Language Generation

Some slides adapted from

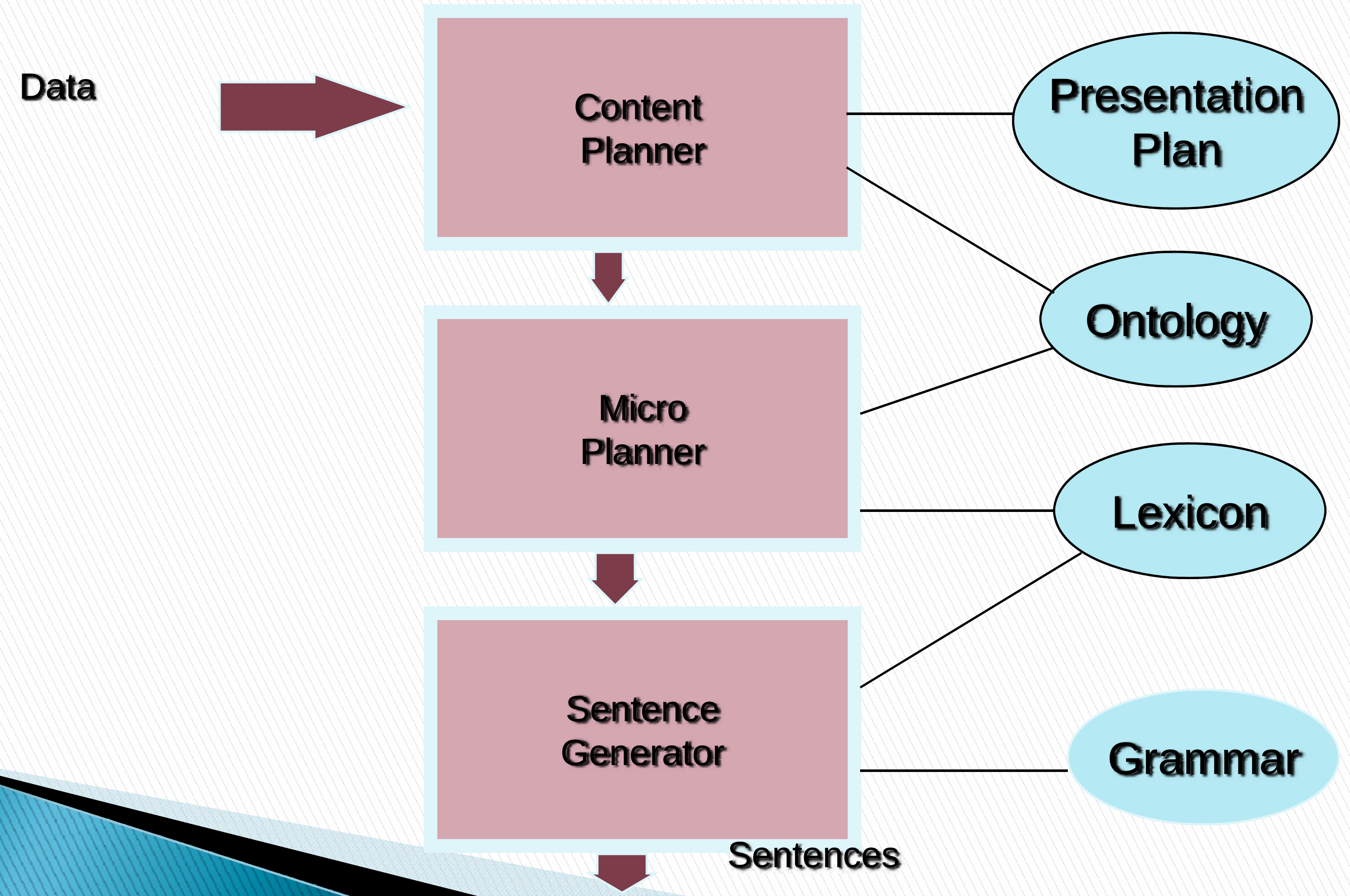
Today

- Linguistic Generation
- Statistical Generation

Two Types of Problems

- Conceptual:
  - What to say
  - How to organize
- Linguistic
  - How to say it
    - Words?
    - Syntactic structure

A Language Generator



Why isn't generation the reverse of parsing?

- Parsing
  - Input = sentence
  - Output = parse tree
- Generation
  - Output = sentence
  - Input = parse tree?

