Special Topics in Computer Science

NLP in a Nutshell

CS492B Spring Semester 2009

Jong C. Park

Computer Science Department

Korea Advanced Institute of Science and Technology

PARSING TECHNIQUES

Parsing Techniques

- Introduction
- Bottom-up Parsing
- Chart Parsing
- Probabilistic Parsing of Context-Free Grammars
- A Description of PCFGs
- Parser Evaluation
- Parsing Dependencies

Chart Parsing

Table 11.3. Some dotted-rules and arcs in the chart while parsing *Bring the meal of the day.*

Position	Rules	Arcs	Constituents
0	s --> • vp	[0, 0]	• *Bring the meal*
1	vp --> v • np	[0, 1]	*Bring* • *the meal*
1	np --> • det noun	[1, 1]	• *the meal*
1	np --> • np pp	[1, 1]	• *the meal*
2	np --> det • noun	[1, 2]	*the* • *meal*
3	np --> det noun •	[1, 3]	*the meal* •
3	np --> np • pp	[1, 3]	*the meal* •
3	vp --> v np •	[0, 3]	*Bring the meal* •
3	s --> vp •	[0, 3]	*Bring the meal* •

Chart Parsing

Fig. 11.7. Some arcs of a chart labeled with dotted-rules while parsing *Bring the meal of the day.*

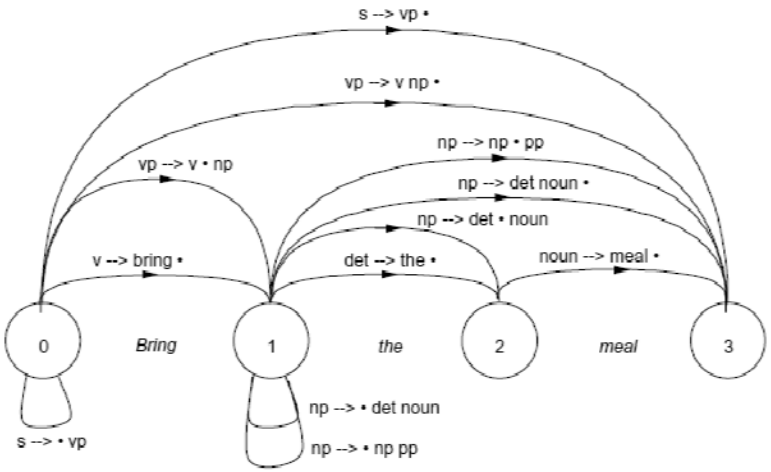


Chart Parsing

- Modules of an Earley Parser
  - The Predictor

Fig. 11.8. Dotted-rules resulting from the recursive run of the predictor with starting goal np.

```
start --> • np          [0, 0]
np --> • det noun       [0, 0]
np --> • det adj noun   [0, 0]
np --> • np pp          [0, 0]
```

Fig. 11.9. Graphic representation of the predictor results.

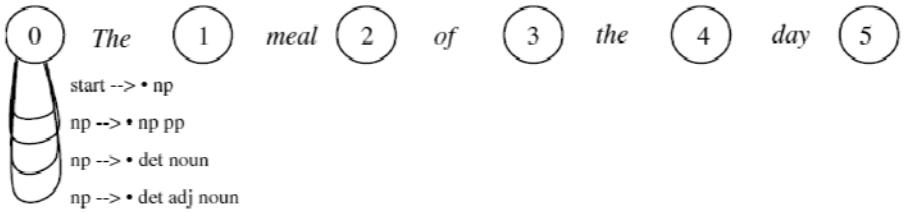


Chart Parsing

- Modules of an Earley Parser
  - The Scanner

Fig. 11.10. The scanner accepts word *The* from the input.

