Issues in Computational Linguistics: Parsing and Generation

Dick Crouch and Tracy King

parc
Palo Alto Research Center

Course Purpose

- To introduce linguistics students and theoretically-oriented LFG researchers to high-level issues in computational linguistics
- To introduce computer science students and researchers to computational linguistics from an LFG perspective

parc
Palo Alto Research Center

Course Outline

- Parsing and generation (Day 1)
- Grammar engineering (Day 2)
- Disambiguation and packing (Day 3)
- Semantics (Day 4)

Parsing and Generation

- Parsing
  - Context-free parsing
  - Unification
- Generation and reversibility

What is parsing?

- Have a natural language string (sentence)
- Want to produce a structure/analysis for it
- Applications use this structure as input

Types of structures for parse analyses

- Phrase structure trees (LFG c-structure)
- Feature attribute value matrices (LFG f-structure)

- Shallow parsing
  - Part of speech tagging

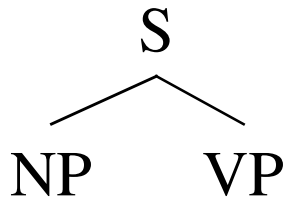
    I/PRP saw/VBD her/PRP duck/VB ./PUNCT
  - Chunking

    [when I read] [a sentence], [I read it] [a chunk] [at a time].

parc
Palo Alto Research Center

Phrase structure trees

- Constituency (and type of constituent)
  - NPs, VPs, Ss
- Dominance
  - S dominates NP and VP
- Linear order
  - NP precedes VP



Context free grammar

- Set of rewrite rules
  - NP --> Det Nbar
  - NP --> ProperNoun
  - Nbar --> Noun
  - Nbar --> Adj Noun
- Lexicon of "leaves"
  - Det --> a
  - Det --> the
  - Noun --> box
- Start symbol
  - S

Parsing with CFG

- Have a string
- Want a tree spanning it
- Need:
  - a grammar
  - a way to apply the grammar to the string
    - Top down parsing
    - Bottom up parsing
    - Chart parsing

Top down parsing with a CFG

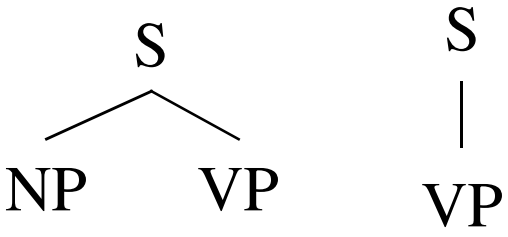
- Start with start symbol (e.g. S)
- Expand all possible ways (1st “ply”)
- Expand the constituents of the new trees
- Once get to parts of speech, map against the words
- Reject trees that don’t match

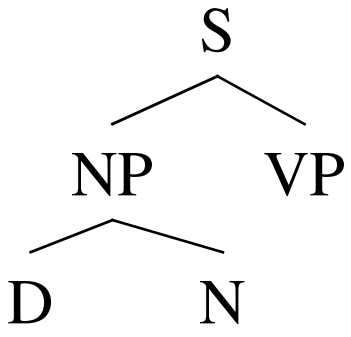
Example CFG rules and parse

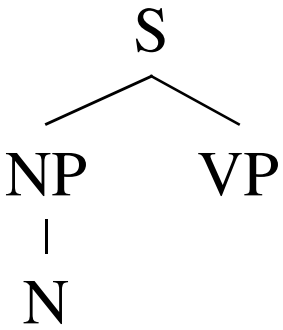
S -> NP VP
S -> VP
NP -> D N
NP -> N
VP -> V NP

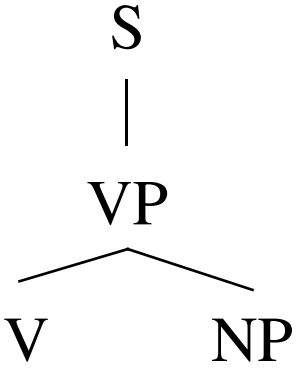
V -> push
N -> push
N -> boxes
D -> the

String: *Push the boxes*









parc
Palo Alto Research Center

Problems: top down parsing

- Generate many trees inconsistent with the input
- Duplicate work by rebuilding constituents

Bottom up parsing with a CFG

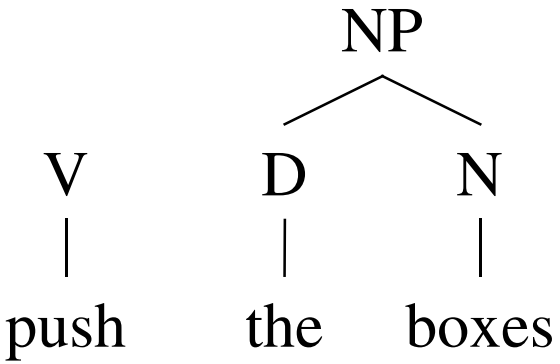
- Apply rules to words in string
- First do part of speech rules
- Expand these upwards
- Reject any trees that do not top out with the start symbol