Generation = Decision making under constraints

- Syntactic
    - Agent = The President
    - Pred = pass
    - Patient = tax bailout plain
    - When = yesterday
  - The President passed the tax bailout plan
  - The tax bailout plan was passed by the President
  - The tax bailout plan was passed
  - It was the President who passed the tax bailout plan
  - It was the tax bailout plan the President passed.
- Constraints?

Lexical Choice

- Bought vs sell
  - Kathy bought the book from Joshua.
  - Joshua sold the book to Kathy.
- Erudite vs. wise
  - The erudite old man taught us ancient history.
  - The wise old man taught us ancient history.
- Polarity vs. “plus/minus”
  - Insert the battery and check the polarity.
  - Insert the battery and make sure the plus lines up with the plus.
- Edged out vs. beat
  - The Denver Nuggets edged out the Boston Celtics 102-101
  - The Denver Nuggets beat the Boston Celtics with a narrow margin 102-101.
- Constraints?

Lexical Choice

- Syntax
  - Allow one to select
  - Allow the selection
- Semantics
  - Rebound vs. point in basketball
- Lexical
  - “grab a rebound” vs. “score a point” and not vice versa
- Domain
  - IBM rebounded from a 3 day loss.
  - Magic grabbed 20 rebounds.
- Pragmatics
  - A glass half-full
  - A glass half-empty