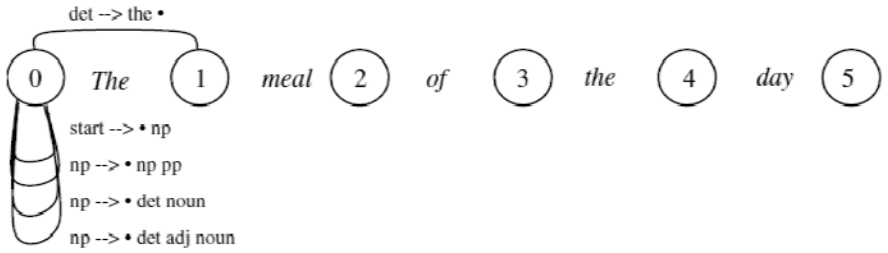


Chart Parsing

- Modules of an Earley Parser
  - The Completer

Fig. 11.11. The completer looks for completed constituents and advances the dot over them.

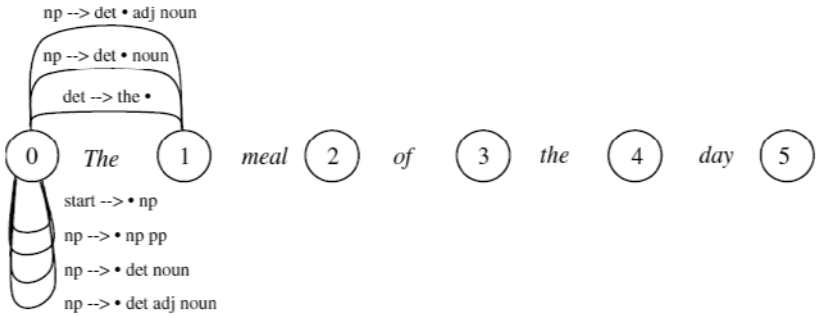


Chart Parsing

Fig. 11.12. Predictor and scanner are run with word *meal*.

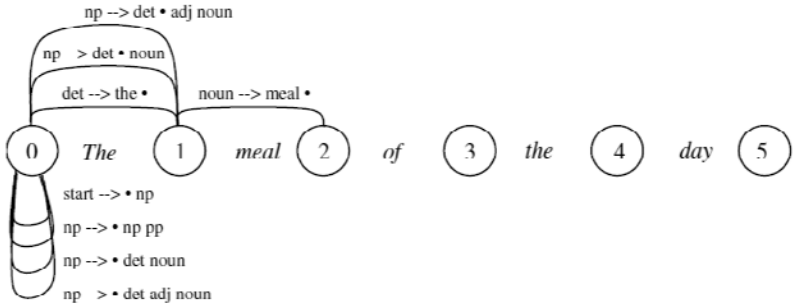


Fig. 11.13. The completer is run to produce new chart entries.

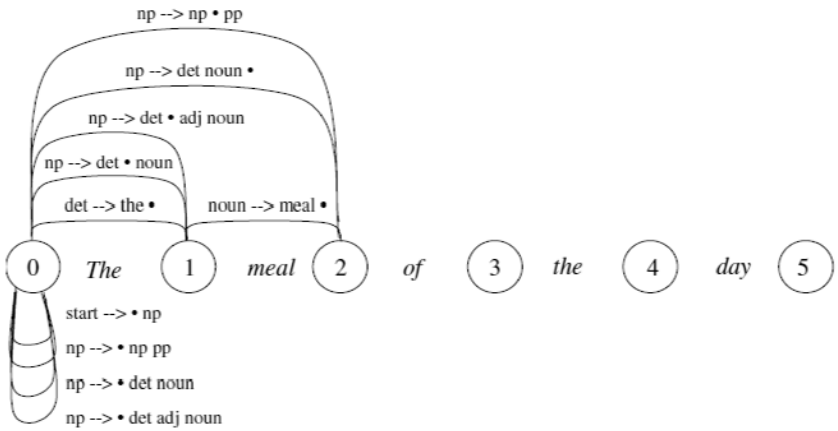


Chart Parsing

- The Earley Algorithm in Prolog
  - Representing the chart
    - np --> np • pp [0, 2]
    - arc(np, [np, '.', pp], 0, 2).
  - The start symbol
    - arc(start, ['.', np], 0, 0).
  - New arcs are stored in the list called *Chart*.

Chart Parsing

Table 11.4. Rules and vocabulary for the chart parser.

