Natural Language Understanding

Henry Kautz

CSE P573, Autumn 2004

Tonight

- Lists in Prolog
- Overview of natural language understanding
- Parsing sentences
- Generating the logical form of a sentence

Lists

- Lists are the same as other languages (such as ML) in that a list of terms of any length is composed of list cells that are 'consed' together.
- The list of length 0 is called nil, written [].
- The list of length *n* is .(*head*,*tail*), where *tail* is a list of length *n*–1.
- So a list cell is a functor '.' of arity 2. Its first component is the head, and the second component is the tail.

Examples of lists

nil

.(a, nil)

.(a, .(b, nil)

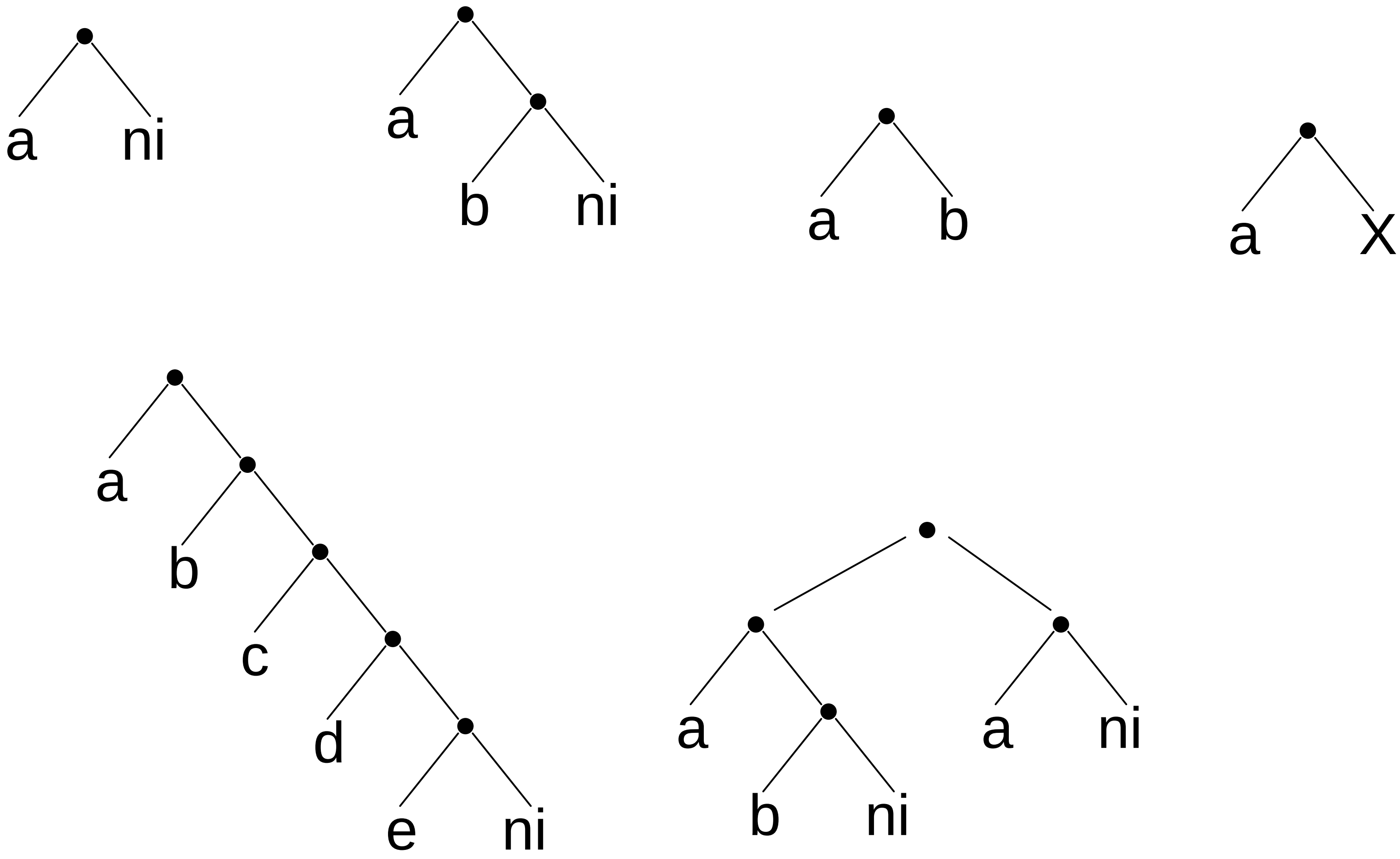
.(a, .(b, .(c, .(d, .(e. nil)))))

.(a,b) *(note this is a pair, not a proper list)*

.(a, X) *(this might be a list, or might not!)*

.(a, .(b, nil)), .(c, nil))

They can be written as trees



Prolog Syntax for Lists

Nil is written [].

The list consisting of n elements t_1, t_2, …,t_n is written [t_1, t_2, …,t_n].

.(X,Y) is written [X|Y]

[X|[]] is written [X]

The term .(a, .(b, .(c,Y))) is written [a,b,c|Y].