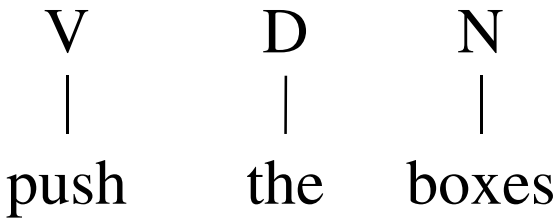
Example CFG rules and parse

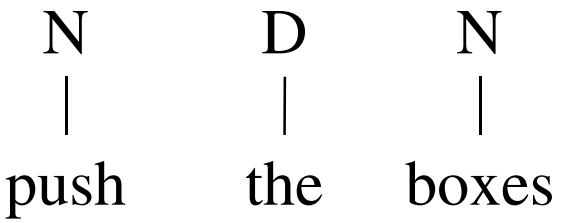
S -> NP VP
S -> VP
NP -> D N
NP -> N
VP -> V NP

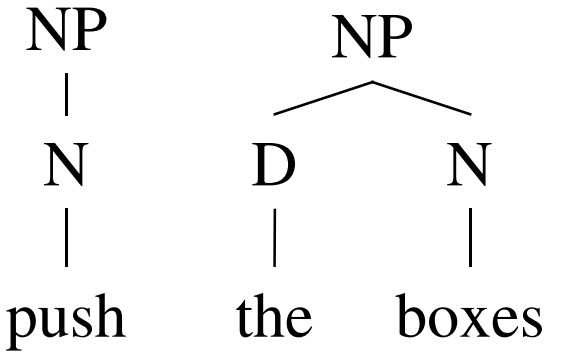
V -> push
N -> push
N -> boxes
D -> the

String: *Push the boxes*









parc
Palo Alto Research Center

Problems: Bottom up parsing

- Explore trees that cannot result in an S
- Duplicate work by rebuilding constituents

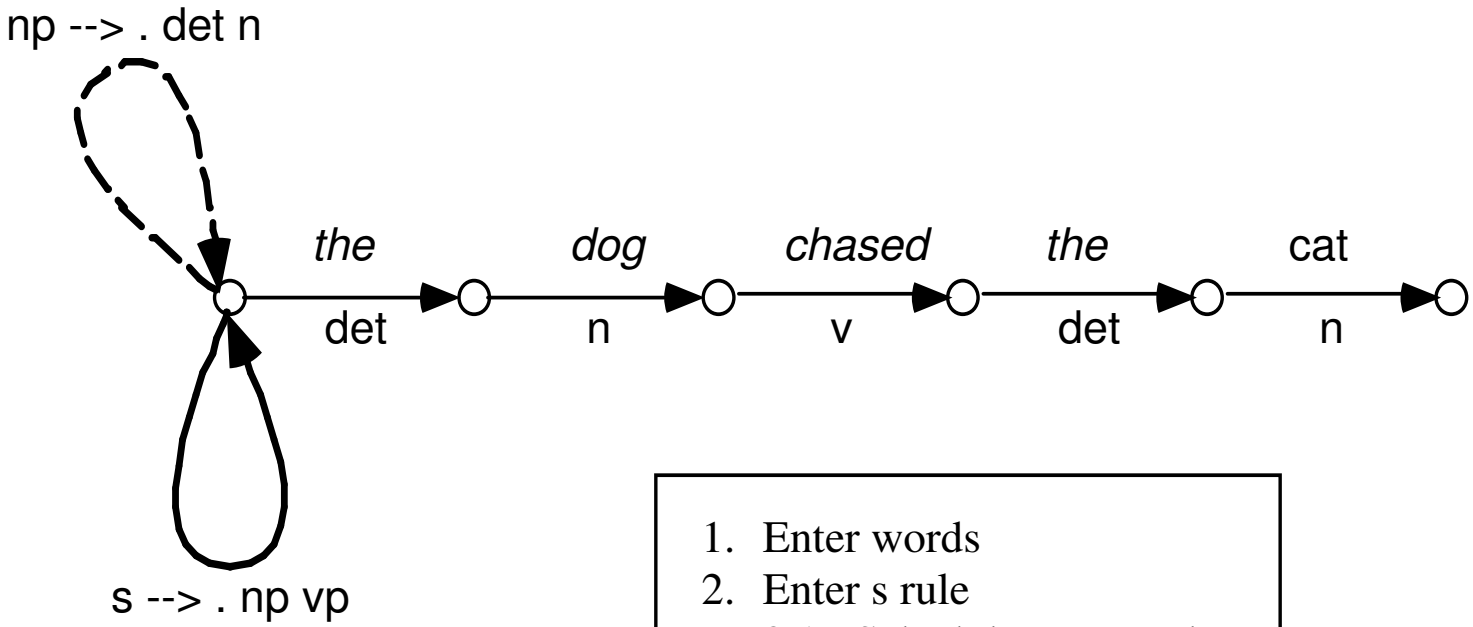
Chart Parsing: Earley algorithm

- Parallel top down search
- Eliminate repetitive solution of subtrees
- Left-right pass creates an array/chart
- At the end of the string, chart encodes all the possible parses
  - each subtree only represented once

Chart Processing

- Look words up in the lexicon. Create an edge for each entry and put it on the agenda.
- Find all words in the lexicon that are substrings of the input specification
- Until the agenda is empty, remove edges from it and put them in the chart
  - Active edge added to chart: schedule to combine with inactive edges to its right
  - Inactive edge added to chart: schedule to combine with active edge to its left
- Solutions are edges that cover all the input.