Rules	Vocabulary	
rule(np, [d, n]). rule(np, [d, a, n]). rule(np, [np, pp]). rule(pp, [prep, np]).	word(d, [the]). word(n, [waiter]). word(n, [meal]). word(n, [day]).	word(prep, [of]). word(v, [brought]). word(v, [slept]).

Fig. 11.5. A small grammar for restaurant orders in English.

```
s --> vp.              np --> det, noun.
vp --> v, np, pp.      np --> det, adj, noun.
vp --> v, np.          np --> np, pp.
                       pp --> prep, np.
```

Chart Parsing

- The main predicate parse/3
  - parse(Words, Category, FinalChart) :-

```
    expand_chart([arc(start, ['.', Category], 0, 0)],
                 [], Chart),
    earley_parser(Words, 0, FinalPos, Chart,
                  FinalChart),
    member(arc(start, [Category, '.'], 0, FinalPos),
                  FinalChart).
```

Chart Parsing

- The aux predicate expand_chart/3
  - expand_chart([], Chart, Chart).
    expand_chart([Entry | Entries], Chart,
    NewChart) :-
    \+ member(Entry, Chart), !,
    expand_chart([Entries, [Entry | Chart],
    NewChart).
    expand_chart([_ | Entries], Chart, NewChart) :-
    expand_chart(Entries, Chart, NewChart).

Chart Parsing

- The predicate earley_parser/5
  - earley_parser([], FinalPos, FinalPos, Chart, Chart) :-
    !.
    earley_parser(Words, CurPos, FinalPos, Chart,
    FinalChart) :-
    predictor(CurPos, Chart, PredChart),
    NextPos is CurPos + 1,
    scanner(Words, RestWords, CurPos, NextPos,
    PredChart, ScanChart),
    completer(NextPos, ScanChart, NewChart),
    !,
    earley_parser(RestWords, NextPos, FinalPos,
    NewChart, FinalChart).

Chart Parsing

- The Earley Algorithm in Prolog
  - The Predictor
  - The Scanner
  - The Completer
  - An Execution Example
    - ?- parse([the, meal, of, the, day], np, Chart).

ch11/ch11-earley algorithm.pl

Chart Parsing

Table 11.5(1). Additions to the Prolog database.

Module	New Chart Entries in Database
	Position 0
start predictor	arc(start, ['.', np], 0, 0) arc(np, [., d, n], 0, 0), arc(np, [., d, a, n], 0, 0), arc(np, [., np, pp], 0, 0)
	Position 1
scanner completer predictor	arc(d, [the, .], 0, 1) arc(np, [d, ., a, n], 0, 1), arc(np, [d, ., n], 0, 1) []
	Position 2
scanner completer completer predictor	arc(n, [meal, .], 1, 2) arc(np, [d, n, .], 0, 2) arc(np, [np, ., pp], 0, 2), arc(start, [np, .], 0, 2) arc(pp, [., prep, np], 2, 2)
	Position 3
scanner completer predictor	arc(prep, [of, .], 2, 3) arc(pp, [prep, ., np], 2, 3) arc(np, [., d, n], 3, 3), arc(np, [., d, a, n], 3, 3), arc(np, [., np, pp], 3, 3)

Chart Parsing

Table 11.5(2). Additions to the Prolog database.

Module	New Chart Entries in Database
	Position 4
scanner completer predictor	arc(d, [the, .], 3, 4) arc(np, [d, ., a, n], 3, 4), arc(np, [d, ., n], 3, 4) []
	Position 5
scanner completer completer completer completer	arc(n, [day, .], 4, 5) arc(np, [d, n, .], 3, 5) arc(np, [np, ., pp], 3, 5), arc(pp, [prep, np, .], 2, 5) arc(np, [np, pp, .], 0, 5) arc(np, [np, ., pp], 0, 5), arc(start, [np, .], 0, 5)