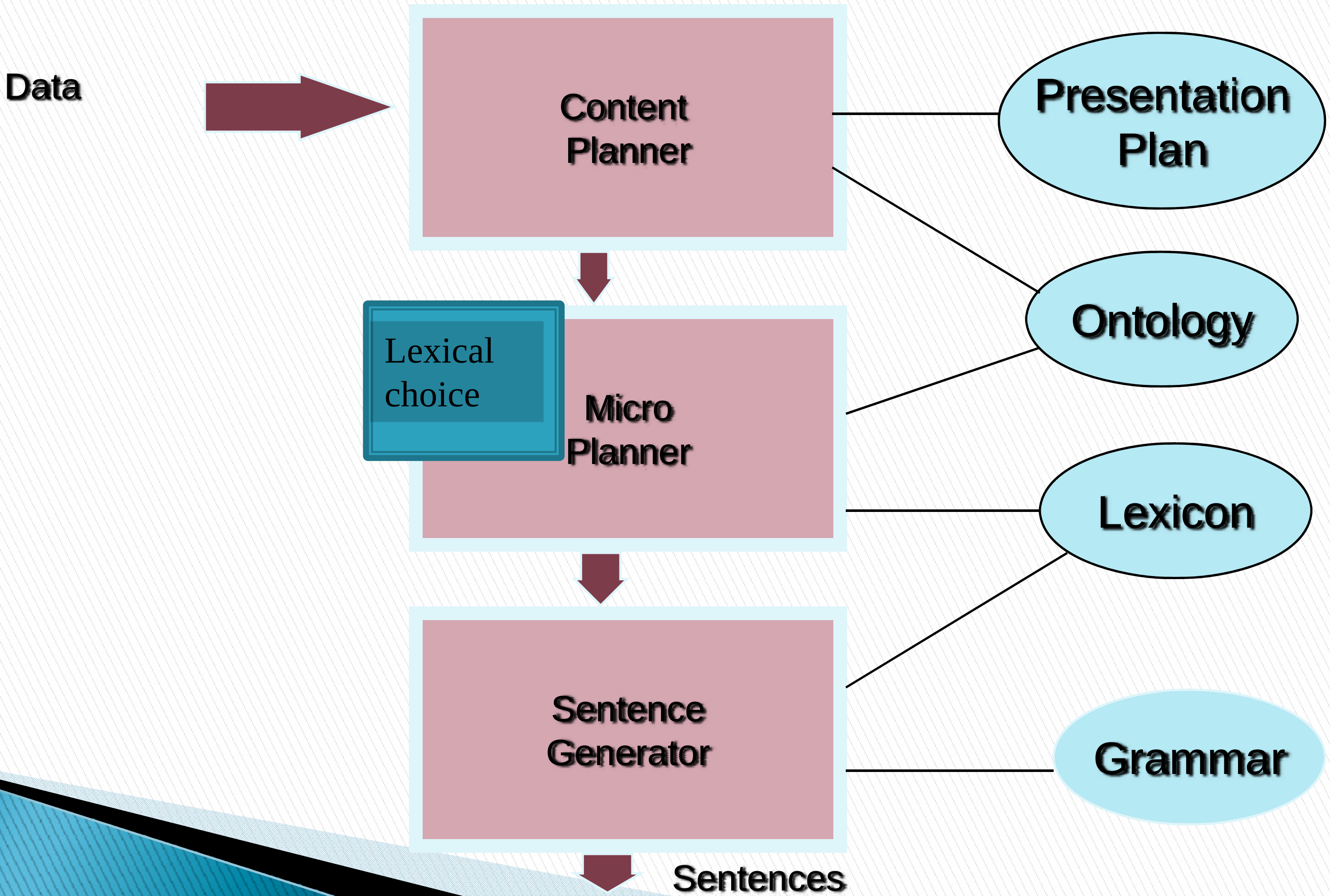
Floating Constraints

- Inter-lexical (e.g., collocations)
- Cross-ranking (content units are not isomorphic with linguistic units)

Constraints on Lexical Choice Float

- Wall Street indexes *opened* **strongly**. (*time* in verb, **manner** as adverb)
- Stock indexes **surged** *at the start of the trading day*. (*time* as PP, **manner** in verb)
- The Denver Nuggets beat the Boston Celtics **with a narrow margin**, 102-101. (game result in verb, **manner** in PP)
- The Denver Nuggets **edged out** the Boston Celtics 102-101. (game result and **manner** in verb)

A Language Generator



Functional Unification Grammar

- Function plays as important a role as syntax
  - Pragmatics, semantics are represented equally with syntactic features, constitutents
- Unification is used to enrich the input with constraints from the grammar
  - Input is recursively unified with grammar
  - Top-down process

Functional Unification

- Functional Descriptions (FDs) as a feature structure
  - Data structure that is partial and structured

- Input and grammar are both specified as functional descriptions

An example grammar

```
((alt GSIMPLE (
   ;; a grammar always has the same form: an alternative
   ;; with one branch for each constituent category.

   ;; First branch of the alternative
   ;; Describe the category clause.
   ((cat clause)
    (agent    ((cat np)))
    (patient ((cat np)))
    (pred   ((cat verb-group)
                (number {agent number})))
    (cset (pred agent patient))
    (pattern  (agent pred patient))

   ;; Second branch: NP
   ((cat np)
    (head ((cat noun) (lex {^ ^ lex})))
    (number  ((alt np-number (singular plural))))
    (alt (  ;; Proper names don't need an article
            ((proper yes)
             (pattern (head)))

            ;; Common names do
            ((proper no)
             (pattern (det head))
             (det ((cat article) (lex "the")))))))

   ;; Third branch: Verb
   ((cat verb-group)
    (pattern (v))
    (aux none)
    (v ((cat verb) (lex {^ ^ lex}))))
))
```

A simple input

- Input to generate: The system advises John.

```
I1 =        ((cat clause)
             (tense present)
             (pred    ((lex "advise")))
             (agent    ((lex "system") (proper no)))
             (patient ((lex "John"))))
```

Unification Output

```
((cat clause)
        (tense present)
        (pred   ((lex "advise")
                      (cat verb-group)
                      (number {agent number})
                      (PATTERN (V))
                      (AUX NONE)
                      (V ((CAT VERB) (LEX {^ ^ LEX})))))
        (agent    ((lex "system") (proper no)
                      (cat np)
                      (HEAD ((CAT NOUN) (LEX {^ ^ LEX})))
                      (NUMBER SINGULAR)
                      (PATTERN (DET HEAD))
                      (DET ((CAT ARTICLE) (LEX "the")))))
        (patient ((lex "John")
                      (cat np)
                      (HEAD ((CAT NOUN) (LEX {^ ^ LEX})))
                      (NUMBER SINGULAR)
                      (PROPER YES)
                      (CSET (HEAD))
                      (PATTERN (HEAD))))
        (cset (pred agent patient))
        (pattern (agent pred patient)))
```

Linearization

- Identify the pattern feature in the top level: for I1, it is (pattern (agent action affected)).
- If a pattern is found:
  - For each constituent of the pattern, recursively linearize the constituent. (That means linearize agent, pred and patient).
  - The linearization of the FD is the concatenation of the linearizations of the constituents in the order prescribed by the pattern feature.
- If no pattern is found:
  - Find the lex feature of the FD, and depending on the category of the constituent, the morphological features needed. For example, if the FD is of (cat verb), the features needed are: person, number, tense.
  - Send the lexical item and the appropriate morphological features to the morphology module. The linearization of the fd is the resulting string. For example, for (lex="advise") when the features are the default values (as they are in I1), the result is *advises*. When the FD does not contain a morphological feature, the morphology module provides reasonable defaults.

