Parsing problem

Use the grammar to derive an explicit representation of the constituency of an input sentence.

Recall: Grammatical derivations
Tree Substitution Grammar

\[
S \\
\uparrow \\
\begin{array}{c}
NP \\
D \\
dog
\end{array} \\
\downarrow \\
\begin{array}{c}
VP \\
N \\
barks
\end{array} \\
\downarrow \\
V
\]

\[T_1\]

Recall: Grammatical derivations
Tree Substitution Grammar

\[
S \\
\uparrow \\
\begin{array}{c}
NP \\
D \\
dog
\end{array} \\
\downarrow \\
\begin{array}{c}
VP \\
N \\
barks
\end{array} \\
\downarrow \\
V
\]

\[T_2\]

Dependency

Sentences built from words by operations

\[
S \\
\uparrow \\
\begin{array}{c}
NP \\
D \\
the
\end{array} \\
\downarrow \\
\begin{array}{c}
VP \\
N \\
barks
\end{array} \\
\downarrow \\
V
\]

\[T_3\]
Dependency formalisms

Tree Substitution Grammar

- barks: T₁
  - substitution at NP node
- dog: T₂
  - substitution at D node
- the: T₃

Parsing problem

Start from words and possible elementary trees for the words:

```
S
  NP
    D
    N
    V
  VP
    barks
```

Key idea of parsing

Incremental structure building

Adapt steps of grammatical derivation to keep track of the order of constituents.

Incremental structure building – symmetric case

You have two things next to each other that you can combine:

- an incomplete constituent
  - with room for an X next
- a complete new X
  - you can add in

and you combine them:

- a more complete constituent
  - that combines the two together

- a complete new X
  - you can add in
- an incomplete constituent
  - with room for an X just before

and you combine them:

- a more complete constituent
  - that combines the two together
Key idea of parsing

Incremental structure building

\[
\begin{array}{c}
\text{NP} \\
\text{D} \\
\text{the} \\
\text{dog}
\end{array}
\quad
\begin{array}{c}
\text{NP} \\
\text{D} \\
\text{N}
\end{array}
\]

a complete new D
you can add in
an incomplete NP with
room for a D just before

Key idea of parsing

Incremental structure building

\[
\begin{array}{c}
\text{S} \\
\text{NP} \\
\text{D} \\
\text{N} \\
\text{barks}
\end{array}
\quad
\begin{array}{c}
\text{NP} \\
\text{VP}
\end{array}
\]

a new complete S
that combines the two together

Key idea of parsing

Incremental structure building

\[
\begin{array}{c}
\text{S} \\
\text{NP} \\
\text{D} \\
\text{N} \\
\text{V} \\
\text{barks}
\end{array}
\quad
\begin{array}{c}
\text{NP} \\
\text{VP}
\end{array}
\]

a new complete S
that combines the two together

Adding things to trees – the periphery

Where is the XP in that first tree \( T \)?

an incomplete constituent
with room for an XP next
a complete new XP
you can add in

The periphery

Consider this example for \( T \)

\[
\begin{array}{c}
A \\
\text{XP} \\
\text{YP} \\
\text{ZP} \\
\text{XP} \\
\text{WP} \\
\text{XP} \\
\text{XP} \\
\text{W} \\
\text{XP} \\
\text{X}
\end{array}
\]
Suppose $C$ is a complement of category $XP$  
Consider this sample $T$  
Can new $XP$ go here?  

Suppose $C$ is a complement of category $XP$  
Consider this sample $T$  
Can new $XP$ go here?  

Suppose $C$ is a complement of category $XP$  
Consider this sample $T$  
Can new $XP$ go here?  

Suppose $C$ is a complement of category $XP$  
Consider this sample $T$  
Can new $XP$ go here?  

Periphery  
A completed constituent on the right can only be added to an incomplete tree on its right periphery – after the last word already accounted for in the incomplete tree and up to the first gap not yet accounted for.
Periphery

A completed constituent on the left can only be added to an incomplete tree on its left periphery – before the first word already accounted for in the incomplete tree and up to the last gap not yet accounted for.

Coding this up

New structure

\[
\text{node(\text{category}, left, head, right)}
\]

Assume that lexical material in structure is a substring including the head.

FTHR - full to head's right

Mutual recursion again!

