration)
 - ▶ Graph-structured stack
- ▶ Graph-structured stack:
 - ▶ Directed, acyclic graph with a single source node

Tomita Parser

- ▶ *An efficient all-paths parsing algorithm for natural languages* (Masaru Tomita, 1984)
- ▶ Other name: GLR parser (generalized LR)
- ▶ Problem: Many grammars are not in LR(k). How can we nevertheless use an LR parser?
- ▶ Idea 1: Backtracking
- ▶ Idea 2: Parallel Simulation of multiple LR parsers
Non-determinism is handled by
 - ▶ Reduplication of the parser
 - ▶ Tree-structured stack (for left-hand side of the configuration)
 - ▶ Graph-structured stack
- ▶ Graph-structured stack:
 - ▶ Directed, acyclic graph with a single source node
 - ▶ Nodes correspond to parser configurations.

Tomita Parser

- ▶ *An efficient all-paths parsing algorithm for natural languages* (Masaru Tomita, 1984)
- ▶ Other name: GLR parser (generalized LR)
- ▶ Problem: Many grammars are not in LR(k). How can we nevertheless use an LR parser?
- ▶ Idea 1: Backtracking
- ▶ Idea 2: Parallel Simulation of multiple LR parsers
Non-determinism is handled by
 - ▶ Reduplication of the parser
 - ▶ Tree-structured stack (for left-hand side of the configuration)
 - ▶ Graph-structured stack
- ▶ Graph-structured stack:
 - ▶ Directed, acyclic graph with a single source node
 - ▶ Nodes correspond to parser configurations.
 - ▶ The end nodes are the active configurations.

Eine ambige Grammatik

PP Attachment? S oder NP

```
-----  
(1)  S  --> NP VP  
(2)  S  --> S  PP  
(3)  NP --> *n  
(4)  NP --> *det *n  
(5)  NP --> NP PP  
(6)  PP --> *prep NP  
(7)  VP --> *v  NP  
-----
```

Figure 2-5: An Example Ambiguous Grammar

Tabelle der Grammatik

PP Attachment? Shift oder Reduce

Figure 2-5: An Example Ambiguous Grammar

State	*det	*n	*v	*prep	\$	NP	PP	VP	S
0	sh3	sh4				2			1
1				sh6	acc		5		
2			sh7	sh6			9	8	
3		sh10							
4			re3	re3	re3				
5				re2	re2				
6	sh3	sh4				11			
7	sh3	sh4				12			
8				re1	re1				
9			re5	re5	re5				
10			re4	re4	re4				
11			re6	re6, sh6	re6			9	
12				re7, sh6	re7			9	