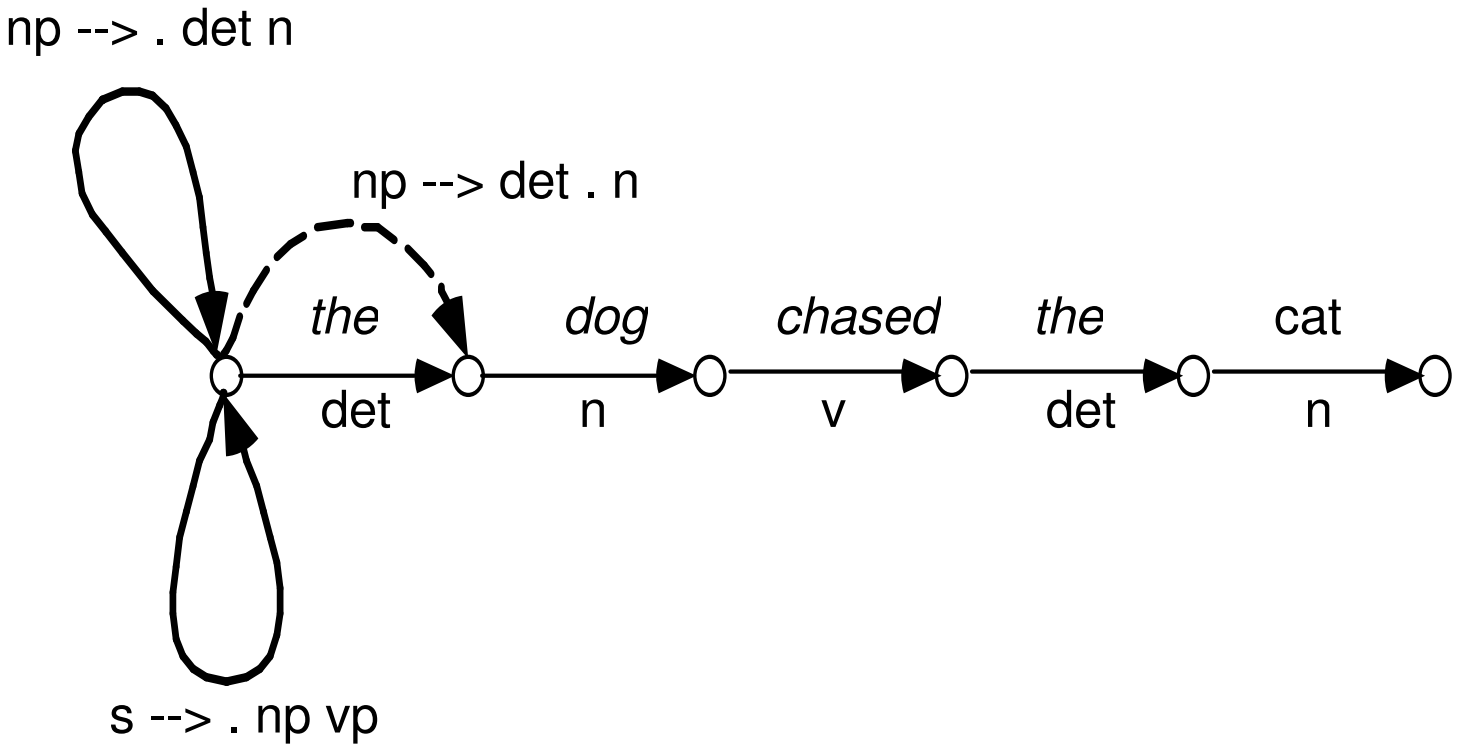
The Initial Situation



1. Enter words
2. Enter s rule
   2.1. Schedule np --> . det n

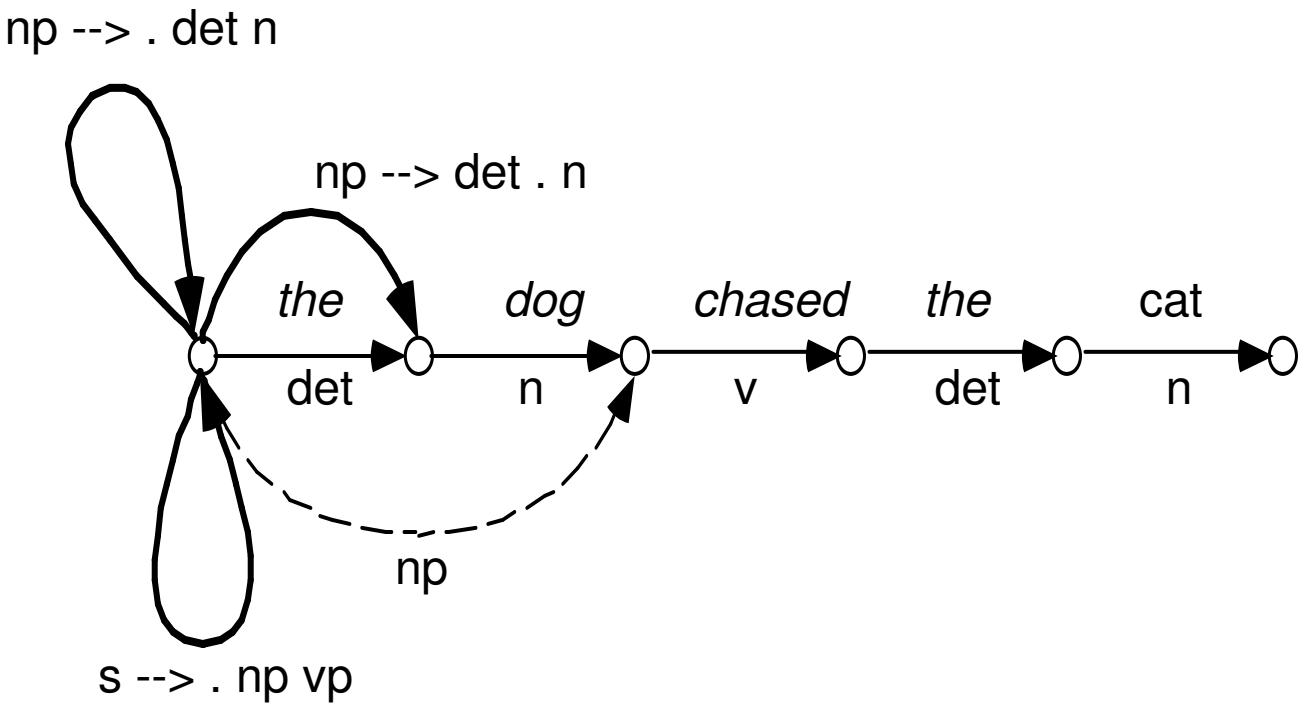
parc
Palo Alto Research Center

Step 1



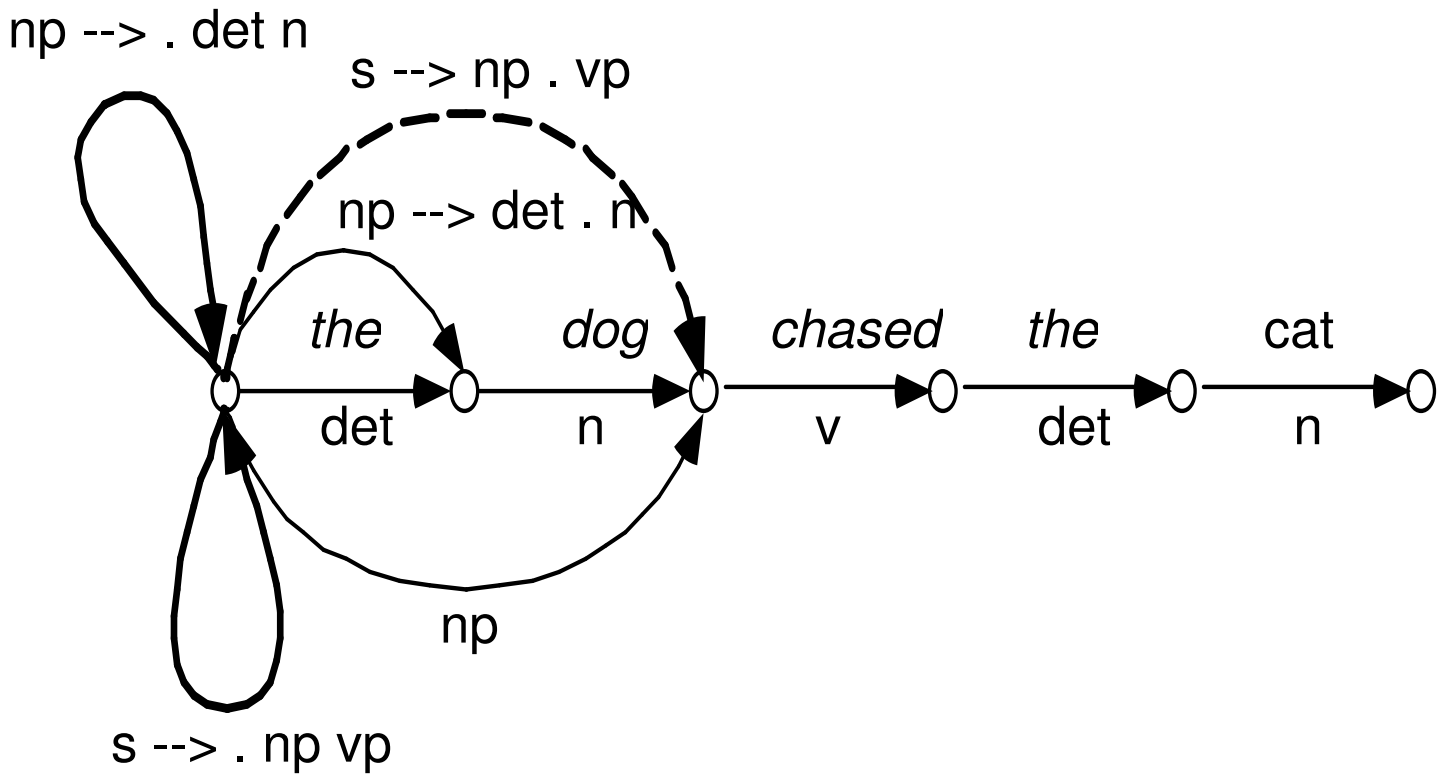