If Y is instantiated to [], then the term is a list, and can be written [a,b,c|[]] or simply [a,b,c].

Exercises

Identify the heads and tails of these lists (if any):

[a, b, c]

[a]

[]

[[the, cat], sat]

[[the, cardinal], [pulled, [off]], [each, [plum, coloured], shoe]

Exercises

Identify the heads and tails of these lists (if any):

a [b, c]

[a]

[]

[[the, cat], sat]

[[the, cardinal], [pulled, [off]], [each, [plum, coloured], shoe]

Exercises

Identify the heads and tails of these lists (if any):

[a, b, c]

a   []

[]

[[the, cat], sat]

[[the, cardinal], [pulled, [off]], [each, [plum, coloured], shoe]

Exercises

Identify the heads and tails of these lists (if any):

[a, b, c]

[a]

[] *(not a list, so doesn't have head and tail. nil is a constant)*

[[the, cat], sat]

[[the, cardinal], [pulled, [off]], [each, [plum, coloured], shoe]

Exercises

Identify the heads and tails of these lists (if any):

[a, b, c]

[a]

[]

[the, cat] [sat]

[[the, cardinal], [pulled, [off]], [each, [plum, coloured], shoe]

Exercises

Identify the heads and tails of these lists (if any):

[a, b, c]

[a]

[]

[[the, cat], sat]

[the, cardinal] [pulled, [off]], [each, [plum, coloured], shoe]

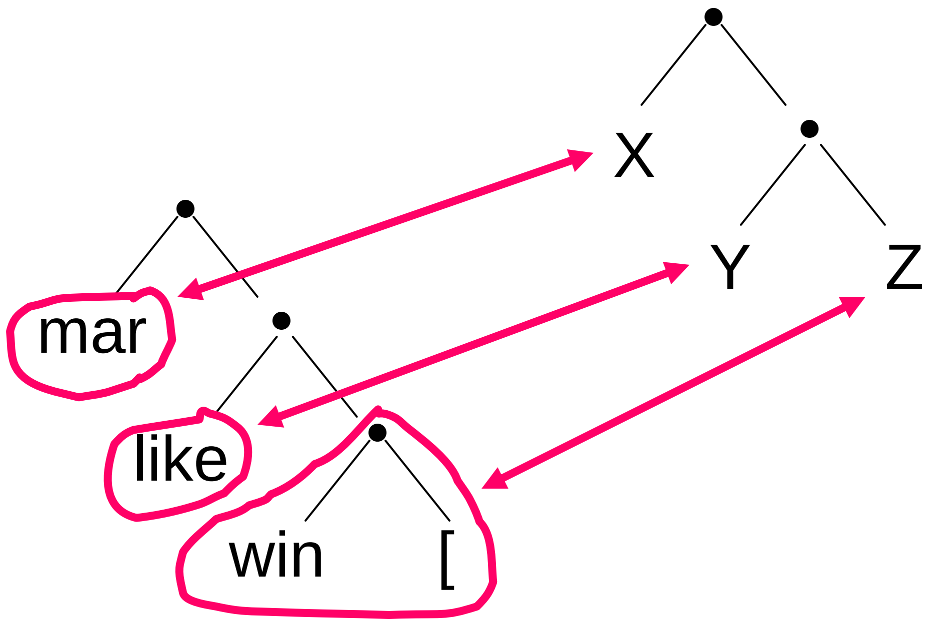
Exercises

For each pair of terms, determine whether they unify, and if so, to which terms are the variables instantiated?

[X, Y, Z]	[john, likes, fish]
[cat]	[X|Y]
[X,Y|Z]	[mary, likes, wine] *(picture on next slide)*
[[the,Y]|Z]	[[X,answer], [is, here]]
[X, Y, X]	[a, Z, Z]
[[X], [Y], [X]]	[[a], [X], [X]]

Remember

A variable may be instantiated to any term.



[mary, likes, wine]

[X,Y|Z]

Fun with Lists (Worksheet 5)

```
/* member(Term, List) */
member(X, [X|T]).
member(X, [H|T]) :- member(X, T).
```

Examples:

```
?- member(john, [paul, john]).
?- member(X, [paul, john]).
?- member(joe, [marx, darwin, freud]).
?- member(foo, X).
```

Communication

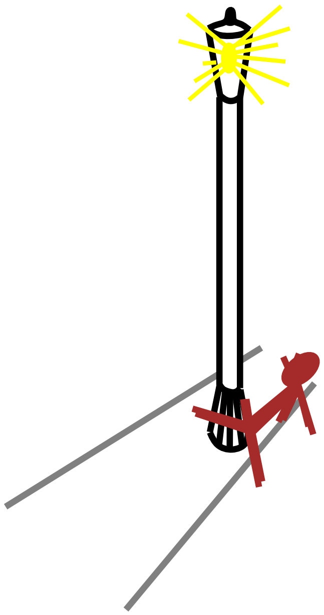
"Classical" view (pre-1953):
language consists of sentences that are true/false (cf. logic)

"Modern" view (post-1953):
language is a form of action

Wittgenstein (1953) **Philosophical Investigations**
Austin (1962) **How to Do Things with Words**
Searle (1969) **Speech Acts**

Why?