Encoding Function

```
((cat clause)
    (agent    ((cat np)))
    (patient ((cat np)))
    (alt (
      ((focus {agent})
        (voice active)
        (pred   ((cat verb-group)
                       (number {agent number})
        (cset (action agent affected))
        (pattern  (agent action affected)))
      ((focus {patient})
        (voice passive)
        (verbs  ((cat verb-group)
                     (aux  ((lex "be")
                                (number {patient number}))
                      (pastp  ({pred}
                                  (tense pastp)))
                      (pattern (aux pastp))))
         (by-agent  {agent})
         (pattern (patient verbs by-agent))))
```

Realization with Statistics

- Problem: What does the input to realization look like?

- Wouldn’t it be easier to automatically learn output?
  - What does it take to scale up linguistic grammars?

Implicit Linguistic Knowledge – Grammar

- Subject-verb agreement
  I am vs. *I are* vs. *I is*

- Corpus counts (Langkilde-Geary, 2002)

I am	2797
I are	47
I is	14

Implicit Linguistic Knowledge – Grammar

- Choice of determininer
  a trust vs. *an trust* vs. *the trust*

- Corpus counts (Langkilde-Geary, 2002)

A trust	394
An trust	0
The trust	1356
A trusts	2
An trusts	0
The trusts	115

Realization with statistics: Key Techniques

- Over-generate and prune
- Automatically acquire grammar from a corpus (if a phrase structure grammar is needed)
- Exploit general-purpose tools and resources when possible & appropriate
  - Tokenizers
  - Part-of-speech taggers
  - Parsers, Penn Treebank
  - WordNet, VerbNet

Overgenerate and prune

- General strategy:
  - Generate multiple candidate sentences with some permissive strategy
- Some sentences may be very ungrammatical!
- Very many sentences (millions) may be generated
  - Assign scores to the candidate sentences using a corpus-based language model
  - Output the highest-ranking sentence(s)

Generate multiple candidates with permissive strategy

- I is not able to betray their trust .
- I cannot betray trust of them .
- I cannot betray the trust of them .
- I am not able to betray their trust .
- I will not be able to betray the trust of them .
- I will not be able to betray their trust .
- I cannot betray their trust .
- I cannot betray trusts of them .
- I are not able to betray their trust .
- I cannot betray a trust of them .s

Assign scores using language model

1. I cannot betray their trust .
2. I will not be able to betray their trust .
3. I am not able to betray their trust .
4. I are not able to betray their trust .
5. I is not able to betray their trust .
6. I cannot betray the trust of them .
7. I cannot betray trust of them .
8. I cannot betray a trust of them .
9. I cannot betray trusts of them .
10. I will not be able to betray the trust

Output highest ranking sentence

1. I cannot betray their trust .
2. I will not be able to betray their trust .
3. I am not able to betray their trust .
4. I are not able to betray their trust .
5. I is not able to betray their trust .
6. I cannot betray the trust of them .
7. I cannot betray trust of them .
8. I cannot betray a trust of them .
9. I cannot betray trusts of them .
10. I will not be able to betray the trust

NITROGEN

- Early, influential statistical realization algorithm
  - Langkilde & Knight (1998)
  - Hatzivassiloglou & Knight (1995)
- Uses an overgenerate and prune strategy

NITROGEN input format

- Input: Abstract Meaning Representation (AMR)
  - Based on Penman Sentence Plan Language (See Kasper 1989, Langkilde & Knight 1998)
- Example AMR: (m1 / |dog<canid|)
  - m1 is an instance of |dog<canid| -- derived from WordNet
  - Might be realized " the dog" , " the dogs" , " a dog" , " dog" ,...
- Another example AMR:
  - (m3 / |eat, take in|

