Natuurlijke Taalverwerking
Natural Language Processing

Gosse Bouma and Begoña Villada

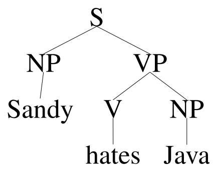
3e trimester 2002/2003

Overview

1. DCGs
   - Trees and other structures,
   - Alpino Demo,
   - Generation,
   - Machine Translation.
2. Parsing
   - Left-recursion,
   - Top-down vs. Bottom-up strategies,
   - Shift-reduce parsing.

Trees

- Sandy hates Java.



Trees

```
s(tree(s,[NP,VP])) --> np(NP), vp(VP).
vp(tree(vp,[V,NP])) --> v(V), np(NP).
np(leaf(np,sandy)) --> [sandy].

?- s(Tree,[sandy,hates,java],[]).

Tree = tree(s,[tree(np,sandy),
               tree(vp,[tree(v,hates),
                        tree(np,java)
                       ])
              ])
```

Semantics

- Kim knows Perl
- know(k,p)

```
s(Sem) --> np(Subj), vp(Subj,Sem).
vp(Sem) --> vp(Obj,Subj,Sem), np(Obj).
np(k) --> [kim].
np(p) --> [perl].
```

Generation: Applications

- Dialogue Systems: formulating the output of a database query in natural language,
- Generation of reports (wheather forecasts, stock market reports, etc.) on the basis of numerical data.
- Machine Translation: generating a string in another language with the same meaning as the original sentence.

Parsing vs Generation

- Parsing maps a string onto a structure (tree, semantics), according to some grammar.
- Generation maps a structure (semantics) onto a string, according to some grammar.

Generation with DCG

```
?- s(know(kim,perl),String,[]).

String = [kim,knows,perl]
```

Machine Translation

- From Systran (www.systransoft.com)
- Bovendien worden woorden met dezelfde betekenis (in het Nederlands en Engels) gerepresenteerd door hetzelfde interlingua symbool.
- *Moreover words with the same meaning (in Dutch and English) are represented by the same interlingua symbol.*

Machine Translation

- Sandy haat Java $\longrightarrow$
- *Sandy hates Java.*
- Kim zegt dat Sandy Java haat $\longrightarrow$
- *Kim says that Sandy hates Java.*
- Welke talen kent Kim? $\longrightarrow$
- *Which languages does Kim know*

Machine Translation

- From Systran (www.systransoft.com)
- De NPs mag je zo eenvoudig mogelijk houden.
- *The NPs can keep you this way simple possible.*

Grammar-based MT

- Parse Dutch input with a grammar for Dutch,
- which produces a language independent syntactic-semantic structure S.
- Generate an English output for structure S using a grammar for English.
- What is a suitable structural representation?

Interlingua

- Represent words by canonical elements (no inflection, tense, agreement, case marking etc)
  - ⋆ Jan, John → john,
  - ⋆ slapen, slaapt, sleeps, sleep → sleep
- Map strings into trees with a fixed order for verbs (and other elements) and arguments.
  - ⋆ Kim kent Perl, dat Kim Perl kent → i_s(kim,i_vp(know,perl))

Machine Translation Problems

- Mismatches in syntax
  - ⋆ Kim zwemt → Kim is swimming,
  - ⋆ Zwemt Kim → Does Kim swim,
  - ⋆ Kim zwemt niet → Kim does not swim,
  - ⋆ Kim zwemt graag → Kim likes to swim.

DCG with Interlingua

- All rules in the grammar must be extended with an Interlingua argument,

```
s(i_s(Subj,VP)) --> np(Subj), vp(VP).
```

- Both grammars must map semantically equivalent sentences onto the same Interlingua representation.

Machine Translation Problems

- Lexical Ambiguity
  - ⋆ Kim zette de wortels op tafel → carrot
  - ⋆ De wortel van 144 is 12. → root
  - ⋆ Het automatisch vertalen van een Nederlandse zin in het Engels betekent dat, ...
  - ⋆ Automatic translating a Dutch sense in English means that,...

Machine Translation Problems

- Syntactic Ambiguity
  - ⋆ Wie kent Kim →
  - ⋆ Who knows Kim,
  - ⋆ Who does Kim know
  - ⋆ Test je programma ...→
  - ⋆ You test programme ....

Parsing

- Prolog provides top-down, depth-first, parsing strategy as default,
- Many alternative strategies exist,
- Often more robust and efficient.