To change the actions of other agents



Speech acts

Speech acts achieve the speaker's goals:

Inform	"There's a pit in front of you"
Query	"Can you see the gold"
Command	"Pick it up"
Promise	"I'll share the gold with you"
Acknowledge	"OK"

Speech act planning requires knowledge of

– Situation

– Semantic and syntactic conventions

– Hearer's goals, knowledge base, and rationality

Stages in communication (informing)

Intention	S wants to inform H that P
Generation	S selects words W to express P
Synthesis	S utters words W
Perception	H perceives W'
Analysis	H infers possible meanings $P_1, \ldots P_n$
Disambiguation	H infers intended meaning P_i
Incorporation	H incorporates P_i into KB

How could this go wrong?

- Insincerity (S doesn't believe P)
- Speech wreck ignition failure
- Ambiguous utterance
- Differing understanding of current situation

Grammar

Vervet monkeys, antelopes etc. use isolated symbols for sentences

$\Rightarrow$ restricted set of communicable propositions, no generative capacity

(Chomsky (1957): **Syntactic Structures**)

Grammar specifies the compositional structure of complex messages

e.g., speech (linear), text (linear), music (two-dimensional)

A formal language is a set of strings of terminal symbols

Each string in the language can be analyzed/generated by the grammar

The grammar is a set of rewrite rules, e.g.,

$S \rightarrow NP\ VP$

$Article \rightarrow \boldsymbol{the} \mid \boldsymbol{a} \mid \boldsymbol{an} \mid \ldots$

Here S is the sentence symbol, NP and VP are nonterminals

Grammar types

Regular: $nonterminal \rightarrow \boldsymbol{terminal}[nonterminal]$

$S \rightarrow \boldsymbol{a}S$
$S \rightarrow \Lambda$

Context-free: $nonterminal \rightarrow anything$

$S \rightarrow \boldsymbol{a}S\boldsymbol{b}$

Context-sensitive: more nonterminals on right-hand side

$ASB \rightarrow AA\boldsymbol{a}BB$

Recursively enumerable: no constraints

Related to Post systems and Kleene systems of rewrite rules

Natural languages probably context-free, parsable in real time!

Wumpus lexicon

$$
\begin{aligned}
\textit{Noun} &\rightarrow \boldsymbol{stench} \mid \boldsymbol{breeze} \mid \boldsymbol{glitter} \mid \boldsymbol{nothing} \\
&\quad \mid \boldsymbol{wumpus} \mid \boldsymbol{pit} \mid \boldsymbol{pits} \mid \boldsymbol{gold} \mid \boldsymbol{east} \mid \ldots \\
\textit{Verb} &\rightarrow \boldsymbol{is} \mid \boldsymbol{see} \mid \boldsymbol{smell} \mid \boldsymbol{shoot} \mid \boldsymbol{feel} \mid \boldsymbol{stinks} \\
&\quad \mid \boldsymbol{go} \mid \boldsymbol{grab} \mid \boldsymbol{carry} \mid \boldsymbol{kill} \mid \boldsymbol{turn} \mid \ldots \\
\textit{Adjective} &\rightarrow \boldsymbol{right} \mid \boldsymbol{left} \mid \boldsymbol{east} \mid \boldsymbol{south} \mid \boldsymbol{back} \mid \boldsymbol{smelly} \mid \ldots \\
\textit{Adverb} &\rightarrow \boldsymbol{here} \mid \boldsymbol{there} \mid \boldsymbol{nearby} \mid \boldsymbol{ahead} \\
&\quad \mid \boldsymbol{right} \mid \boldsymbol{left} \mid \boldsymbol{east} \mid \boldsymbol{south} \mid \boldsymbol{back} \mid \ldots \\
\textit{Pronoun} &\rightarrow \boldsymbol{me} \mid \boldsymbol{you} \mid \boldsymbol{I} \mid \boldsymbol{it} \mid \ldots \\
\textit{Name} &\rightarrow \boldsymbol{John} \mid \boldsymbol{Mary} \mid \boldsymbol{Boston} \mid \boldsymbol{UCB} \mid \boldsymbol{PAJC} \mid \ldots \\
\textit{Article} &\rightarrow \boldsymbol{the} \mid \boldsymbol{a} \mid \boldsymbol{an} \mid \ldots \\
\textit{Preposition} &\rightarrow \boldsymbol{to} \mid \boldsymbol{in} \mid \boldsymbol{on} \mid \boldsymbol{near} \mid \ldots \\
\textit{Conjunction} &\rightarrow \boldsymbol{and} \mid \boldsymbol{or} \mid \boldsymbol{but} \mid \ldots \\
\textit{Digit} &\rightarrow \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \mathbf{4} \mid \mathbf{5} \mid \mathbf{6} \mid \mathbf{7} \mid \mathbf{8} \mid \mathbf{9}
\end{aligned}
$$

