Natural language processing using constraint-based grammars

Ann Copestake

University of Cambridge Computer Laboratory

Center for the Study of Language and Information, Stanford

aac@cl.cam.ac.uk

Overview of course

- NLP applications. State-of-the-art, deep vs shallow processing, deep processing modules. What are constraint-based grammars, why use constraint-based grammars?
- Implementing and using constraint-based grammars. Formalism (inheritance, type constraints), semantic representation and generation, grammar engineering.
- Test suites and efficiency issues. Some research issues: stochastic HPSG, multiword expressions, combining deep and shallow processing.

Although these are mostly general issues, specific examples and demos will mostly be of LinGO technology.

Overview of lecture 1

- NLP applications
- Deep vs. shallow processing
- Architecture of deep processing systems
- Constraint-based grammar
- Constraint-based grammar formalisms in NLP applications: why and how?
- Demo of LKB and ERG

Some NLP applications

- spelling and grammar checking
- screen readers and OCR
- augmentative and alternative communication
- machine aided translation
- lexicographers’ tools
- information retrieval
- document classification (filtering, routing)
- document clustering
- information extraction
- question answering
- summarization

- text segmentation
- exam marking
- report generation (mono- and multi-lingual)
- machine translation
- natural language interfaces to databases
- email understanding
- dialogue systems

Example 1: Email routing

Email sent to a single address (e.g. a company) is sorted into categories depending on subject, so it can be routed to the right department. For instance:

New orders

Questions about orders

General queries

Junk email

Most such systems depend on a mixture of types of evidence: e.g., words in the email body, address of sender etc, number of exclamation marks (for detecting junk email). Systems can be trained based on manually classified data.

Example 2: automatic response to email

Within-domain questions

1. Has my order number 4291 been shipped yet?
2. Is FD5 compatible with a Vaio 505G?
3. What is the speed of the Vaio 505G?
4. How long will 4291 take?
5. How long is FD5?

Out of domain

1. My order did not arrive on time. You will be hearing from my lawyers.
2. What is the speed of an African swallow?

How automatic question response works

1. Analyze the incoming question to produce a query in some formal meaning representation
2. If no possible query can be constructed, pass the question to a human
3. Otherwise, run the query against the relevant database
4. Generate a response

Database querying

ORDER

Order number	Date ordered	Date shipped
4290	2/2/02	2/2/02
4291	2/2/02	2/2/02
4292	2/2/02	-

1. USER QUESTION: Have you shipped 4291?
2. DB QUERY: order(number=4291,date_shipped=?)
3. RESPONSE TO USER: Order number 4291 was shipped on 2/2/02

Shallow and deep processing

Most NLP applications fall into one of two categories:

1. Narrow coverage deep processing (e.g., email response): target is a fully described data or knowledge base.
2. Broad-coverage shallow processing (e.g., email routing): extract partial information from (relatively) unstructured text.

Some applications are intermediate: good MT requires limited domains, but MT on unrestricted text can involve relatively deep processing (semantic transfer). Recently, systems for question answering on unrestricted text have been developed: some of these use relatively deep processing.

Methodology

The deep/shallow distinction is partially aligned with methodology:

1. Knowledge-intensive NLP methods (i.e., methods that require extensive ‘linguistic’ hand-coding) are generally used for deep processing (though also sometimes for shallow processing, like POS tagging).
2. Machine-learning techniques are generally used for shallow processing (though some attempts to use them for deep processing).
3. Statistical NLP is always associated with machine-learning, and generally with shallow processing, but most full systems combine statistical and symbolic techniques.

Most deep processing assumes a limited domain, but this isn’t true of question answering and machine translation.

Some history

Natural language interfaces were the ‘classic’ NLP problem in the 70s and early 80s. LUNAR was a natural language interface to a database (Woods, 1978 — but note most of the work was done several years earlier): it was capable of translating elaborate natural language expressions into database queries.

SHRDLU (Winograd, 1973) was a system capable of participating in a dialogue about a microworld (the blocks world) and manipulating this world according to commands issued in English by the user.