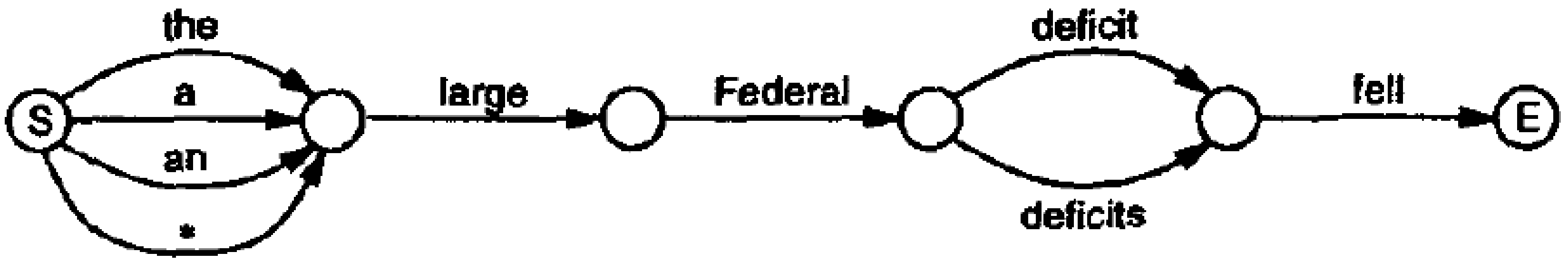
        :agent (m4 / |dog<canid| :quant plural)
         :patient (m5 / |os,bone|))
  - Might be realized as " the dogs ate the bone" , "Dogs will eat a bone" , " The dogs eat bones" , "Dogs eat bone" ,...

NITROGEN Lattices

- In practice, overgeneration can produce millions of sentences for a single input
  - So need very compact representations or prune aggressively
- Nitrogen uses a lattice representation
  - Lattice is an acyclic graph where each arc is labeled with a word.
  - A complete path from the left-most node to rightmost node through the lattice represents a possible expression/sentence.

NITROGEN Lattices: idea

- Suppose realizer, looking at an AMR input, is uncertain about definiteness and number. Can generate a lattice fragment like this:



- Generates:

The large Federal deficit fell.

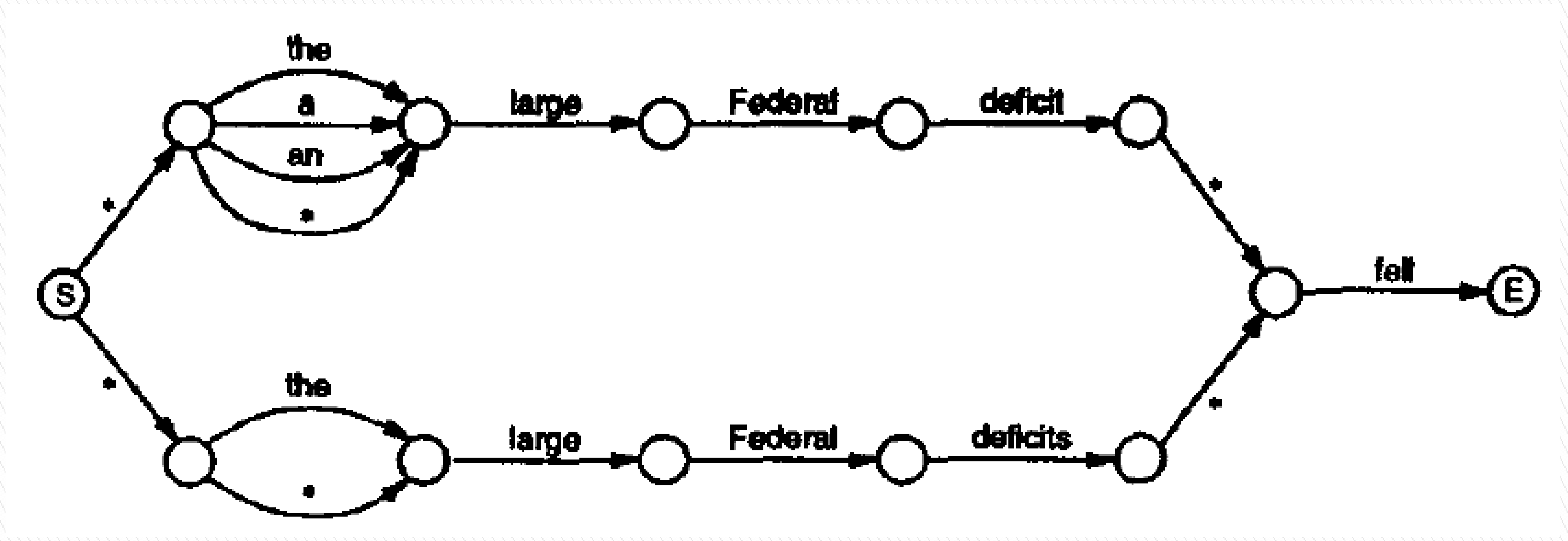
A large Federal deficit fell.