Figure 2-6: LR Parsing Table with Multiple Entries

Stack strukturiert als Graph

I saw a man on the bed in the apartment **with** a telescope

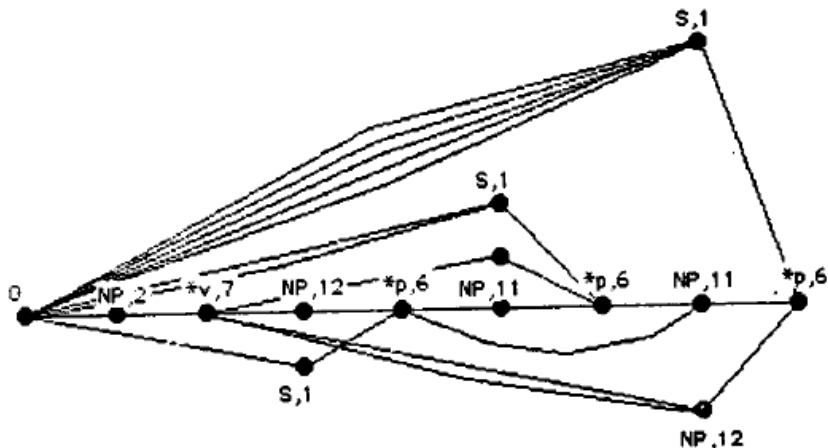


Figure 3-3: A Graph-Structured Stack

Überblick

Idee und Hintergrund

Daten-Strukturen

Tomita Erkennen

Tomita-Erkennen: Operationen

Parse-Wald

Tomita Erkennen: Daten-Strukturen

- ▶ $\{a, b, c\}$: Set containing the elements a, b and c .
- ▶ $[a, b, c]$: List containing the elements a, b and c .
- ▶ $\circ$: Concatenation operator for lists
- ▶ U_i : Set of top-most stack nodes after i input symbols have been processed
- ▶ $\langle i, s, l \rangle$: a node of the graph-structured stack
- ▶ $\langle 0, s_0, \{\} \rangle$: Root of the stack
- ▶ $Actions(s, x)$: Set of actions in state s with look-ahead symbol x
- ▶ $Goto(s, D)$: New state after the non-terminal D has been consumed in state s

Überblick

Idee und Hintergrund

Daten-Strukturen

Tomita Erkenner

Tomita-Erkenner: Operationen

Parse-Wald

Tomita Erkennen

Recognizer($x_1 \dots x_n$)

```

 $x_{n+1} := \$$                                 %%Add end symbol to input string
 $U_0 := [\langle 0, S_0, \{\} \rangle]$                 %%initial stack element with start configuration
For  $i := 1$  to  $n + 1$ 
     $U_i := []$                                 %%position- $i$  stack elements
     $P := []$                                 %%nodes already considered
    For all  $v = \langle i - 1, s, l \rangle$  such that  $v \in U_{i-1}$  from left to right
         $P := P \circ [v]$ 
        For all  $a \in \text{Actions}(s, x_i)$ 
            If  $a = \text{shift } s'$  then
                Shift( $v, s', i$ )
            else if  $a = \text{reduce } p$  then
                Reduce( $v, p, i$ )
            else if  $a = \text{accept}$  then
                accept
    If  $U_i = []$  then
        Report error
Accept

```


Überblick

Idee und Hintergrund

Daten-Strukturen

Tomita Erkennen

Tomita-Erkennen: Operationen

Parse-Wald

Tomita-Erkennen: Shift

Shift(v, s, i)

If $\exists v' = \langle i, s, l \rangle$ such that $v' \in U_i$ then

$l := l \cup \{v\}$

else

$U_i := U_i \circ [\langle i, s, \{v\} \rangle]$

Tomita-Erkennen: Predecessor

Predecessor($v = \langle j, s, l \rangle, k$)

If $k = 0$ then

 return $\{v\}$

else

 return $\cup_{v' \in l} \text{predecessor}(v', k - 1)$

Tomita-Erkennen: Reduce

Reduce(v, p, i)

For all $v'_1 = \langle j', s', l'_1 \rangle$ such that $v'_1 \in \text{Predecessor}(v, |p|)$

$s'' := \text{Goto}(s', D_p)$

If $\exists v'' = \langle i-1, s'', l'' \rangle$ such that $v'' \in U_{i-1}$

then If $v'_1 \in l''$

then do nothing

else if $\exists v'_2 = \langle j', s', l'_2 \rangle$ such that $v'_2 \in l''$

then $v''_c := \langle i-1, s'', \{v'_1\} \rangle$

For all 'reduce $p' \in \text{Actions}(s'', x_i)$

$\text{Reduce}(v''_c, p, i)$

else $l'' := l'' \cup \{v'_1\}$

If $v'' \in P$ then $v''_c := \langle i-1, s'', \{v'_1\} \rangle$

For all 'reduce $p' \in \text{Actions}(s'', x_i)$

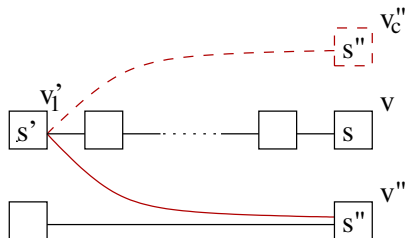
$\text{Reduce}(v''_c, p, i)$

else

$U_{i-1} := U_{i-1} \circ [\langle i-1, s'', \{v'_1\} \rangle]$

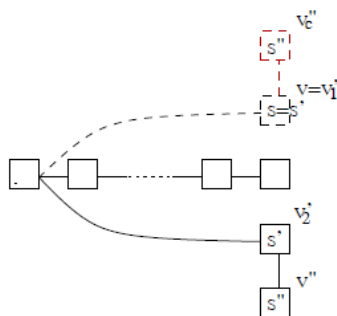
Tomita-Erkennen: Kopien

no s' predecessor yet included, v'' already in $P \dots$



Tomita-Erkennen: Kopien

s' predecessor already included (v'' already in P) ...