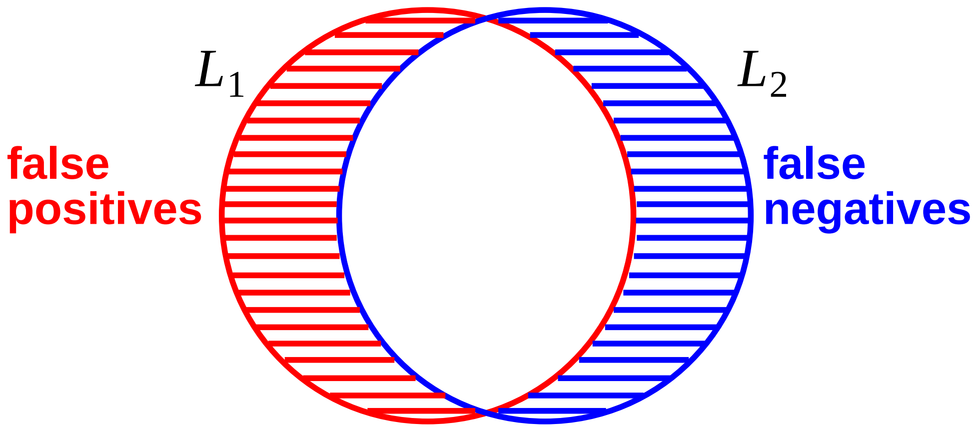
Divided into closed and open classes

Wumpus grammar

S	$\rightarrow$	$NP\ VP$	I + feel a breeze
	$\mid$	$S\ Conjunction\ S$	I feel a breeze + and + I smell a wumpus
NP	$\rightarrow$	$Pronoun$	I
	$\mid$	$Noun$	pits
	$\mid$	$Article\ Noun$	the + wumpus
	$\mid$	$Digit\ Digit$	3 4
	$\mid$	$NP\ PP$	the wumpus + to the east
	$\mid$	$NP\ RelClause$	the wumpus + that is smelly
VP	$\rightarrow$	$Verb$	stinks
	$\mid$	$VP\ NP$	feel + a breeze
	$\mid$	$VP\ Adjective$	is + smelly
	$\mid$	$VP\ PP$	turn + to the east
	$\mid$	$VP\ Adverb$	go + ahead
PP	$\rightarrow$	$Preposition\ NP$	to + the east
$RelClause$	$\rightarrow$	***that*** VP	that + is smelly

Grammaticality judgements

Formal language L_1 may differ from natural language L_2



Adjusting L_1 to agree with L_2 is a learning problem!

* the gold grab the wumpus
* I smell the wumpus the gold
   I give the wumpus the gold
* I donate the wumpus the gold

Intersubjective agreement somewhat reliable, independent of semantics!
Real grammars 10–500 pages, insufficient even for "proper" English