Chart Parsing

- The Earley Parser to Handle Left-Recursive Rules and Empty Symbols
  - The Earley parser handles left-recursive rules without looping infinitely, except for the predictor, which may be the only place where the parser could be trapped into an infinite execution.
    - \+ member(arc(CAT,['.'|RHS],CrPos,CrPos),Chart)
  - It can also parse null constituents, by revising the scanner.
    - Example: *meals of the day*

ch11/ch11-revised earley algorithm.pl

Remaining Issues

- Probabilistic Parsing of Context-Free Grammars
- A Description of PCFGs
  - The Bottom-Up Chart
  - The Cocke-Younger-Kasami Algorithm in Prolog
  - Adding Probabilities to the CYK Parser

A Description of PCFGs

- The Bottom-up Chart
  - A symbolic bottom-up chart parser known as the Cocke-Younger-Kasami (CYK) algorithm
    - Uses grammars in Chomsky normal form
      - lhs --> rhs1, rhs2.
      - lhs --> [terminal_symbol]
    - Considers constituents of increasing length from the words – length 1 – up to the sentence – length N.
    - Stores completely parsed constituents in the chart, in contrast to the Earley parser.

A Description of PCFGs

- The Bottom-up Chart
  - Procedure
    - Step 1: Annotate the words with all their possible parts of speech.
    - Step 2: Consider contiguous pairs of chart entries that it tries to reduce in a constituent of length *l*, *l* ranging from 2 to *N*.

A Description of PCFGs

The Bottom-up Chart

Fig. 11.16. Annotation of the words with their possible part of speech. Here words are not ambiguous.

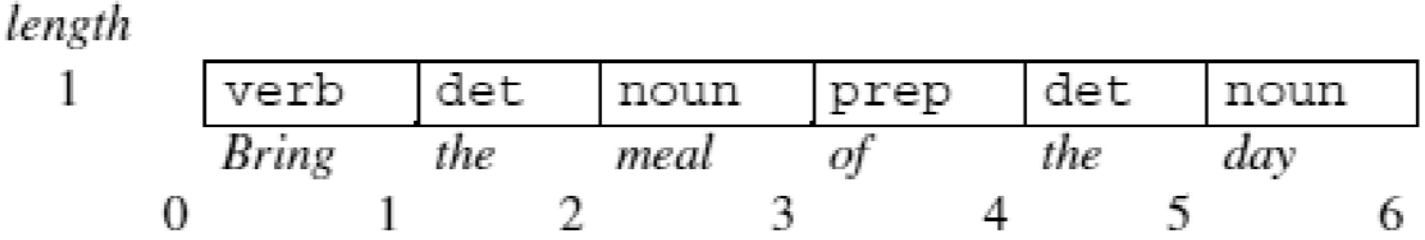
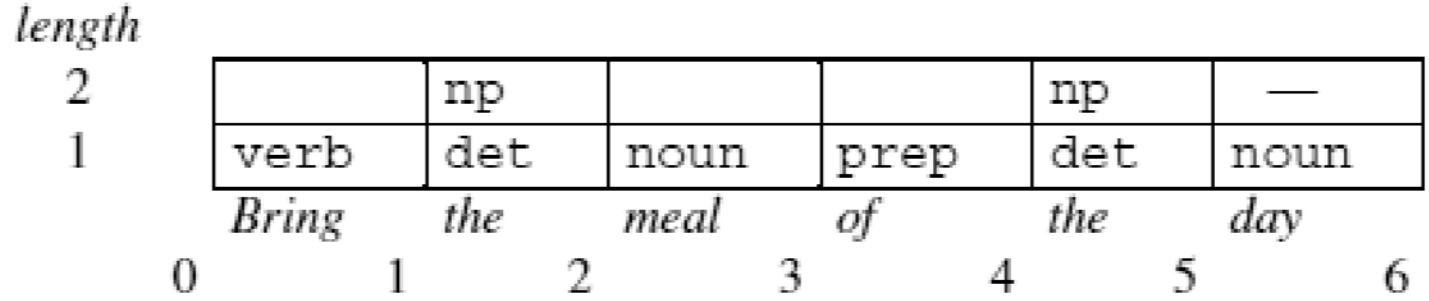