An large Federal deficit fell large.

Federal deficit fell.

A large Federal deficits fell.

Perhaps a better lattice

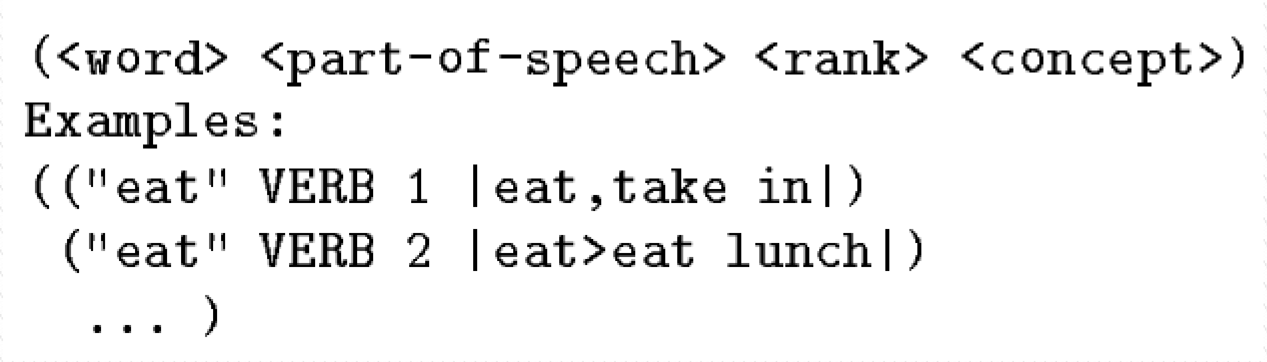


NITROGEN lattices: generation

- Set of hand-built rules link AMR patterns to lattice fragments
- Each AMR pattern is deliberately mapped to many different realizations (overgeneration)
- A lexicon describes alternative words that can express AMR concepts.