parc
Palo Alto Research Center

Step 2

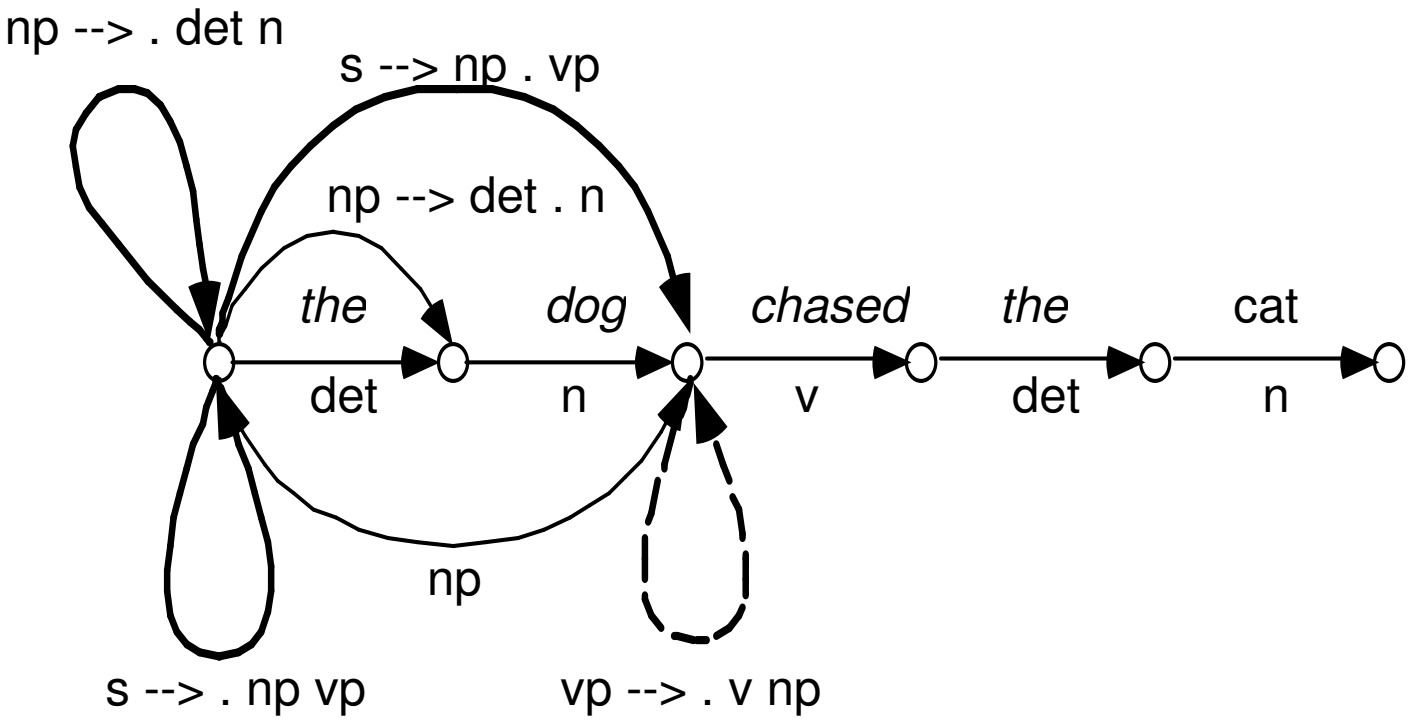


parc
Palo Alto Research Center

Step 3

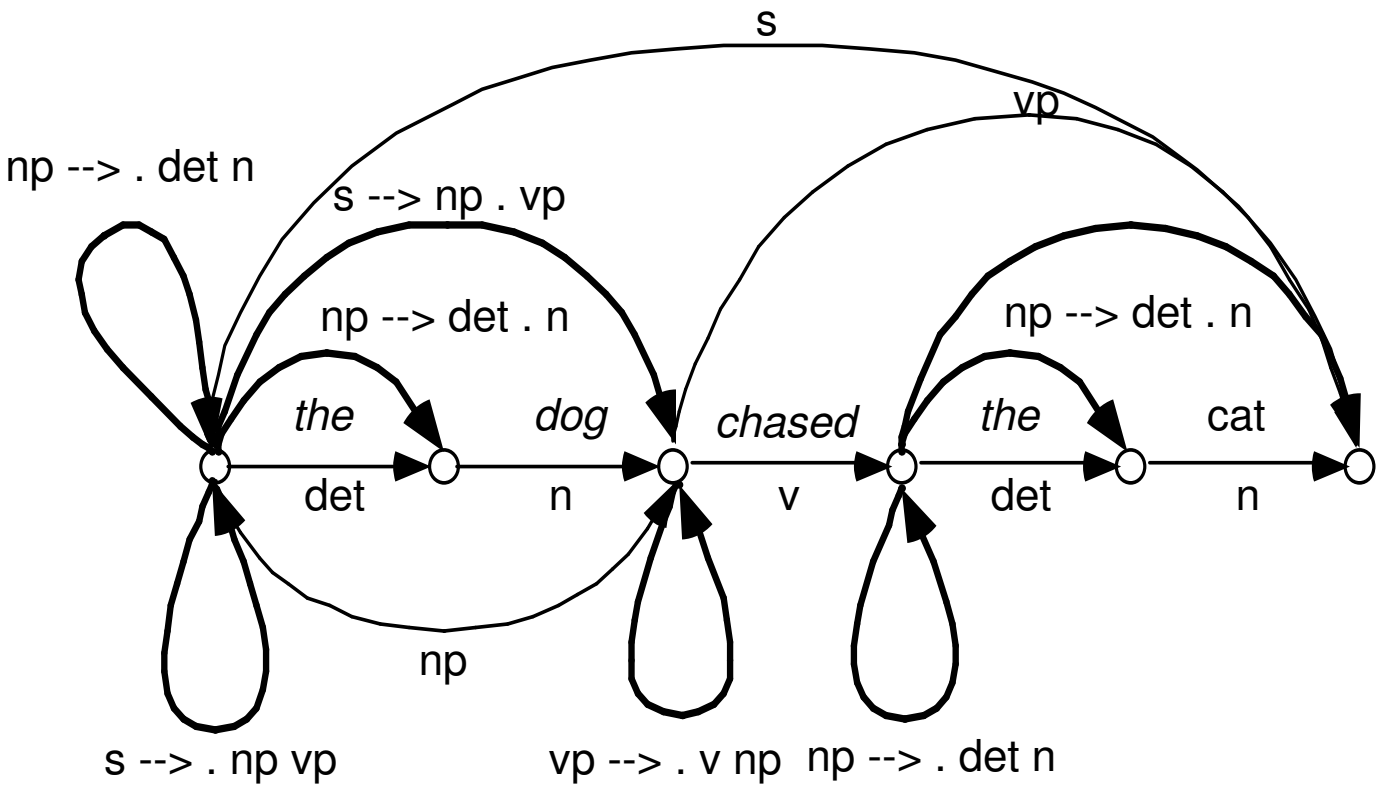


Step 4



parc
Palo Alto Research Center

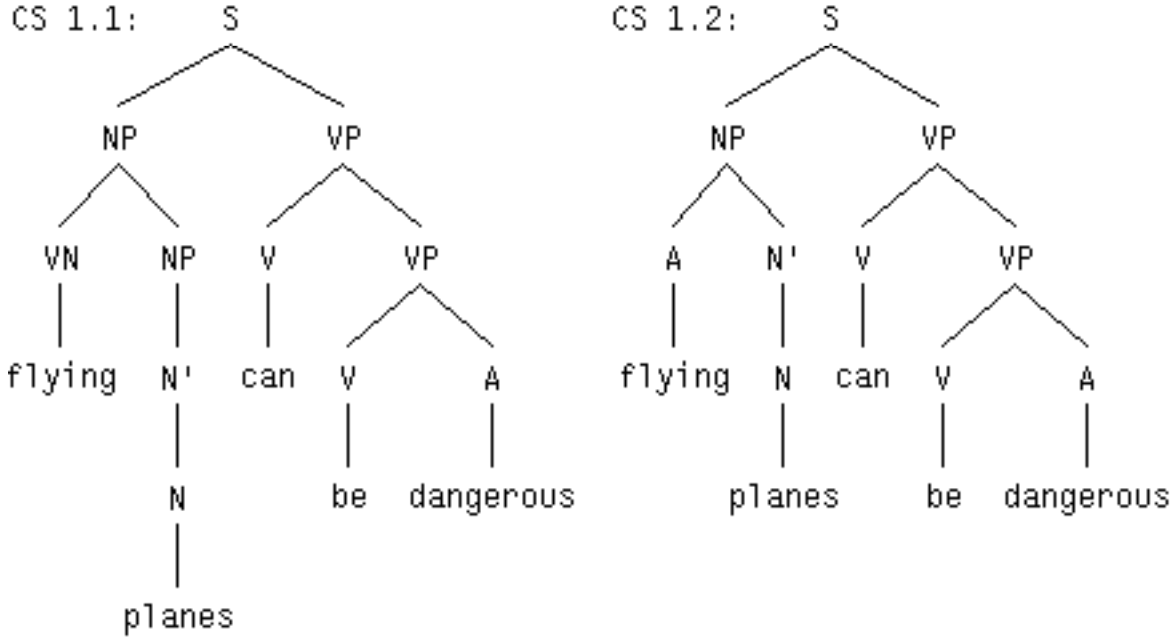
The Final Situation



parc
Palo Alto Research Center

The Context-free parsing paradox

- The number of structures a sentence may have grows exponentially with its length, *but*
- You can characterize them all in polynomial (cubic) time
- HOWEVER It takes exponential time to examine (read out) all the structures



parc
Palo Alto Research Center

UNIFICATION

- Feature structures
  - LFG f(unctional)-structures
- Unifying feature structures
  - Merging f-structure information