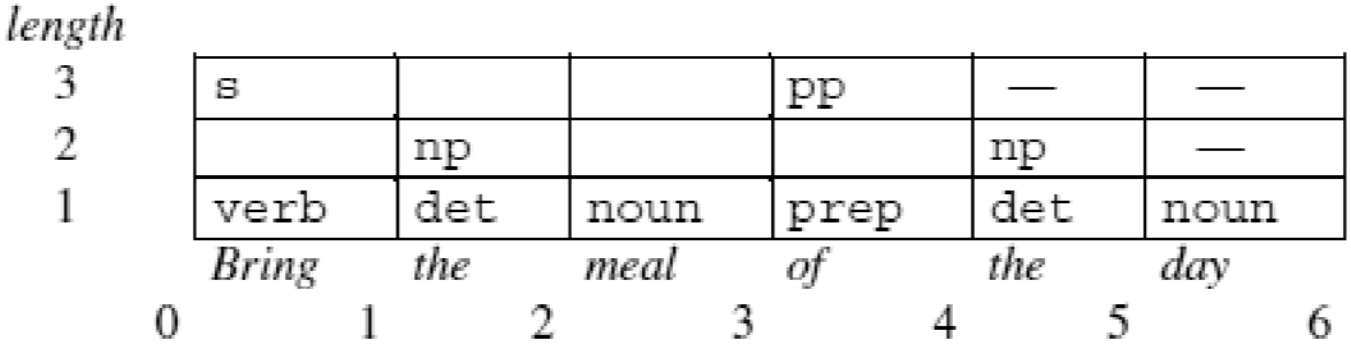
Fig. 11.17. Constituents of length 1 and 2.



A Description of PCFGs

- The Bottom-up Chart

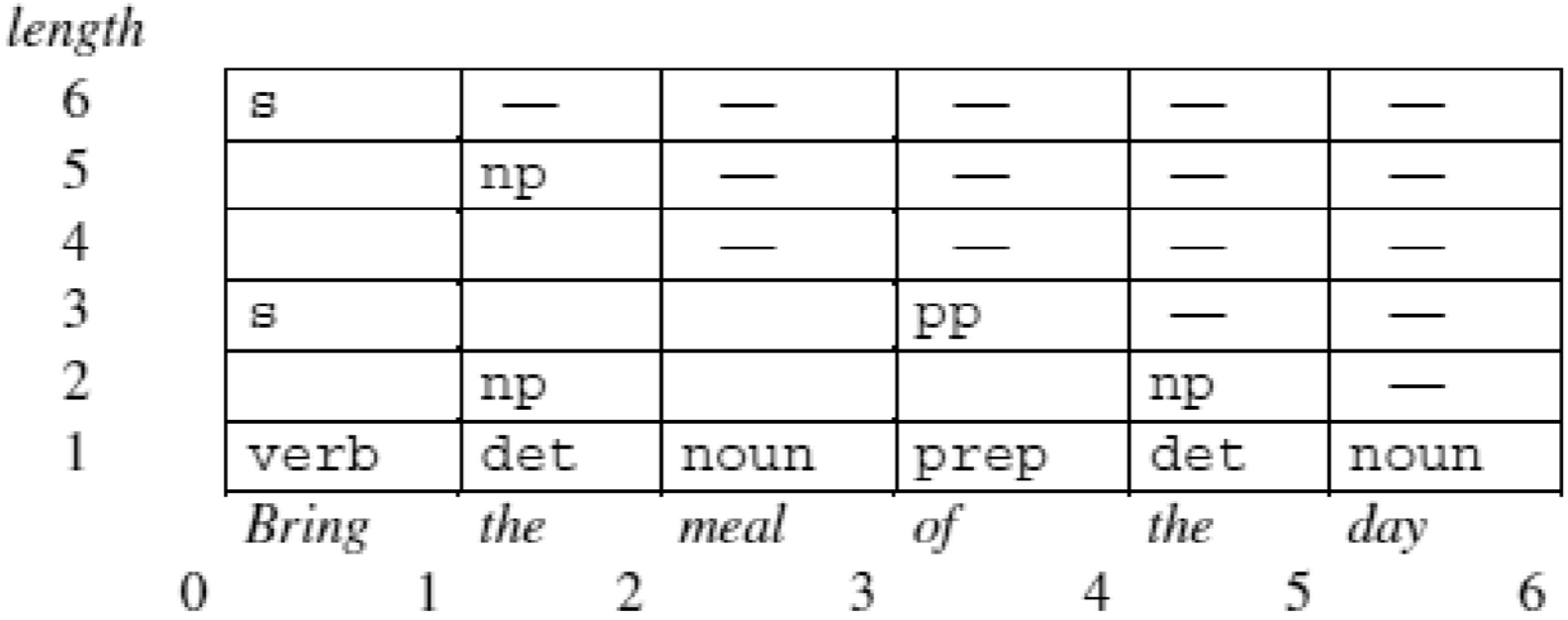
Fig. 11.18. Constituent of lengths 1, 2, and 3.