NITROGEN Lexicon

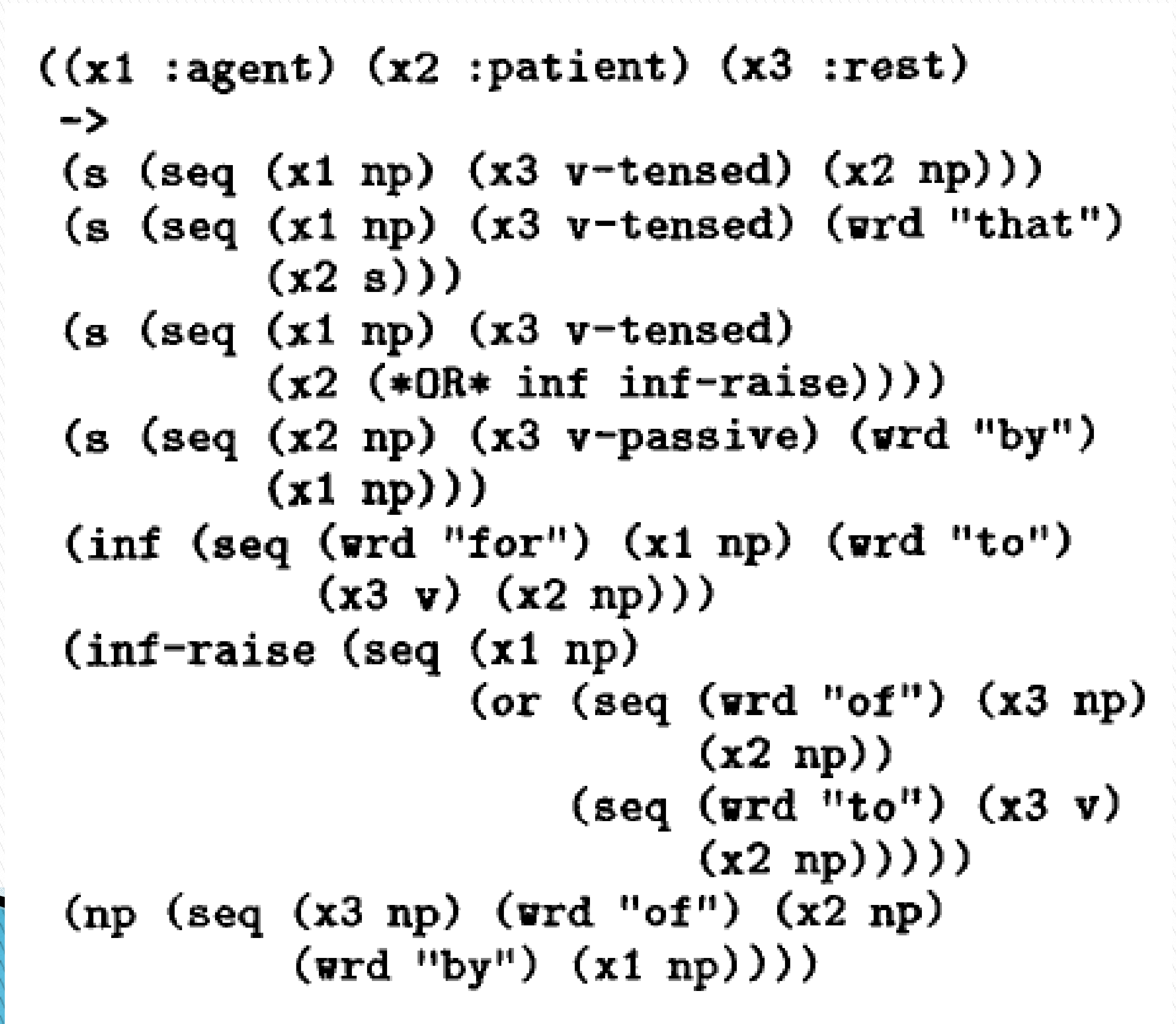
- A lexicon of 110,000 entries connects concepts to alternative English words. Format:



```
(<word> <part-of-speech> <rank> <concept>)
Examples:
(("eat" VERB 1 |eat,take in|)
 ("eat" VERB 2 |eat>eat lunch|)
  ... )
```

- Important note: no features like transitivity, subcategorization, gradability (for adjectives), or countability (for nouns).
  - This is a substantial advantage for development.