LUNAR and SHRDLU both exploited the limitations of the domain to make the natural language understanding problem tractable. For instance, disambiguation, compound noun analysis, quantifier scope, pronoun reference.

Domain knowledge for disambiguation

Schematically, in the blocks world:

1. Context: blue(b1), block(b1), on(b1,b2), red(b2), block(b2), pyramid(p3), green(p3), on(p3,ground) etc
2. Input: Put the green pyramid on the blue block on the red blocks
3. Parser:
   (a) (Put (the (green pyramid on the blue block)) (on the red blocks))
   (b) (Put (the green pyramid) (on the (blue block (on the red blocks))))
4. Context resolves to: (Put (the green pyramid) (on the (blue block (on the red blocks))))

But doesn't scale up well: AI-complete for arbitrary domains.

Developments since 1970s

No really good way of building large-scale detailed knowledge bases has been found, but there have advances in deep NLP since LUNAR:

1. powerful, declarative grammar formalisms
2. more motivated approaches to semantics
3. better methodology for evaluation
4. modularity reduces difficulty of porting between domains
5. large scale, domain-independent grammars have been built
6. disambiguation etc is yielding (slowly) to corpus-based methods
7. systems are much easier to build

Commercial systems remain rare.

Domain-independent linguistic processing

Most linguistically-motivated deep processing work assumes a level of representation constructed by a (somewhat) domain-independent grammar that can be mapped into the domain-dependent application.

For instance:

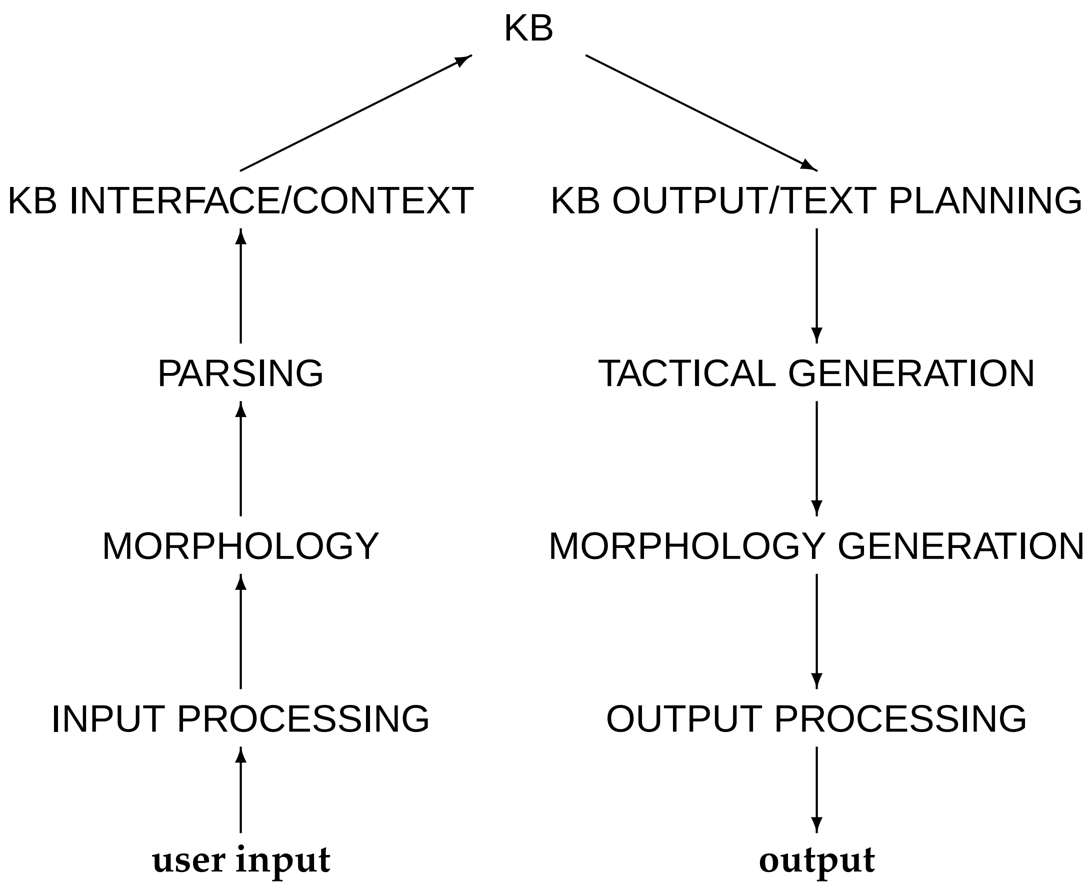
1. USER QUESTION: Have you shipped 4291?
2. SEMANTIC REP: ynq(2pers(y) and def(x, id(x,4291), ship(e,y,x) and past(e)))
3. DB QUERY: order(number=4291,date_shipped=?)

