

# Towards more efficient parsers

- Combining bottom-up parsing with top-down prediction
  - From shift-reduce to left-corner parsing
  - Adding more top-down filtering: link tables
- Memoization of partial results
  - well-formed substring tables
  - active charts

# From shift-reduce to left-corner parsing

- Shift-reduce parsing is not goal directed at all:
  - Reduction of every possible substring,
  - obtaining every possible analysis for it.
- Idea to revise shift-reduce strategy:
  - Take a particular element  $x$  (here: the leftmost).
  - $x$  triggers those rules it can occur in, to make predictions about the material occurring around  $x$ .

## A shift-reduce parser for grammars in CNF

```
% ?- recognise([], <list(word)>, []).
```

```
recognise([s], [], []).
```

```
recognise([Y,X|Rest]) -->           % reduce
    {LHS ---> X,Y},
    recognise([LHS|Rest]).
```

```
recognise(Stack) -->                % shift
    [Word],
    {Cat ---> [Word]},
    recognise([Cat|Stack]).
```

## A left-corner parser for grammars in CNF

```
% ?- recognise(s,<list(word)>,[ ]).
```

```
recognise(Phrase) --> [Word],  
                {Cat ----> [Word]},  
                lc(Cat,Phrase).
```

```
lc(Phrase,Phrase) --> [ ].
```

```
lc(SubPhrase,SuperPhrase) -->  
  {Phrase ----> SubPhrase,Right},  
  recognise(Right),  
  lc(Phrase,SuperPhrase).
```

## Problems of basic left-corner approach

- There can be a choice involved in picking a rule which
  - projects a particular word
  - projects a particular phrase
- How do we make sure we only pick a category which is on our path up to the goal?
  - Define a **link table** encoding the transitive closure of the left-corner relation. This is always a finite table!
  - Use it as an **oracle** guiding us to pick a reasonable candidate.

## Example for a link table

For a grammar with the following non-terminal rules

s	---->	np, vp.	vp	---->	v, np.
np	---->	det, n.	n	---->	n, pp.
pp	---->	p, np.			

one can define or automatically deduce the link table

link(s,s).	link(np,np).	link(det,det).
link(n,n).	link(pp,pp).	link(p,p).
link(n0,n0).	link(np,s).	link(det,np).
link(p,pp).	link(v,vp).	

## Using a link table in a left-corner parser

```
recognise(Phrase) --> [Word],  
                    {Cat ----> [Word]},  
                    {link(Cat,Phrase)},  
                    lc(Cat,Phrase).
```

```
lc(Phrase,Phrase) --> [].
```

```
lc(SubPhrase,SuperPhrase) -->  
  {Phrase ----> SubPhrase,Right},  
  {link(Phrase,SuperPhrase)},  
  recognise(Right),  
  lc(Phrase,SuperPhrase).
```

## Observation: Inefficiency of backtracking

Two example sentences:

(1) He [gave [the young cat] [to Bill]].

(2) He [gave [the young cat] [some milk]].

The corresponding grammar rules:

`vp --> v_ditrans, np, pp_to.`

`vp --> v_ditrans, np, np.`



## Solution: Memoization

- Store intermediate results:
  - a) completely analyzed constituents:  
**well-formed substring table** or **(passive) chart**
  - b) complete or partial analyses:  
**(active) chart**
- All intermediate results need to be stored for completeness.
- All possible solutions are explored in parallel.

# CYK Parser

- Developed independently by Cocke, Younger, and Kasami
- Grammar has to be in Chomsky Normal Form (CNF), only
  - RHS with a single terminal:  $A \rightarrow a$
  - RHS with two non-terminals:  $A \rightarrow BC$
- The well-formed substring table, henceforth (passive) chart, for a string of length  $n$  is an  $n \times n$  matrix.

# The passive chart

- Sentence representation showing position and word indices:

$\cdot_0 \ w_1 \ \cdot_1 \ w_2 \ \cdot_2 \ w_3 \ \cdot_3 \ w_4 \ \cdot_4 \ w_5 \ \cdot_5 \ w_6 \ \cdot_6$

- An entry in a field  $(i, j)$  of the chart encodes the set of categories which spans the string from position  $i$  to  $j$ .
- More formally:  $\text{chart}(i, j) = \{A \mid A \Rightarrow^* w_{i+1} \dots w_j\}$

# Coverage represented in the chart

An input sentence with 6 words:

$\cdot_0$   $W_1$   $\cdot_1$   $W_2$   $\cdot_2$   $W_3$   $\cdot_3$   $W_4$   $\cdot_4$   $W_5$   $\cdot_5$   $W_6$   $\cdot_6$

Coverage represented in the chart:

TO:

	1	2	3	4	5	6
0	0-1	0-2	0-3	0-4	0-5	0-6
1		1-2	1-3	1-4	1-5	1-6
2			2-3	2-4	2-5	2-6
3				3-4	3-5	3-6
4					4-5	4-6
5						5-6

FROM:

# Example for coverage represented in chart

Example sentence:

·<sub>0</sub> the ·<sub>1</sub> young ·<sub>2</sub> boy ·<sub>3</sub> saw ·<sub>4</sub> the ·<sub>5</sub> dragon ·<sub>6</sub>

Coverage represented in chart:

	1	2	3	4	5	6
0	the	the young	the young boy	the young boy saw	the young boy saw the	the young boy saw the dragon
1		young	young boy	young boy saw	young boy saw the	young boy saw the dragon
2			boy	boy saw	boy saw the	boy saw the dragon
3				saw	saw the	saw the dragon
4					the	the dragon
5						dragon