Beispiel: Stacks teilen und zusammenführen

 $F \rightarrow D E$

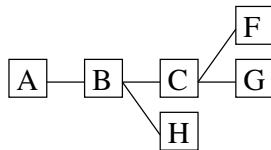
Auszug aus einer Grammatik ... $G \rightarrow D E$

 $H \rightarrow C D E$

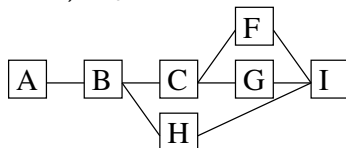
Parsing-Situation...



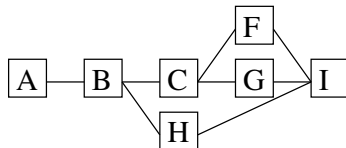
Drei Reduktionen...



Shift wird auf alle (offenen) Top-Elemente des Stacks angewendet



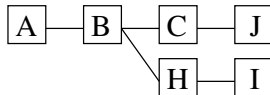
Beispiel: Stacks teilen und zusammenführen



Lokale Ambiguität. . .

$J \rightarrow F I$

$J \rightarrow G I$



Beispiel

(Grammatik und Zustände von "ABgram.pdf")

2	a	U2	[$\langle 2, s3, \{v12\} \rangle_{Aa}, \langle 2, s5, \{v13\} \rangle_{Ba}, \langle 2, s6, \{v12\} \rangle_{AB},$ $\langle 2, s7, \{v13\} \rangle_{BA}, \langle 2, s8, \{v0\} \rangle_S]$
1	a	U1	[$v11\langle 1, s1, \{v0\} \rangle_a, v12\langle 1, s2, \{v0\} \rangle_A, v13\langle 1, s4, \{v0\} \rangle_B]$
0	0	U0	[$v0\langle 0, s0, \{\} \rangle]$

Kontroll-Tabelle Beispiel

Grammar

1: $S \rightarrow NP VP$

2: $VP \rightarrow VP PP$

3: $VP \rightarrow v NP$

4: $PP \rightarrow p NP$

5: $NP \rightarrow NP PP$

6: $NP \rightarrow d N1$

7: $NP \rightarrow N1$

8: $N1 \rightarrow a N1$

9: $N1 \rightarrow n$

Kontroll-Tabelle Beispiel

Table

	d	a	n	p	v	\$	N1	NP	PP	VP	S
0	s9	s5	s4				8	2			1
1						acc					
2				s12	s14				3	10	
3				r5	r5	r5					
4				r9	r9	r9					
5		s5	s4				6				
6				r8	r8	r8					
7				r6	r6	r6					
8				r7	r7	r7					
9		s5	s4				7				
10				s12		r1			11		
11				r2		r2					
12	s9	s5	s4				8	13			
13				r4/s12	r4	r4			3		
14	s9	s5	s4				8	15			
15				r3/s12		r3			3		

Überblick

Idee und Hintergrund

Daten-Strukturen

Tomita Erkennen

Tomita-Erkennen: Operationen

Parse-Wald

Parse-Wald

- ▶ Problem: Sätze können exponentiell viele Analysen haben
⇒ Output der Analysen hat exponentielle Laufzeit
- ▶ Lösung: Kompakte Repräsentation in der Form eines *Parse-Waldes*
- ▶ Parse-Wald:
 - ▶ Gerichteter Graph mit eindeutigem Wurzelknoten
 - ▶ Teilbäume werden in verschiedenen Analysen benutzt (⇒ Graph)
 - ▶ Parse-Bäume die sich nur in einem einzigen Teilbaum (über demselben Teilstring) unterscheiden werden zusammengefasst und die Top-Knoten der Teilbäume werden in einem *gepackten Knoten* (*packed node*) zusammengeführt.
- ▶ Extraktion der individuellen Analysen mit Backtracking

Parse-Wald

with packed nodes and shared nodes