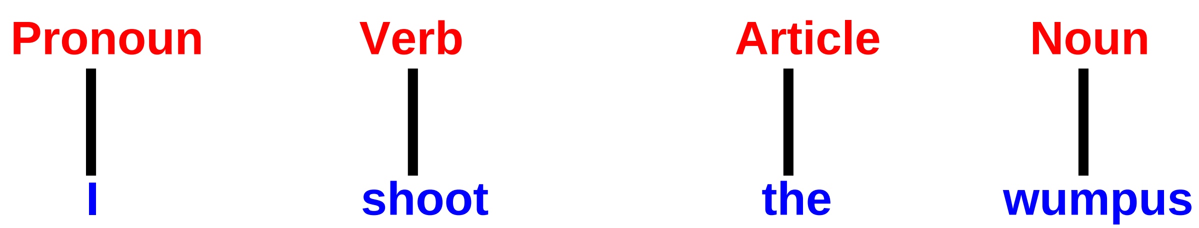
Parse trees

Exhibit the grammatical structure of a sentence

I shoot the wumpus

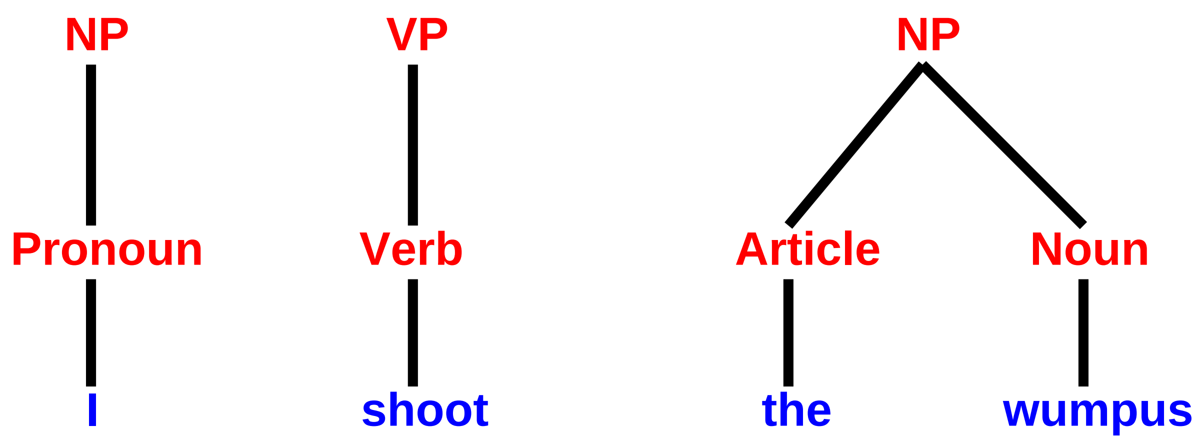
Parse trees

Exhibit the grammatical structure of a sentence



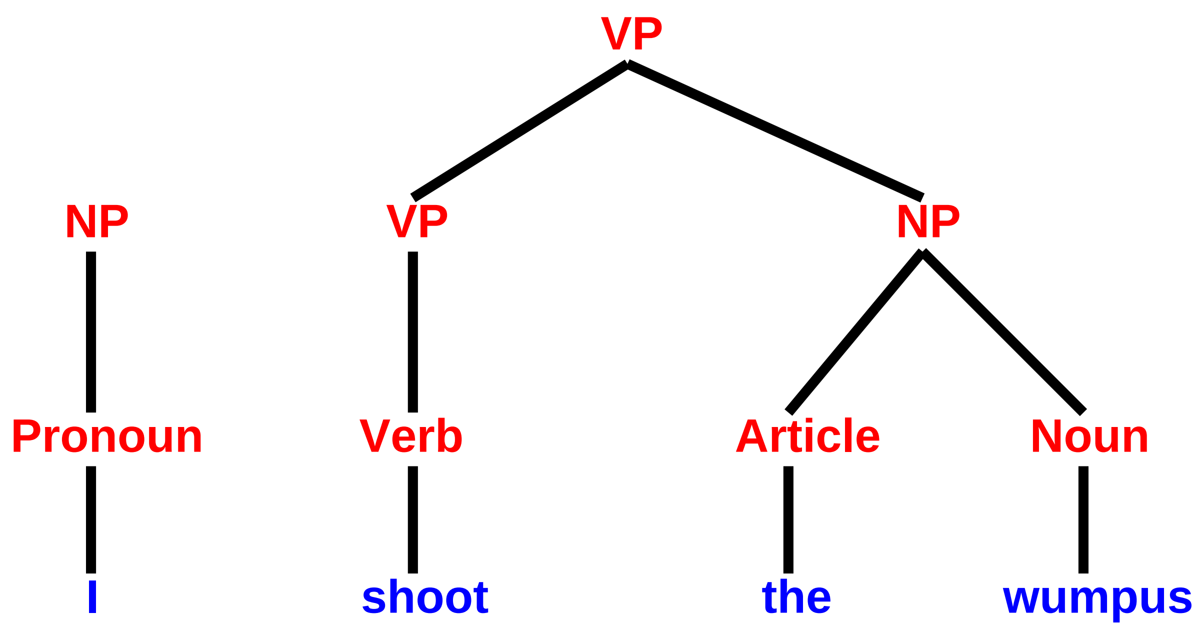
Parse trees

Exhibit the grammatical structure of a sentence



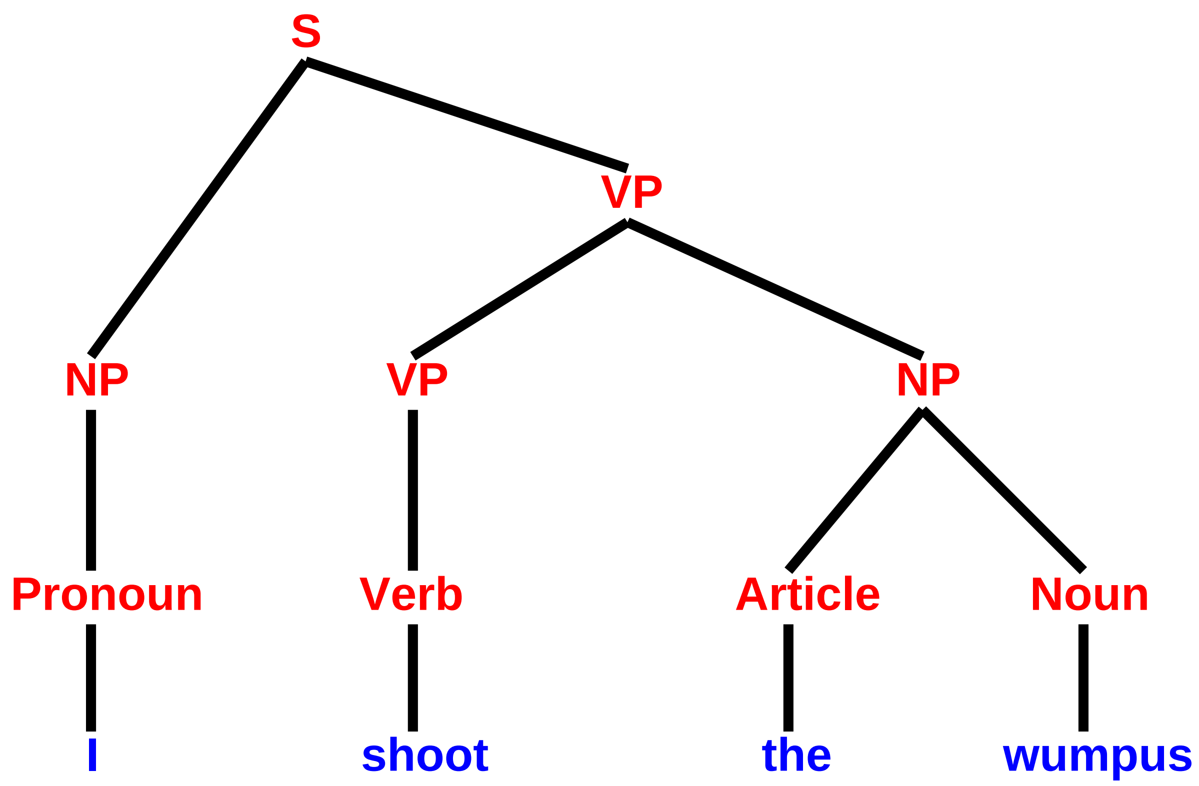
Parse trees

Exhibit the grammatical structure of a sentence



Parse trees

Exhibit the grammatical structure of a sentence



2 A formal grammar for English

We can describe simple sentences of English with the following grammar:

$$
\begin{array}{rcl}
\mathrm{S} & \rightarrow & \mathrm{NP}, \mathrm{VP} \\
\mathrm{NP} & \rightarrow & \mathrm{Det}, \mathrm{N} \\
\mathrm{VP} & \rightarrow & \mathrm{V}, \mathrm{NP} \\
\mathrm{VP} & \rightarrow & \mathrm{V} \\
\mathrm{Det} & \rightarrow & \mathrm{the} \\
\mathrm{N} & \rightarrow & \mathrm{apple} \\
\mathrm{N} & \rightarrow & \mathrm{man} \\
\mathrm{V} & \rightarrow & \mathrm{eats} \\
\mathrm{V} & \rightarrow & \mathrm{sings}
\end{array}
$$

3 Writing a parser for this grammar in Prolog

For simplicity, we suppose that we are trying to parse a list of Prolog atoms.

We will write Prolog predicates like:

- `sentence(Sentence)`, which will be true if `Sentence` is a list of words which can be parsed as a sentence,
- `det(Det)`, which will be true if `Det` is a list of words which can be parsed as a determiner,
- and so on.

The structure of our Prolog program will reflect closely the structure of the grammar.

3.1 Sentences

Our grammar tells is that a list of atoms will be a sentence if it can be split up into a list of atoms which can be parsed as a noun phrase followed by a list of atoms which can be parsed as a verb phrase.

We know how to split lists up: we use `append`.

So we have:

```
sentence(Sentence) :-
     append(NounPhrase, VerbPhrase, Sentence),
     nounphrase(NounPhrase),
     verbphrase(VerbPhrase).
```

And this is more-or-less all there is to it!

```
nounphrase(Nounphrase) :-