Feature Structures

- Sets of feature-value pairs
  - attribute-value matrices
- Values can be
  - atomic

    [PERS 3]

    [NUM sg]
  - feature-structures

    [SUBJ [PERS 3]]

    [TNS-ASP [TENSE present]]

Unifying feature structures

- Merge the information of two feature structures
  - Takes two feature structures as arguments
  - Returns a feature structure if successful:

    if the information is compatible
- Simple example:

  √ [NUM sg] UNI [NUM sg] = [NUM sg]

  * [NUM sg] UNI [NUM pl]

More unification

- Empty value is compatible: a value can unify with the empty value

  √ [NUM sg] UNI [NUM [ ]] = [NUM sg]

- Merger: two different features can unify to form a complex structure

  √ [NUM sg] UNI [PERS 3] = [NUM sg
  PERS 3]

Mathematics and computation

- LFG: simple combination of two simple theories
  - Context-free grammars for c-structures
  - Quantifier free theory of equality for f-structures
- Both theories are easy to compute
  - Cubic CFG Parsing
  - Linear equation solving
- Combination is difficult: Recognition problem is NP Complete
  - Exponential/intractible in the worst case

parc
Palo Alto Research Center

Sources of worst-case complexity

- Exponentially many trees
  - CFG parser quickly produces packed representation of all trees
  - But trees must be enumerated to produce f-structure constraints, determine satisfiability
  - CFG can be exponentially ambiguous
- Boolean combinations of per-tree constraints

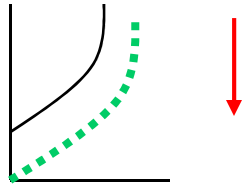
  $(\uparrow \text{SUBJ NUM}) \neq \text{SG} \vee (\uparrow \text{SUBJ PERS}) \neq 3$

Exponentially many exponential problems

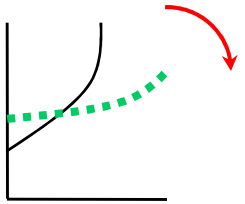
parc
Palo Alto Research Center

Strategies for efficient processing

- Restrict the formalism
  - Still expressive enough for natural language?
- Drop the curve
  - Better CFG parser, unifier
  - Compile to (virtual) machine language
  - Wait



- Change the shape of the curve (on typical cases)
  - Bet on common cases, optimize for those
  - Polynomial representation of exponentially many ambiguities when solutions are combinations of independent subtrees



Parsing summary

- Have a natural language string (sentence)
- Want to produce a structure for it
  - Tree (LFG c-structure)
  - Attribute value matrix (LFG f-structure)
- Some computational issues
  - Efficiency
  - Ambiguity

Generation: Basic idea

- Have:
  - an analysis (LFG f-structure)
  - a grammar (reversibility issues)
- Want the natural language string(s) that correspond to the analysis
- Issues
  - Is there any such string
  - Underspecified input

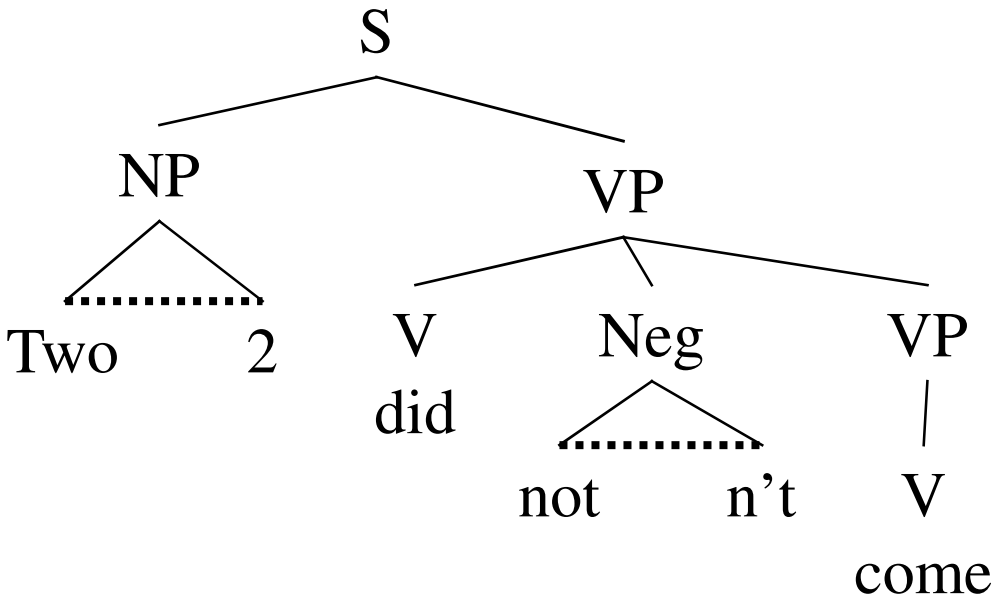
Reversibility

- Parsing grammar and generation grammar need not be the same
- However, if they are:
  - only have to write one grammar and lexicon (modulo minor differences)
  - easier maintenance