A Description of PCFGs

- The Bottom-up Chart

Fig. 11.19. The completed parse.



A Description of PCFGs

- The Cocke-Younger-Kasami Algorithm in Prolog

ch11/ch11-CKY algorithm.pl

Parser Evaluation

- Constituency-Based Evaluation

Table 11.8. Bracketing of order *Bring the meal of the day* and crossing brackets.

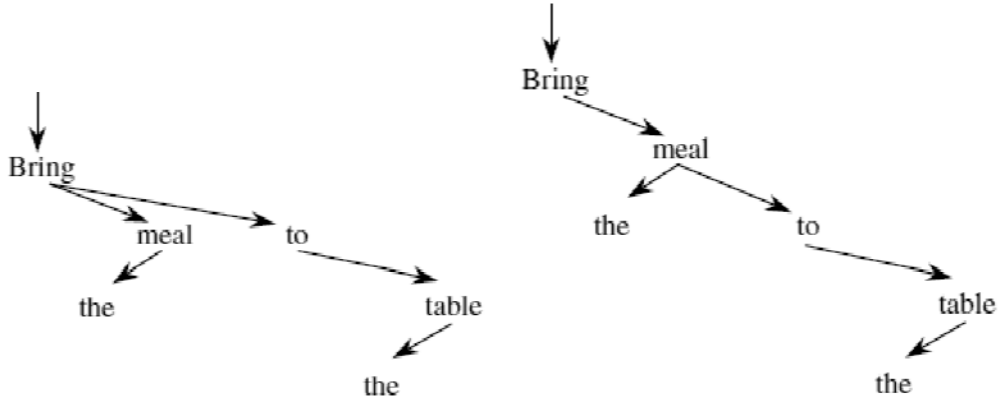
Bracketing	Crossing brackets
(((bring) (the meal)) (of the day))	( ) ( )
((bring) ((the meal) (of the day)))	( )( )

Parser Evaluation

- Dependency-Based Evaluation

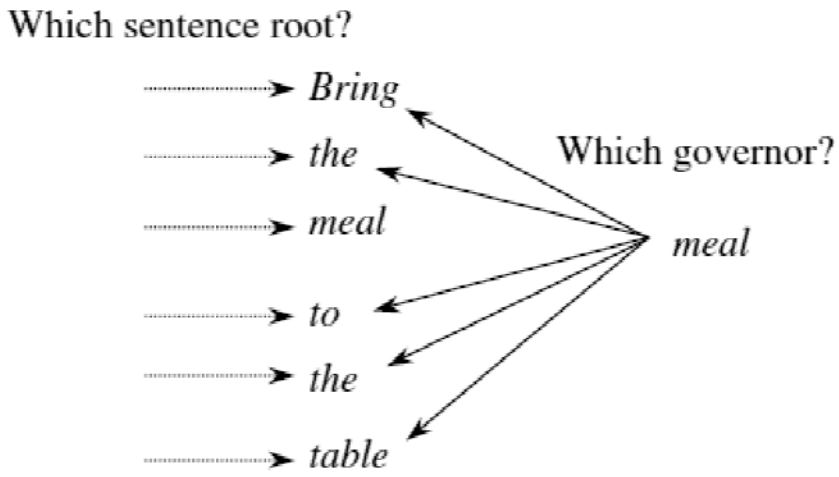
error count

Fig. 11.20. Evaluation of dependency trees: The reference dependency tree (left) and a possible parse output (right).