    append(Det, Noun, Nounphrase),
    det(Det),
    noun(Noun).

verbphrase(Verbphrase) :-
     verb(Verbphrase).

verbphrase(Verbphrase) :-
    append(Verb, Nounphrase, Verbphrase),
    verb(Verb),
    nounphrase(Nounphrase).

noun([man]). noun([apple]). verb([eats]). verb([sings]).
det([the]).
```

And now we can ask queries like:

```
?- sentence([the, man, eats, the, apple])

yes

?- sentence([the, apple])

no

?- nounphrase([the, apple])

yes
```

3.2 Comments

This code use a “generate-and-test” strategy:

- generate possible solutions (the different splittings of the input list);
- test them to see if they are suitable.

This is a common strategy when solving AI problems, and is very useful. However the code we have written generates a lot of possible solutions which turn out to be incorrect, so this is not a very good algorithm.

Can we find a more efficient algorithm?

3.3 Finding a more efficient algorithm

We know that we can often make “generate-and-test” more efficient by pushing the test closer to the generation. How can we do this in the current situation?

We do this by letting predicates like `noun` perform both the recognition *and* the splitting. We do this by letting them accept the front of a list, and return the rest of the list.

```
sentence(Tokens, Rest) :-
     nounphrase(Tokens, More),
     verbphrase(More, Rest).
```

```
nounphrase(Tokens, Rest) :-
    det(Tokens, More),
    noun(More, Rest).

verbphrase(Tokens, Rest) :-
     verb(Tokens, Rest).

verbphrase(Tokens, Rest) :-
    verb(Tokens, More),
    nounphrase(More, Rest).

noun([man | Rest], Rest). noun([apple | Rest], Rest).
verb([eats | Rest], Rest). verb([sings | Rest], Rest).
det([the | Rest], Rest).
```

We can now ask queries like:

```
sentence([the, man, eats, the, apple], Rest)

No.1 : Rest = [the, apple]

No.2 : Rest = []

No more solutions
```

4 Extending the grammar

As is usual we are not interested merely in whether a string parses, but in the *parse tree* that is constructed. Prolog lets us build up a parse tree very easily.