Base cases – lexical nodes

\[
\text{fthr(leaf(\_))}.
\]
\[
\% \text{fthr(gap(\_))} : - \text{fail}.
\]

FTHR ct'd

Recursive case: make sure there are no gaps in the right list and the head subtree is fthr.

\[
\text{fthr(node(\_,\_,H,R))} : -
\]
\[
\text{nogaps(R), fthr(H)}.
\]
\[
\text{nogaps([\]).}
\]
\[
\text{nogaps([leaf(\_)|L])} : - \text{nogaps(L)}.
\]
\[
\text{nogaps([node(\_,\_,\_,\_)|L])} : - \text{nogaps(L)}.
\]

Suppose \( C \) is a complement of category \( XP \)

Consider this sample \( T \)

\[
\begin{align*}
A & \quad \text{XP} \\
\text{XP} & \quad \text{YP} \\
\text{YP} & \quad \text{XP} \\
\text{ZP} & \quad \text{XP} \\
\text{WP} & \quad \text{XP} \\
\text{XP} & \quad \text{XP} \\
\text{XP} & \quad \text{XP} \\
\text{WP} & \quad \text{XP} \\
\text{XP} & \quad \text{XP} \\
W & \quad \text{XP} \\
\text{XP} & \\
X
\end{align*}
\]

Replace next gap with node

\[
\text{replace_next([\text{gap}(C) | \text{Rest}],}
\]
\[
\text{node(C, L, H, R),}
\]
\[
\text{[node(C, L, H, R) | Rest]).}
\]
\[
\text{replace_next([node(C, L, H, R) | \text{Rest}], N,}
\]
\[
\text{[node(C, L, H, R) | Result]) : -}
\]
\[
\text{replace_next(\text{Rest}, N, Result).}
\]

Substitute at next place

Replace next gap with node

\[
\text{replace_next([\text{gap}(C) | \text{Rest}],}
\]
\[
\text{node(C, L, H, R),}
\]
\[
\text{[node(C, L, H, R) | Rest]).}
\]
\[
\text{replace_next([node(C, L, H, R) | \text{Rest}], N,}
\]
\[
\text{[node(C, L, H, R) | Result]) : -}
\]
\[
\text{replace_next(\text{Rest}, N, Result).}
\]
Suppose $C$ is a complement of category $XP$.

Consider this sample $T$:

```
A
  / \  
XP   YP
  /   |
ZP   XP
  |
WP   XP XP
  |
W   XP
  |
X
```

Substitute at next place

```prolog
subst_next(node(C,L,H,R), N, node(C,L,H,X)) :- fthr(H), replace_next(R, N, X). subst_next(node(C,L,H,R), N, node(C,L,X,R)) :- subst_next(H, N, X).
```

Suppose $C$ is a complement of category $XP$.

Consider this sample $T$:

```
A
  / \  
XP   YP
  /   |
ZP   XP
  |
WP   XP XP
  |
W   XP
  |
X
```

Summary

```prolog
combine(T1, T2, T3) :- fthr(T2), fthl(T2), subst_next(T1, T2, T3).
```

Parsing

Our basic strategy will be to use this parsing combination operation to add complete constituents that we find into incomplete constituents that we are still building.

What kind of data structures do we need to keep track of all these constituents?

Parsing – stack and search

Consider this example:

```
S
  / \  
VP   NP
  /   |
V   D   N
  |
walk the dog
```
Parsing - stack and search

Consider this example during parsing

\[
S \\
/ \ \\
VP \quad NP \\
/ \ \\
V \quad D \\
/ \ \\
walk \quad the \\
\]

Stack

Walk does not combine directly with the

\[
S \\
/ \ \\
VP \quad NP \\
/ \ \\
V \quad D \\
/ \ \\
walk \quad the \\
\]

We have to push walk and see what we can do with the.

Stack

We now consider the and dog

\[
S \\
/ \ \\
VP \quad NP \\
/ \ \\
V \quad D \\
/ \ \\
walk \quad the \quad dog \\
\]

(Walk is still on the stack.) We simplify.

Stack

We now consider the and dog

\[
S \\
/ \ \\
VP \quad NP \\
/ \ \\
V \quad D \\
/ \ \\
walk \quad the \quad dog \\
\]

(Walk is still on the stack.) We simplify.

Stack

Now look at combinations using the stack:

\[
S \\
/ \ \\
VP \quad NP \\
/ \ \\
V \quad D \\
/ \ \\
walk \quad the \quad dog \\
\]

We can simplify again.

Stack

Now look at combinations using the stack:

\[
S \\
/ \ \\
VP \quad NP \\
/ \ \\
V \quad D \\
/ \ \\
walk \quad the \quad dog \\
\]

We can simplify again.
In general, we have to postpone consideration of larger, earlier incomplete constituents while we assemble smaller, later constituents.

Consider this example during parsing:

\[ S \rightarrow VP \quad NP \]
\[ VP \rightarrow V \quad NP \]
\[ V \rightarrow \text{answer} \]
\[ NP \rightarrow \text{her} \]
\[ D \rightarrow \text{her} \]
\[ N \rightarrow \text{question} \]

Language is ambiguous:

There's no way to just keep track of the right tree incrementally, as we parse through the string.

