Computational Linguistics II: Parsing

Left-corner-Parsing

Frank Richter & Jan-Philipp Söhn

fr@sfs.uni-tuebingen.de, jp.soehn@uni-tuebingen.de

January 15th, 2007

Overview (Modified!)

	Top-down	Bottom-up
Non-directional methods	Unger parser	CYK parser
Directional methods	predict/match Depth-first (backtrack) Breadth-first, DCGs	shift/reduce Depth-first (backtrack) Breadth-first (Earley)
	Left-corner	
Linear directional methods		LR(k) SLR(1)
Efficient general directional methods		Tomita

The idea behind left-corner-parsing

Left corner: the leftmost symbol on the right of a rule:
$k_0 \longrightarrow k_1\ k_2\ \ldots\ k_n$

- Normal bottom-up: all k_1 to k_n must be recognized for applying the rule
- Left-corner: it suffices that k_1 is recognized
- k_2 to k_n and the dominating nodes of k_1 are predicted in a top-down fashion

Left-corner combines bottom-up and top-down strategies.

The order of nodes

- If a node n immediately dominates the nodes $n_1, \ldots, n_m$, then all nodes below n_1 precede n
- n precedes all other nodes dominated by n
- all nodes dominated by n_i precede all nodes dominated by n_{i+1}

Infix notation!!

An example

Assume the following grammar:

1: S $\longrightarrow$ AS	2: S $\longrightarrow$ BB	3: A $\longrightarrow$ bAA
4: A $\longrightarrow$ a	5: B $\longrightarrow$ b	6: B $\longrightarrow$ c

Sentence: bbaaacc

Differences between left-corner and Earley

- Both algorithms combine TD and BU. However, left-corner parsing starts with a BU step and continues TD, Earley proceeds the other way round.
- Left-corner parsing is originally stack-based, Earley parsing is chart-based.
- Both algorithms use a way to store categories which are to be completed. Left-corner uses an additional stack, Earley uses active arcs in the chart.

Differences between left-corner and Earley

- Both algorithms combine TD and BU. However, left-corner parsing starts with a BU step and continues TD, Earley proceeds the other way round.
- Left-corner parsing is originally stack-based, Earley parsing is chart-based.
- Both algorithms use a way to store categories which are to be completed. Left-corner uses an additional stack, Earley uses active arcs in the chart.

Differences between left-corner and Earley

- Both algorithms combine TD and BU. However, left-corner parsing starts with a BU step and continues TD, Earley proceeds the other way round.
- Left-corner parsing is originally stack-based, Earley parsing is chart-based.
- Both algorithms use a way to store categories which are to be completed. Left-corner uses an additional stack, Earley uses active arcs in the chart.

Parsing strategies summarized

Assume the following grammar:

1: S $\longrightarrow$ NP VP   2: NP $\longrightarrow$ n   3: NP $\longrightarrow$ d n   4: VP $\longrightarrow$ v NP

5: n $\longrightarrow$ John
6: n $\longrightarrow$ apple
7: d $\longrightarrow$ an
8: v $\longrightarrow$ eats

Parsing strategies: Top-down parsing

input	*derivation*	*pos.*	*action*	*rule*
John eats an apple	S	1		
John eats an apple	NP VP	1	predict	1
John eats an apple	n VP	1	predict	2
John eats an apple	n VP	2	match	5

Parsing strategies: Top-down parsing

input	*derivation*	*pos.*	*action*	*rule*
John eats an apple	S	1		
John eats an apple	NP VP	1	predict	1
John eats an apple	n VP	1	predict	2
John eats an apple	n VP	2	match	5
eats an apple	v NP	2	predict	4
eats an apple	v NP	3	match	8
an apple	n	3	predict	2
an apple	n	3	ERROR	
an apple	NP	3	backtrack	
an apple	d n	3	predict	3
an apple	d n	4	match	7
apple	n	5	match	6