Top-down Parsing

- DCG uses Prolog top-down search strategy,
- Therefore, left-recursion leads to problems,

```
ancestor(X,Y) :-
    parent(X,Y).
ancestor(X,Y) :-
    ancestor(X,Z), parent(Z,Y).
```

Left-recursion in Grammar

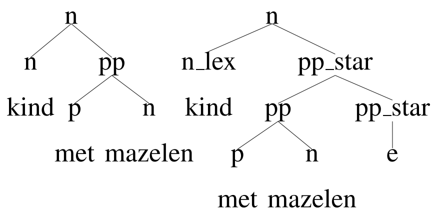
- een kind, een kind in het park
- `n --> n, pp.`
- Kim slaapt, Kim slaapt tot 10 uur
- `vp --> vp, pp`
- Kim slaapt en Sandy werkt
- `s --> s, [en], s.`

Left-recursion in Grammar

- Peter's (broer's) huis
- `np --> det, n`
- `det --> np, [s].`
- een (erg) aardig kind
- `a --> int, a.`
- `int --> [erg];[heel];[vet]; [].`

Removing Left-recursion

- Changing the grammar also changes the structure assigned to input-strings,
- Although this can be fixed as well.



Removing Left-recursion

```
n --> n, pp
pp --> p, n
```

```
n --> n_lex, pp_star
n_lex --> [kind].
pp_star --> pp, pp_star.
pp_star --> [].
pp --> p, np.
```

Removing Left-recursion

- Removing left-recursion can lead to explosion of the number of rules in the grammar. (Moore, Proceedings NAACL, 2000)

Grammar	Number of Rules
Toy	88
Paull "Best"	156
Paull "Lexicographic"	970
Paull "Worst"	5696

Alternative Parsing Methods

- Prolog searches top-down, depth-first,
- Alternatives:
  - Bottom-up Parsing: work from the input towards the goal (s).
  - Breadth-first (parallel): Explore all ways to expand a rule in parallel.

Top-down Parsing in Prolog

```
top_down(Cat,P0,P1) :-
    rule(Cat,Daughters),
    find_ds(Daughters,P0,P1).
top_down(Cat,[Word|Ws],Ws) :-
    lex(Cat,Word).

find_ds([D1|Ds],P0,P2) :-
     top_down(D1,P0,P1),
     find_ds(Ds,P1,P2).
find_ds([],P0,P0).
```

Alternative Parse Strategy

- Requires separation of rules (data) and parser (algorithm)
- Grammar rules as data:

```
% rule(Mother_Category,List_Daughters)
rule(s,[np,vp]).

% lex(Category,Word).
lex(np,kim).
```

Top-down Parsing

- S
- NP VP
- DET N VP
- the N VP
- the dog VP
- the dog V
- the dog barks

Bottom-up Parsing

- the dog barks
- DET dog barks
- DET N barks
- NP barks
- NP V
- NP VP
- S

Shift-reduce Algorithm

- Stack: for storing intermediate result,
- Shift: Remove the leftmost element of the input and add its category to the Stack,
- Reduce: Replace $C_1..C_n$ on the Stack by C_0 given a rule $C_0 \rightarrow C_1..C_n$.

Shift-reduce parsing

- Bottom-up parsing!
- Start with the input, and search for lexical categories,
- Try to combine categories into phrases
- Try to combine phrases into larger phrases or a sentence.
- Bottom-up parsers do not loop on left-recursive rules.

Shift-reduce Algorithm

	String	Stack	action	rule
1	the dog barks	[]	sh	lex(det,the)
2	dog barks	[det]	sh	lex(n,dog)
3	barks	[det,n]	red	rule(np,[det,n])
4	barks	[np]	sh	lex(v,barks)
5		[np,v]	red	rule(vp,[v])
6		[np,vp]	red	rule(s,[np,vp])
7		[s]		

Shift-reduce in Prolog

```
sr(Input,Stack) :-
  reduce(Stack,NewStack),
  sr(Input,NewStack).
sr(Input,Stack) :-
  shift(Input,Rest,Cat),
  sr(Rest,[Cat|Stack]).

shift([Wrd|Input],Input,Cat) :-
  lex(Cat,Wrd).
```

Shift-reduce in Prolog

```
reduce(Stack,[M|NewStack]) :-
  reduce_rule(M,Ds),
  append(Ds,NewStack,Stack).

reduce_rule(s,[vp,np]).
```

- Note the order of Ds in reduce_rule is reversed!

Optimized Reduce

```
reduce([vp,np|Stack],[s|Stack]).
reduce([n,det|Stack],[np|Stack]).
```

- No need for append or search,
- Rules can be automatically converted in reduce actions.