We somehow need to try all the alternatives.

Try this combination first, perhaps:

\[ S \rightarrow VP \quad NP \]
\[ VP \rightarrow V \quad NP \]
\[ V \rightarrow \text{answer} \]
\[ NP \rightarrow \text{her} \]
Parsing – search

Now need this combination, somehow:

```
S
  VP  NP   NP
  |   |   D  N
  V  her  
```

answer

Crash!

Parsing – search

Backtrack to the last choice, and consider another possibility:

```
S
  VP  NP   D
  |   |   
  V  her  
```

answer

*Delay answer* – no combo possible here.

Parsing – search

Consider the next word:

```
S
  VP  NP       NP
  |   |   D  N
  V  her  
```

Make the combination.

Parsing – search

Reconsider elements from the stack.

```
S
  VP  NP       NP
  |   |   D  N
  V  her  
```

answer

Consider the next word:

Parsing – chart and efficiency

Backtracking search is expensive, but in fact it makes the parser do the \textit{same work} over and over.

In backtracking, if you change your mind about word \(N\), you have to \textit{reanalyze} everything after \(N\).

But in this reanalysis, you’ll just find the \textit{same} complete constituents starting after \(N\).
Parsing - chart and efficiency

What you can do instead is store all the smaller constituents you find, in a structure called the chart.

Then when you go to build larger constituents, you look in the chart for smaller constituents, rather than searching to derive them.

Exploring possible paths

Imagine guessing the structure of a derivation, top-down.

You can put the midpoint after any word.

After you guess, you break the smaller segments up recursively.

Now imagine putting those pieces together, bottom up

You have all the smaller pieces already.

By running through the possible midpoints, you can find all the ways of putting those pieces together.

Visualization
Parsing algorithm *CKY*

Work your way forward through the string

At each stage, work backwards to build larger substrings

By exploring alternative midpoints

This builds a complete table

At this stage, we have all the elements we need from earlier rounds

we got this when we considered shorter prefixes

we got this earlier in considering this prefix

Summary

For end = 2 up to n
For start = end-2 down to 0
For mid = start+1 up to end-1
Combine (start-mid) (mid-end)

Prolog implementation

Store facts in the knowledge base for constituents, using assert

Use predicate chart(St,End,Tree) for results
Prolog implementation

Setting up for chart parsing:

setup_words(Words) :-
  true if
  for each word W at position P in Words,
  all trees T lexicalized to W have been
  asserted as:
  chart(P, P+1, T).

Prolog implementation

Innermost loop

loop(Start, End) :-
  chart(Start, Mid, T1),
  chart(Mid, End, T2),
  combine(T1, T2, T3),
  assert(chart(Start, End, T3)),
  fail.
  loop(Start, End).

Prolog implementation

Outer loop – edges beginning at Start or
earlier and ending at End.

backward(Start, End) :-
  Start < 0, !.
backward(Start, End) :-
  loop(Start, End),
  Next is Start - 1,
  backward(Next, End).

Prolog implementation

Outer loop – edges ending somewhere between
End and Max.

forward(End, Max) :-
  End > Max, !.
forward(End, Max) :-
  I is End - 2,
  backward(I, End),
  Next is End+1,
  forward(Next, Max).

Prolog implementation

Main parse rule:

parse(Words, T) :-
  retractall(chart(_, _, _)),
  length(Words, N),
  setup_words(Words),
  forward(2, N),
  chart(0, N, T).

Feature structures

How does parsing interact with linguistic
representations? How can you improve
linguistic representations for parsing?

Feature structures provide a way of stating
linguistic constraints concisely and allowing
the parser to collapse together ambiguities.
Feature structures

Motivation

S
   /\NP VP
  /\  /
 /\  /
D N V
 1 1 1
the fish swims

S
   /\NP VP
  /\  /
 /\  /
D N V
 1 1 1
the fish swim

Feature structures

Motivation

S
   /\NP VP
  /\  /
 /\  /
D N V
 1 1 1
the dog swims

S
   /\NP VP
  /\  /
 /\  /
D N V
 1 1 1
the dog swim

Feature structures

English has subject-verb agreement.
number is either singular or plural
person is either first, second or third
nouns are always third person and may be singular or plural
(first person is I/we, second person is you)
verbs in third person present agree with number of the subject

Really English needs different categories of noun phrase for each of these cases.
That means different trees, different parses when different categories are used.
Feature structures

For example

\[
\begin{array}{c}
\text{NP3s} \quad \text{VP} \\
D \quad N \quad V \\
\text{s}\quad \text{dog} \quad \text{swims} \\
\end{array}
\]

\[
\begin{array}{c}
\text{NP3p} \quad \text{VP} \\
D \quad N \quad V \\
\text{p}\quad \text{dogs} \quad \text{swim} \\
\end{array}
\]