Parsing strategies: Bottom-up parsing

input	*derivation-stack*	*pos.*	*action*	*rule*
John eats an apple	–	1		
eats an apple	John	2	shift	
eats an apple	n	2	reduce	5
eats an apple	NP	2	reduce	2

Parsing strategies: Bottom-up parsing

input	*derivation-stack*	*pos.*	*action*	*rule*
John eats an apple	–	1		
eats an apple	John	2	shift	
eats an apple	n	2	reduce	5
eats an apple	NP	2	reduce	2
an apple	NP eats	3	shift	
an apple	NP v	3	reduce	8
apple	NP v an	4	shift	
apple	NP v d	4	reduce	7
–	NP v d apple	5	shift	
–	NP v d n	5	reduce	6
–	NP v NP	5	reduce	3
–	NP VP	5	reduce	4
–	S	5	reduce	1

Parsing strategies: Left-corner parsing

input	*categories*	*constituents*	*action*	*rule*
John eats an apple	S	–		
eats an apple	S t	n	reduce	5
n eats an apple	S	–	move	
n eats an apple	S t	NP	reduce	2
NP eats an apple	S	–	move	
eats an apple	S t VP	S	reduce	1
an apple	S t VP t	S v	reduce	8
v an apple	S t VP	S	move	
an apple	S t VP t NP	S VP	reduce	4
apple	S t VP t NP t	S VP d	reduce	7
d apple	S t VP t NP	S VP	move	

Parsing strategies: Left-corner parsing

input	*categories*	*constituents*	*action*	*rule*
John eats an apple	S	–		
eats an apple	S t	n	reduce	5
n eats an apple	S	–	move	
n eats an apple	S t	NP	reduce	2
NP eats an apple	S	–	move	
eats an apple	S t VP	S	reduce	1
an apple	S t VP t	S v	reduce	8
v an apple	S t VP	S	move	
an apple	S t VP t NP	S VP	reduce	4
apple	S t VP t NP t	S VP d	reduce	7
d apple	S t VP t NP	S VP	move	
apple	S t VP t NP t n	S VP NP	reduce	3
–	S t VP t NP t n t	S VP NP n	reduce	6
n	S t VP t NP t n	S VP NP	move	
–	S t VP t NP t	S VP NP	remove	
NP	S t VP t NP	S VP	move	
–	S t VP t	S VP	remove	
VP	S t VP	S	move	
–	S t	S	remove	
S	S	–	move	
–	–	–	remove	

Problems of left-corner parsing

- Ambiguity (identical left corners in several rules) $\longrightarrow$ look-ahead!
- Left recursion
- ϵ-rules

Problems of left-corner parsing

- Ambiguity (identical left corners in several rules) $\longrightarrow$ look-ahead!
- Left recursion
- ϵ-rules

Problems of left-corner parsing

- Ambiguity (identical left corners in several rules) $\longrightarrow$ look-ahead!
- Left recursion
- ϵ-rules

Left recursion

⇒ No problem for left-corner parser with look-ahead

Example 1 (direct recursion):

1: S → S b   2: S → a

yield: a b*

Example 2 (indirect recursion):

1: S → A d   2: A → B C   3: A → a

4: B → A   5: B → b   6: C → c

yield: (a|b) c* d

Direct left recursion

input	*categories*	*constituents*	*action*	*rule*
a b b b	S	–		
b b b	S t	S	reduce	2
S b b b	S	–	move	
b b b	S t b	S	reduce	1
b b	S t	S	remove	
S b b	S	–	move	
b b	S t b	S	reduce	1
b	S t	S	remove	
S b	S	–	move	
b	S t b	S	reduce	1
–	S t	S	remove	
S	S	–	move	
–	–	–	remove	
–	S t b	S	reduce	1