We augment the predicates with an extra argument like:

```
sentence(Tokens, Rest, sentence(NP, VP)) :-
     nounphrase(Tokens, More, NP),
     verbphrase(More, Rest, VP).
```

```
nounphrase(Tokens, Rest, np(Det, N)) :-
    det(Tokens, More, Det),
    noun(More, Rest, N).


verbphrase(Tokens, Rest, iv(Verb)) :-
     verb(Tokens, Rest, Verb).


verbphrase(Tokens, Rest, tv(Verb, NP)) :-
    verb(Tokens, More, Verb),
    nounphrase(More, Rest, NP).

noun([man | Rest], Rest, man). noun([apple | Rest], Rest, apple).
verb([eats | Rest], Rest, eats). verb([sings | Rest], Rest, sings).
det([the | Rest], Rest, the).
```

Now we can ask;

```
?- sentence([the, man, eats, the, apple], Rest, Tree)

No.1 : Rest = [the, apple],
       Tree = sentence(np(the, man), iv(eats))

No.2 : Rest = [],
       Tree = sentence(np(the, man), tv(eats, np(the, apple)))

No more solutions
```

Recall that we can think of Prolog as programming with *relations*. Hence we can also ask queries like:

```
?- sentence(Words,
            [],
            sentence(np(the, man), tv(eats, np(the, apple))))

No.1 : Words = [the, man, eats, the, apple]
```

Exercise

- In groups (no lone wolves!):
  - download directory /cse/courses/csep573/04au/prolog
  - launch Prolog
  - consult(genesis_syntax).
  - Try some examples:
    - sentence([Eve,gives,Adam,the,apple], [], Tree).

Sentence Meaning

- In many applications we want to know the meaning of a sentence, not just its parse tree
- We can do this in Prolog by adding an additional argument for the logical form

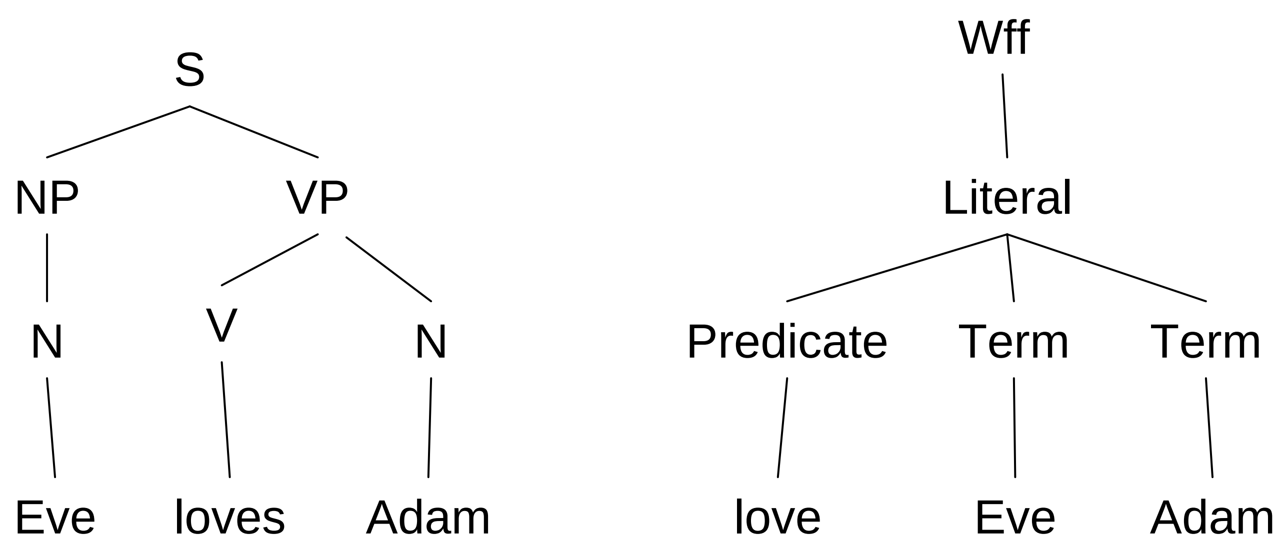
  sentence(Tokens, Rest, s(NP, VP), Wff)

Logical Form

- There is no one “right” way to define the logical form of a sentence
- One approach:
  - Nouns become Terms
  - Verbs and Adjectives become Predicates
- Example:
  - “Eve loves Adam” $\Rightarrow$ love(eve,adam)
  - “Adam is loved by Eve” $\Rightarrow$ love(eve,adam)

Semantic Structure

- Issue: the structure of the logical form may be different from the parse tree
- How then can it be created while parsing?