Feature structures – notation

\[
\begin{array}{c}
\text{f} \left( \text{number}, \text{sing} \right) \quad \text{dog} \\
\text{f} \left( \text{person}, \text{third} \right) \\
\end{array}
\]

\[
\begin{array}{c}
\text{f} \left( \text{number}, \text{plural} \right) \quad \text{dogs} \\
\text{f} \left( \text{person}, \text{third} \right) \\
\end{array}
\]

Feature structures – data structures

Use prolog terms to represent feature structures:

\[
\begin{array}{c}
f(\text{Number}, \text{Person}) \\
\text{Use unification to enforce feature constraints.} \\
\text{Examples:} \\
f(\text{sing, third}) \quad \text{dog} \\
f(\text{plural, third}) \quad \text{dogs} \\
f(\text{sing, third}) \quad \text{swims} \\
\end{array}
\]

Another idea – categories as complex data structures

Use terms for categories, and allow variables and unification.

\[
\begin{array}{c}
\text{np}(3,N) \quad \text{vp} \\
\text{d} \quad \text{n} \quad \text{v} \\
\text{the} \quad \text{fish} \quad \text{swim} \\
\end{array}
\]
Semantics and interpretation

How does parsing interact with linguistic representations? How can you improve linguistic representations for parsing?

Adding semantic variables to our syntactic representations

Examples:

Adding semantic variables to our syntactic representations

Examples:

Semantic compositionality

Conjoining formulas and unifying variables:

Incremental interpretation

As we parse, we can keep track of which assignments to semantic variables are compatible with a knowledge base:

Incremental interpretation

As we parse, we can keep track of which assignments to semantic variables are compatible with a knowledge base:
Incremental interpretation – data structures

We can insert semantic constraints and their interpretations into our chart entries:

\[
\text{chart} \left( \text{St}, \text{End}, \text{Tree}, \text{Constr}, \text{Interp} \right)
\]

Example:

\[
\begin{align*}
\text{Constr} &= \left[ \text{dog}(X), \text{swims}(X) \right] \\
\text{Interp} &= \left[ \left[ \text{dog(spot)}, \text{swims(spot)} \right] \right]
\end{align*}
\]

Prolog implementation

Updated innermost loop:

\[
\text{loop} \left( \text{Start}, \text{End} \right) :-
\begin{align*}
&\text{chart} \left( \text{Start}, \text{Mid}, \text{Tree}_1, \text{Constr}_1, \text{Interp}_1 \right), \\
&\text{chart} \left( \text{Mid}, \text{End}, \text{Tree}_2, \text{Constr}_2, \text{Interp}_2 \right), \\
&\text{combine} \left( \text{Tree}_1, \text{Tree}_2, \text{Tree}_3 \right), \\
&\text{append} \left( \text{Constr}_1, \text{Constr}_2, \text{Constr}_3 \right), \\
&\text{findall} \left( \text{Constr}_3, \left( \text{member} \left( \text{Constr}_1, \text{Interp}_1 \right), \\
&\text{member} \left( \text{Constr}_2, \text{Interp}_2 \right) \right), \text{Interp}_3 \right), \\
&\text{assert} \left( \text{chart} \left( \text{Start}, \text{End}, \text{Tree}_3, \text{Constr}_3, \text{Interp}_3 \right) \right), \\
&\text{fail}. \\
\end{align*}
\]

Adding statistical information

Consider the sentence: "Fruit flies like a banana."

\[
\begin{align*}
\text{Adj} &\rightarrow \text{N} \quad \text{V} \\
\text{fruit} &\rightarrow \text{flies} \\
\text{like} &\rightarrow \text{a banana}
\end{align*}
\]

Adding statistical information

We can associate probabilities with each step in a derivation:

\[
\begin{align*}
\text{Adj} &\rightarrow \text{NP} \quad \text{V} \\
\text{fruit} &\rightarrow \text{flies} \\
\text{like} &\rightarrow \text{a banana}
\end{align*}
\]

Adding statistical information

We can associate probabilities with each step in a derivation:

\[
\begin{align*}
\text{NP} &\rightarrow \text{Adj} \\
\text{N} &\rightarrow \text{V} \\
\text{NP} &\rightarrow \text{PP} \\
\text{NP} &\rightarrow \text{NP}
\end{align*}
\]

Adding statistical information

The probability of each step in a derivation can be estimated from a corpus of correct derivations.

A probability score is associated with each chart entry according to the probabilities of the steps needed to derive it.

These probabilities may be used to guide the search for a complete parse.