# An example for a filled-in chart

**Input sentence:**

$\cdot_0$  the  $\cdot_1$  young  $\cdot_2$  boy  $\cdot_3$  saw  $\cdot_4$  the  $\cdot_5$  dragon  $\cdot_6$

**Chart:**

	1	2	3	4	5	6
0	{Det}	{}	{NP}	{}	{}	{S}
1		{Adj}	{N}	{}	{}	{}
2			{N}	{}	{}	{}
3				{V}	{}	{VP}
4					{Det}	{NP}
5						{N}

**Grammar:**

$S \rightarrow NP VP$

$VP \rightarrow Vt NP$

$NP \rightarrow Det N$

$N \rightarrow Adj N$

$Vt \rightarrow \text{saw}$

$Det \rightarrow \text{the}$

$Det \rightarrow \text{a}$

$N \rightarrow \text{dragon}$

$N \rightarrow \text{boy}$

$Adj \rightarrow \text{young}$

## Filling in the chart left-to-right, depth-first

	1	2	3	4	5	6
0	<b>1!</b>	<b>3</b>	<b>6</b>	<b>10</b>	<b>15</b>	<b>21</b>
1		<b>2!</b>	<b>5</b>	<b>9</b>	<b>14</b>	<b>20</b>
2			<b>4!</b>	<b>8</b>	<b>13</b>	<b>19</b>
3				<b>7!</b>	<b>12</b>	<b>18</b>
4					<b>11!</b>	<b>17</b>
5						<b>16!</b>

for  $j := 1$  to 6

lexical-chart-fill( $j - 1, j$ )

for  $i := j - 2$  down to 0

syntactic-chart-fill( $i, j$ )

## lexical-chart-fill( $j-1,j$ )

- Idea: Lexical lookup. Fill the field  $(j - 1, j)$  in the chart with the preterminal category dominating word  $j$ .
- Realized as:

$$\text{chart}(j - 1, j) := \{X \mid X \rightarrow \text{word}_j \in P\}$$



## syntactic-chart-fill(i,j)

- Idea: Perform all reduction step using syntactic rules such that the reduced symbol covers the string from  $i$  to  $j$ .
- Realized as:

$$chart(i, j) = \left\{ A \left| \begin{array}{l} A \rightarrow BC \in P, \\ i < k < j, \\ B \in chart(i, k), \\ C \in chart(k, j) \end{array} \right. \right\}$$

## Explicit version of syntactic-chart-fill(i,j)

- Needed: version making explicit enumerations of
  - every possible value of  $k$  and
  - every context free rule

- Code:

$chart(i, j) := \{\}$ .

for  $k := i + 1$  to  $j - 1$  do

    for every  $A \rightarrow BC \in P$  do

        if  $B \in chart(i, k)$  and  $C \in chart(k, j)$  then

$chart(i, j) := chart(i, j) \cup \{A\}$ .

## The complete CYK algorithm

for  $j := 1$  to  $n$  do

$chart(j - 1, j) := \{X \mid X \rightarrow \text{word}_j \in P\}$

for  $i := j - 2$  down to  $0$  do

$chart(i, j) := \{\}$

for  $k := i + 1$  to  $j - 1$  do

for every  $A \rightarrow BC \in P$  do

if  $B \in chart(i, k)$  and  $C \in chart(k, j)$  then

$chart(i, j) := chart(i, j) \cup \{A\}$

if  $S \in chart(0, n)$  then accept else reject

# The CYK algorithm in PROLOG

## (cky/cky.pl)

```
% Data structures: chart(From,To,Category)
:- dynamic chart/3.

% Operator for grammar rules
:- op(1200,xfx,'--->').
```

```
% recognize(+WordList,?Startsymbol)
% top-level predicate for CYK recognizer

recognize(S,Cat) :-
    retractall(chart(_,_,_)),
    length(S,N),
    fill(0,N,S),
    chart(0,N,Cat).
```

```

% fill(+Current minus one,+Last,+WordList)
% Main j-loop from 1 to number of words in string.

fill(N,N, []).
fill(JminOne,N,[W|Ws]) :-
    J is JminOne + 1,
    lexical_chart_fill(J,JminOne,W),
    %
    I is J - 2,
    syntactic_chart_fill(I,J),
    %
    fill(J,N,Ws).

```

```
% lexical_chart_fill(+J,+JminOne,+Word)
% fill main diagonal with preterminal categories

lexical_chart_fill(J,JminOne,W) :-
    findall_unique(X,(X ----> [W]),Cats),
    add_all_to_chart(JminOne,J,Cats).
```

```
% syntactic_chart_fill(+I,+J)
% i-loop from J-2 down to 0

syntactic_chart_fill(-1,_) :- !.
syntactic_chart_fill(I,J) :-
    K is I+1,
    build_phrases_from_to(I,K,J),
    IminOne is I-1,
    syntactic_chart_fill(IminOne,J).
```



```

% build_phrases_from_to(+From,+Current,+To)

build_phrases_from_to(_,J,J) :- !.
build_phrases_from_to(I,K,J) :-
    findall_unique(A,(chart(I,K,B),
                      chart(K,J,C),
                      (A ---> [B,C])),
                  List),
    add_all_to_chart(I,J,List),
    KplusOne is K+1,
    build_phrases_from_to(I,KplusOne,J).

```

```

% add_one_to_chart(+FromIndex,+ToIndex,+Contents)
% a) only add if it does not yet exist:
add_one_to_chart(From,To,Cat) :- chart(From,To,Cat), !.

% b) add a chart entry
add_one_to_chart(From,To,Cat) :-
    assertz(chart(From,To,Cat)).

add_all_to_chart(_,_,[]).
add_all_to_chart(From,To,[Cat|Cats]) :-
    add_one_to_chart(From,To,Cat),
    add_all_to_chart(From,To,Cats).

```