The predicate word “loves” is buried inside the VP

Compositionality

- One approach: make logical structure the same as parse tree, by using lamda-expressions (R. Montague 1973)
- Our approach: use additional Prolog variables to move pieces of structure around

Moving Subject Into the Predicate

```
sentence(Tokens, Rest, s(NP, VP), Wff) :-
    nounphrase(Tokens, More, NP, Term),
    verbphrase(More, Rest, VP, Term, Wff).

verbphrase(Tokens, Rest, tvp(Verb, NP),
  Term1, [Predicate,Term1,Term2]) :-
      verb(Tokens, More, Verb, Predicate),
      nounphrase(More, Rest, NP, Term2).
```

Definite Clause Grammars

- All those “Tokens, Rest” variables make the grammar hard to read
- Definite Clause Grammar notation adds them automatically.

```
sentence(s(NP, VP), Wff)  -->
    nounphrase(NP, Term),
    verbphrase(VP, Term, Wff).
```

Exercise

- Back to Prolog…
  - consult(genesis_dcg).
  - Try some examples:
    - phrase(sentence(T,M),[Eve,gives,Adam,the,apple]).
  - consult(toplevel).
  - Try the read/eval loop (blank line or error to exit):
    - nlp.
    - > Eve gives Adam the apple
    - > Eve loves the snake

Workshop

- In your groups, extend genesis_dcg.pl so that it handles sentences that contain “and” or “or” joining independent sentences:
  - Adam loves Eve and Eve loves the snake
  - compound_s(
    ```
    s(np(n(adam)), tvp(v(loves),
                        np(n(eve)))),
    c(and),
    s(np(n(eve)), tvp(v(loves),
                       np(det(the),
                          n(snake)))))
    ```
  - [and, [love,Adam, Eve], [love, Eve, satan]]

Conjoined Objects

```
verbphrase(tvp(Verb,
   compound_np(NP1,C,NP2)), Term1,
   [Operator, [Predicate,Term1,Term2],
               [Predicate,Term1,Term3]]) -->
     verb(Verb, Predicate),
     nounphrase(NP1, Term2),
     connective(C, Operator),
     nounphrase(NP2, Term3).
```

Exercise

- Back to Prolog…
  - consult(genesis_compound).
  - nlp.
  - > Eve loves Adam and the snake.
  - > Adam and Eve eat the apple and the snake.
  - ERROR
  - CHANGE: verb(v(eats),eat) --> [eats | eat].

Referring Expression

- Determining the object being referred to by a noun phrase can require taking into account syntactic, semantic, and pragmatic (contextual) information
- Pronouns are an obvious case:
  - I put my quarter in the vending machine but it was broken.
  - I put my quarter in the vending machine but it was bent.

Referring Expressions

- In truth, contextual information is needed to determine the referent of any noun:
  - Henry teaches P573.
  - Henry became king in 1399 AD.
  - When I got married, the minister was tipsy.
- Some applications can handle nouns by defining predicates that search for the object.

Example

Instead of:

```
noun(n(man),adam) --> [man].
```

Define:

```
noun(n(man),P) -->
            [man], {male(P), in_focus(P)}.
male(adam).
male(cain).
in_focus(P) :- ordered list of most recently
                         mentioned objects
```

Disambiguation

- Reaction time experiments show the brain disambiguates language on a word-by-word basis, not by whole sentences or phrases
  - "The gardener found a bug in the petunias"

recognize MICROPHONE: fast
recognize INSECT: fast

recognize MICROPHONE: slow
recognize INSECT: fast

Disambiguation

- Reaction time experiments show the brain disambiguates language on a word-by-word basis, not by whole sentences or phrases
  - "The spy found a bug in the ceiling"

recognize MICROPHONE: fast
recognize INSECT: fast

recognize MICROPHONE: fast
recognize INSECT: slow

Practical Disambiguation

- The most successful approaches to disambiguation for NLP use tables of word trigram frequency
  - {gardener, saw, bug[Noun,Insect]}   0.0002
  - {spy, saw, bug[Noun,Insect]}   0.0000001
- Approach: use trigrams to tag each word in sentence, then parse

What’s in a Tag?

- Understanding
  - Tag = {part of speech, meaning}
  - Main problem: limited amount of fully-tagged data for creating trigram tables
- Parsing
  - Tag = {part of speech}
  - Much more data available
- Speech recognition
  - No tag needed (just predict next word)
  - Limitless amounts of data available

Garden-Path Sentences

- “Leading someone down the garden path” = “Leading someone astray without them being aware of it”
- Is this proper English?

  The horse raced past the barn fell.
- Ordinary language understanding relies upon our implicit knowledge of language
- Cases like these require conscious thought

Assignment

- Either:
  - Enrich genesis_compound.pl to handle more kinds of statements
- OR
  - Generalize genesis_parser.pl to handle a paragraph of real-world text
- Work alone or in groups. Feel free to share questions and ideas with the class on csep573@cs.washington.edu