

Towards more efficient parsers

Detmar Meurers: Intro to Computational Linguistics I
OSU, LING 684.01, February 9., 2004

Ideas

- 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

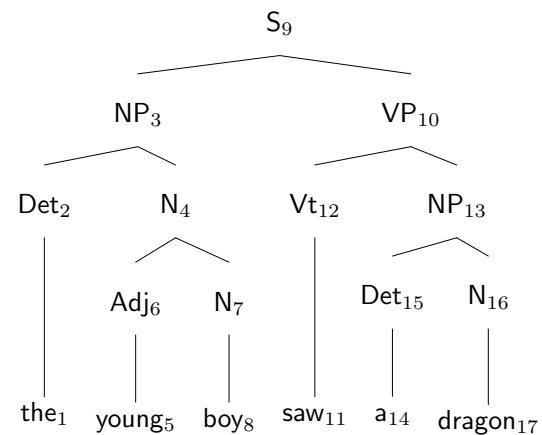
2

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 .

3

Left-corner, left-right, depth-first tree traversal



S → NP VP
VP → Vt NP
NP → Det N
N → Adj N

Vt → saw
Det → the
Det → a
N → dragon
N → boy
Adj → young

In the figure above, we numbered the mother in the tree at the time the rule is looked up of which it is the left-hand side category. Alternatively, one could number the mother only at the time when the parser tries to prove it's the left corner of something.

4

A left-corner parser for grammars in CNF using ordinary strings (parser/simple/cnf_lc.pl)

```
:- op(1100,xfx,'--->').

recognise(Phrase, [Word|Rest]) :-
    (Cat ---> [Word]),
    lc(Cat, Phrase, Rest).

lc(Phrase, Phrase, _).

lc(SubPhrase, SuperPhrase, String) :-
    (Phrase ---> [SubPhrase,Right]),
    append(SubString,Rest,String),
    recognise(Right, SubString),
    lc(Phrase, SuperPhrase, Rest).
```

5

A left-corner parser for grammars in CNF using difference lists to encode the string (parser/simple/cnf_lc_diff_list.pl)

```
:- op(1100,xfx,'--->').

recognise(Phrase, [Word|S0], S) :-
    (Cat ---> [Word]),
    lc(Cat, Phrase, S0, S).

lc(Phrase,Phrase, S, S).

lc(SubPhrase, SuperPhrase, S0, S) :-
    (Phrase ---> [SubPhrase,Right]),
    recognise(Right, S0, S1),
    lc(Phrase, SuperPhrase, S1, S).
```

6

A left-corner parser for grammars in CNF using DCG notation to encode the string (parser/simple/cnf_lc_dcg.pl)

```
:- op(1100,xfx,'--->').

% ?- 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).
```

7

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.

8

Example for a link table

For a grammar with the following non-terminal rules

```
:- op(1100,xfx,'--->').
```

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

Using a link table in a left-corner parser

```
:- op(1100,xfx,'--->').
```

```
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).
```