Parsing Dependencies

Fig. 11.21. Possible sentence roots and governors for word meal. There are *N* possible roots and each remaining word has theoretically *N – 1* possible governors.



Parsing Dependencies

Table 11.9. A representation of dependencies due to Lin (1995). Direction gives the direction of the governor. Symbol '>' means that it is the first occurrence of the word to the right, '>>' the second one, etc. '*' denotes the root of the sentence.

Word position	Word	Direction	Governor	Governor position	Function
1	*Bring*	*		Root	Main verb
2	*the*	>	*meal*	3	Determiner
3	*meal*	<	*Bring*	1	Object
4	*to*	<	*Bring*	1	Location
5	*the*	>	*table*	6	Determiner
6	*table*	<	*to*	4	Prepositional complement

Parsing Dependencies

- Dependency Rules

Fig. 11.22. Examples of *D*-rules

1. determiner ← noun.
2. adjective ← noun.
3. preposition ← noun.
4. noun ← verb.
5. preposition ← verb.
6. verb ← root.

$$\begin{bmatrix} category : noun \\ number : N \\ person : P \\ case : nominative \end{bmatrix} \leftarrow \begin{bmatrix} category : verb \\ number : N \\ person : P \end{bmatrix}$$

Parsing Dependencies

- Extending the Shift-Reduce Algorithm to Parse Dependencies

Table 11.10. The parser transitions where *W* is the initial word list; *I*, the current input word list; *A*, the graph of dependencies; and *S*, the stack.

Actions	Parser actions	Conditions
Initialization	$\langle nil, W, \emptyset \rangle$	
Termination	$\langle S, nil, A \rangle$	
Left-arc	$\langle n\|S,n'\|I,A \rangle \rightarrow \langle S,n'\|I,A \cup \{(n',n)\} \rangle$	$LEX(n) \leftarrow LEX(n') \in R$ $\neg \exists\, n''(n'',n) \in A$
Right-arc	$\langle n\|S,n'\|I,A \rangle \rightarrow \langle n'\|n\|S,I,A \cup \{(n',n)\} \rangle$	$LEX(n) \leftarrow LEX(n') \in R$ $\neg \exists\, n''(n'',n') \in A$
Reduce	$\langle n\|S,I,A \rangle \rightarrow \langle S,I,A \rangle$	$\exists\, n'(n',n) \in A$
Shift	$\langle S,n\|I,A \rangle \rightarrow \langle n\|S,I,A \rangle$	

Parsing Dependencies

- Nivre's Parser in Prolog

ch11/ch11-Nivre Parser.pl

```
shift_reduce([w(the,1),w(waiter,2),w(brought,3),
w(the,4),w(meal,5)], G).
```

Parsing Dependencies

- Finding Dependencies Using Constraints
  - Constraint dependency parsing annotates words with dependencies and function tags.
  - It then applies a set of constraints to find a tag sequence consistent with all the constraints.

Table 11.11. Possible functions according to a word's part of speech.

Parts of speech	Possible governors	Possible functions
Determiner	Noun	det
Noun	Verb	object,iobject
Noun	Prep	pcomp
Verb	Root	root
Prep	Verb, Noun	mod, loc

Parsing Dependencies

- Finding Dependencies Using Constraints

Words	*Bring*	*the*	*meal*	*to*	*the*	*table*
Position	1	2	3	4	5	6
Part of speech	verb	det	noun	prep	det	noun
Possible tags	<nil, root>	<3, det> <6, det>	<4, pcomp> <1, object> <1, iobject>	<3, mod> <1, loc>	<3, det> <6, det>	<4, pcomp> <1, object> <1, iobject>
Words	*Bring*	*the*	*meal*	*to*	*The*	*table*
Position	1	2	3	4	5	6
Part of speech	verb	det	noun	prep	Det	noun
Possible tags	<nil, root>	<3, det>	<1, object> <1, iobject>	<3, mod> <1, loc>	<6, det>	<4, pcomp>