Indirect left recursion

input	*categories*	*constituents*	*action*	*rule*
b c c c d	S	–		
c c c d	S t	B	reduce	5
B c c c d	S	–	move	
c c c d	S t C	A	reduce	2
c c d	S t C t	A C	reduce	6
C c c d	S t C	A	move	
c c d	S t	A	remove	
A c c d	S	–	move	
c c d	S t	B	reduce	4
B c c d	S	–	move	
c c d	S t C	A	reduce	2
c d	S t C t	A C	reduce	6
C c d	S t C	A	move	
c d	S t	A	remove	
A c d	S	–	move	
c d	S t	B	reduce	4
B c d	S	–	move	
c d	S t C	A	reduce	2
d	S t C t	A	reduce	6
C d	S t C	A	move	
d	S t	A	remove	
A d	S	–	move	
d	S t d	S	reduce	1
–	S t	S	remove	
S	S	–	move	
–	–	–	remove	

ϵ-rules

Two different kinds of ϵ-rules:

1. VP $\rightarrow$ V Adv   Adv $\rightarrow$ ϵ *A man sleeps (quietly).*
2. NP $\rightarrow$ Det N   Det $\rightarrow$ ϵ *(The) men sleep.*

Left corner of the NP rule might be empty. Problem: Left corners are parsed bottom-up.
Empty productions at other places are parsed top-down, therefore there is no problem.

ϵ-rules

Two different kinds of ϵ-rules:

1. VP $\rightarrow$ V Adv   Adv $\rightarrow \epsilon$ *A man sleeps (quietly).*
2. NP $\rightarrow$ Det N   Det $\rightarrow \epsilon$ *(The) men sleep.*

Left corner of the NP rule might be empty. Problem: Left corners are parsed bottom-up.
Empty productions at other places are parsed top-down, therefore there is no problem.

ϵ-rules II

Idea: predefine left corners in a relation `link`:

```
link(np, s)
link(det, np)
link(det, s) (transitivity)
link(v, vp)
...
```

ϵ-rules III

This prevents an erroneous application of an ϵ-rule.

`lcp(C, [Word|Rest]-RestDiff) :-` #Left-corner parse (symbol, input)
`word(Word, LC),` #Take a word from input and get its category LC
`complete(LC, C, Rest-RestDiff).` #Complete LC to a constituent using the rest of the input (we won't define `complete` here)

`lcp(C, S-Rest) :-` #Left-corner parse (symbol, input) with ϵ-rules
`rule(LHS, []),` #a rule LHS $\rightarrow \epsilon$
`link(LHS, C),` #LHS and C must be in a `link` relation
`complete(LHS, C, S-Rest).` #complete LHS to a constituent using the rest of the input

ϵ-rules IV

Problem solved? Yes and No!

- Rules of type 2 are handled with `link`.

```
lcp(np, [det,n]) :-
rule(det, []),
link(det, np),
complete...
```

- Rules of type 1 are not! Why? C and LHS are identical.

```
lcp(adv, S-Rest) :-
rule(adv, []),
link(adv, adv),
complete...
```

Therefore, `link` must be reflexive.

ϵ-rules IV

Problem solved? Yes and No!

- Rules of type 2 are handled with `link`.

```
lcp(np, [det,n]) :-
rule(det, []),
link(det, np),
complete...
```

- Rules of type 1 are not! Why? C and LHS are identical.

```
lcp(adv, S-Rest) :-
rule(adv, []),
link(adv, adv),
complete...
```

Therefore, `link` must be reflexive.

Summary and open questions

- Summary:
  With the correct definition of `link` (transitive and reflexive closure) ϵ-rules are not a problem any more.
- Open questions:
  What about similar rules? (VP $\rightarrow$ V NP PP   VP $\rightarrow$ V NP NP)
  Use a chart to store intermediate results!

Summary and open questions

- Summary:
  With the correct definition of `link` (transitive and reflexive closure) ϵ-rules are not a problem any more.
- Open questions:
  What about similar rules? (VP $\rightarrow$ V NP PP   VP $\rightarrow$ V NP NP)
  Use a chart to store intermediate results!