NITROGEN: Example rule

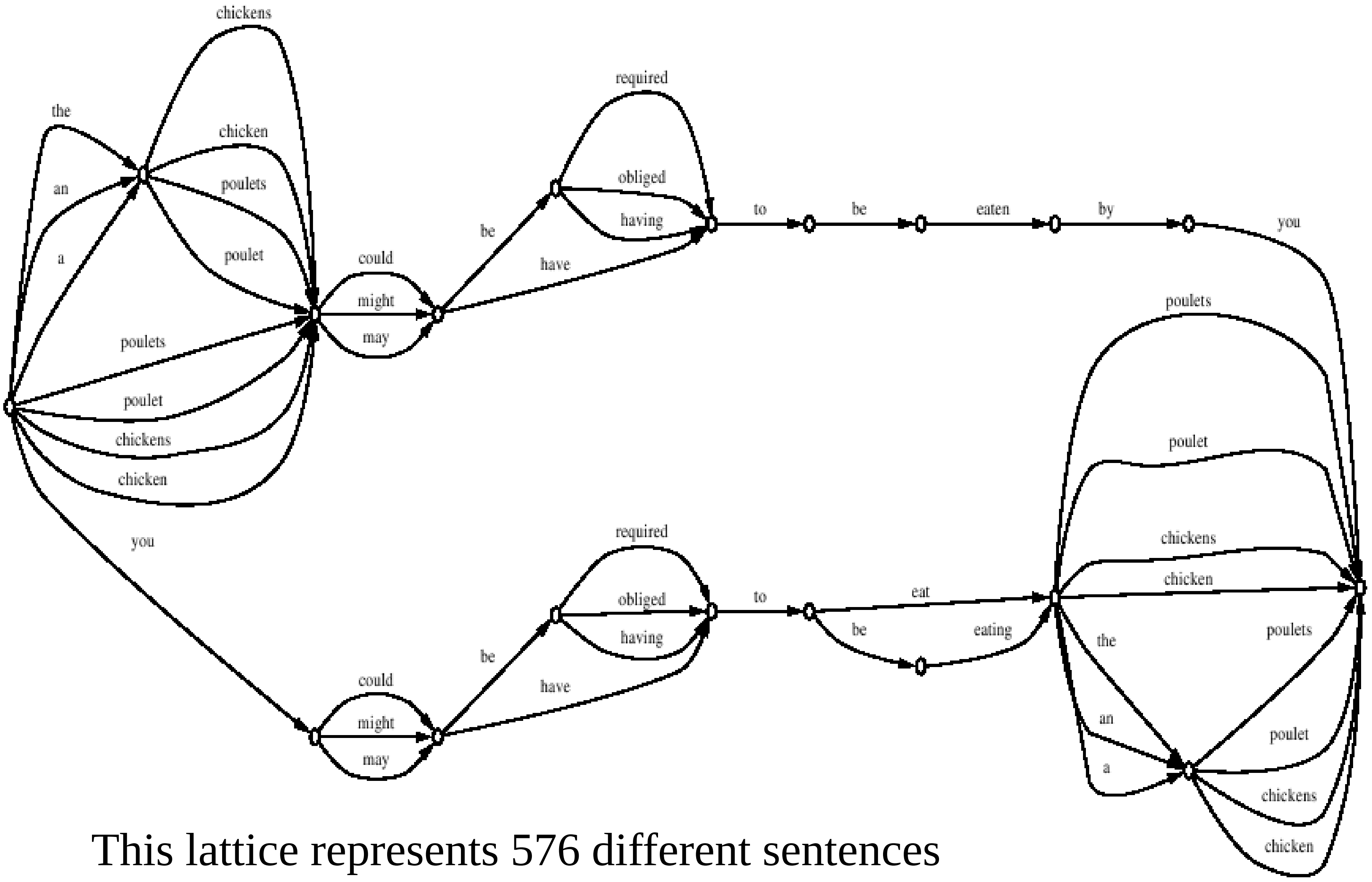


```
((x1 :agent) (x2 :patient) (x3 :rest)
 ->
 (s (seq (x1 np) (x3 v-tensed) (x2 np)))
 (s (seq (x1 np) (x3 v-tensed) (wrd "that")
         (x2 s)))
 (s (seq (x1 np) (x3 v-tensed)
         (x2 (*OR* inf inf-raise))))
 (s (seq (x2 np) (x3 v-passive) (wrd "by")
         (x1 np)))
 (inf (seq (wrd "for") (x1 np) (wrd "to")
           (x3 v) (x2 np)))
 (inf-raise (seq (x1 np)
                 (or (seq (wrd "of") (x3 np)
                              (x2 np))
                     (seq (wrd "to") (x3 v)
                              (x2 np)))))
 (np (seq (x3 np) (wrd "of") (x2 np)
          (wrd "by") (x1 np))))
```

Generation algorithm

- Algorithm sketch: Traverse input AMR bottomup, building lattices for the leaves (innermost nested levels of the input) first, to be combined at outer levels according to relations between the leaves
  - (see Langkilde & Knight 1998 for details)

- Result is a large lattice like...



This lattice represents 576 different sentences

Extracting high-probability sentences from a lattice

- Nitrogen uses a bigram/trigram language model built from 46 million words of Wall Street Journal text from 1987 and 1988.
- As visit each state *s, maintain list of most* probable sequences of words from start to *s:*
  - Extend all word sequences to predecessors of *s,*recompute scores, prune down to 1000 most probable sequences per state.
- At end state, emit most probable sequence.

Questions

- Do the two approaches handle the same phenomena?
- Could they be integrated?

References

- 1989 Kasper, A flexible interface for linking applications to Penman's sentence generator
- 1995 Hatzivassiloglou & Knight, Unification Based Glossing
- 1995 Knight & Hatzivassiloglou, Two Level Many Paths Generation
- 1998 Langkilde & Knight, Generation that Exploits Corpus Based Statistical Knowledge
- 2000 Langkilde, Forest Based Statistical Sentence Generation
- 2002 Langkilde-Geary, An Empirical Verification of Coverage and Correctness for a General Purpose Sentence Generator
- 1998 Langkilde & Knight, The practical value of n grams in generation
- 2002 Langkilde & Geary, A foundation for general purpose natural language generation sentence realization using probabilistic models of language
- 2002 Oh & Rudnicky, Stochastic natural language generation for spoken dialog systems
- 2000 Ratnaparkhi, Trainable methods for surface natural language generation