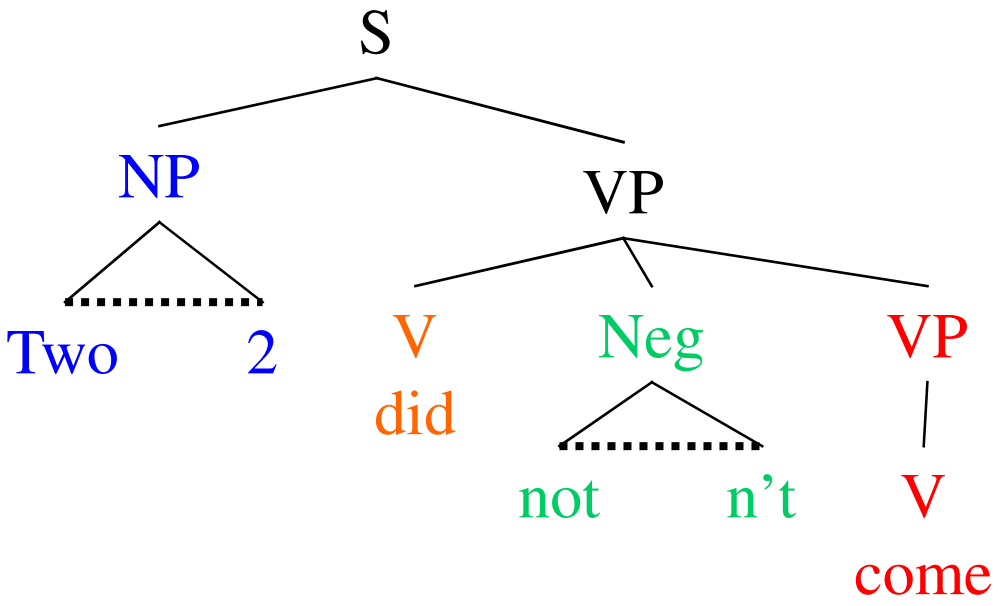
Equivalence classes

- Generation produces different equivalence classes than parsing
- For a given f-structure, there may be several c-structures that rise to it.
  - Grammar dependent
- Examples:
  - { Two | 2 } { didn't | did not } come.
  - { Yesterday we fell. | We fell yesterday. }

C-structure forest



C-structure forest with f-structure correspondence



$$\begin{bmatrix} \text{PRED} & \text{`come<SUBJ>'} \\ \text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{`two'} \\ \text{NUM} & \text{pl} \end{bmatrix} \\ \text{ADJ} & \{[\ \text{PRED `not'}\ ]\} \\ \text{TNS} & \text{past} \end{bmatrix}$$

Generating the c-structure/string

- For each f-structure (e.g. SUBJ, ADJ), the generator produces all the c-structures consistent with it
- This gives a generation chart
  - similar to a parsing chart
  - different equivalence classes
- Completeness
  - everything generated at least once
  - semantic forms (PREDs) generated at most once

    the red, red dog vs. *the red dog*

Generation chart: Kay/Maxwell

- Each edge lists the semantic facts generated beneath it
- Root edge must list all the semantic facts
- Exponential in number of Adjuncts due to optionality in the grammar

XLE generation algorithm

- Two phases
- Quickly construct a guide, a representation from which at least all the proper strings, and some incorrect ones, can be produced.
  - The guide is constructed on the basis of only some of the functional information.
  - The guide gives a finite-set of possibilities.
- Run all (or most) of the parsing machinery on these possibilities to evaluate the functional constraints.

XLE generation issues

- XLE is oriented towards process efficiency.
- Issues:
  - If the generated language is infinite, then XLE must depend on some pumping mechanism to enumerate (in principle) all possible strings on the basis of a finite basis-set.
  - Underspecified input

Underspecified input

- F-structure may be underspecified (features may be missing)
  - machine translation
  - knowledge representation
- Generator must be able to fill in the missing values
  - which attributes and values are addable
  - how to stop the process

Bounded underspecification

- Problem if input is an arbitrary underspecification of some intended f-structure (Context-free results do not hold)
- Fail for arbitrary extensions.
- OK if only a bounded number of features can be added: The set of all possible extensions is then finite (Context-free result does hold).

parc
Palo Alto Research Center

Generation: Where do the analyses come from?

- To generate, need an analysis (f-structure)
- Producing these structures can be challenging
  - for a given sentence
  - for a discourse (connected set of sentences)
- F-structure rewrites
  - translation
  - sentence condensation
- Challenge comes with non-f-structure sources
  - Question answering

parc
Palo Alto Research Center

Conclusions

- Parsing: have a string, produce an analysis
  - LFG: produce c-structure and f-structure
  - Context free backbone for c-structure
  - Unification for f-structure
- Generation: have an f-structure, produce a string
  - Efficient, complete generation
  - Reversibility of the grammar
  - Underspecified input