So don't have to completely rewrite the grammar for each new application. (Currently deployed spoken dialogue systems don't do this, however.)

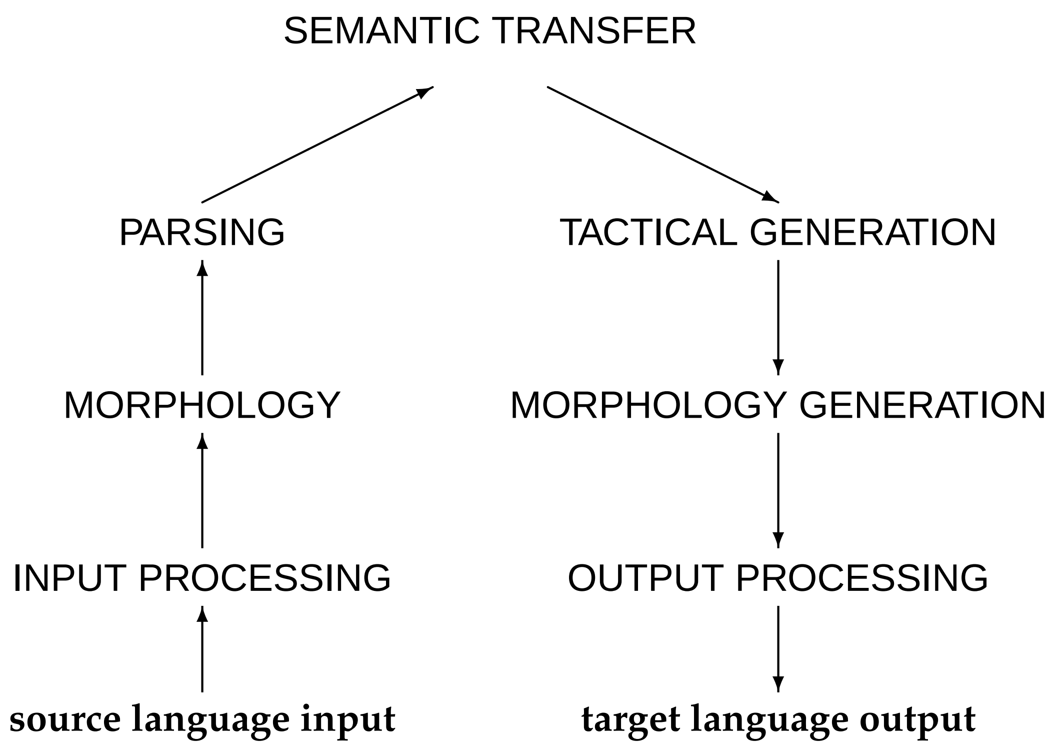
Generic NLP application architecture

- input preprocessing: speech recogniser or text preprocessor (non-trivial in languages like Chinese) or gesture recogniser.
- morphological analysis
- parsing: this includes syntax and compositional semantics
- disambiguation
- context module
- text planning: the part of language generation that's concerned with deciding what meaning to convey
- tactical generation: converts meaning representations to strings.
- morphological generation
- output processing: text-to-speech, text formatter, etc.

Natural language interface to a knowledge base



MT using semantic transfer



Candidates for the parser/generator

1. Finite-state or simple context-free grammars. Used for domain-specific grammars.
2. Augmented transition networks. Used in the 1970s, most significantly in LUNAR.
3. Induced probabilistic grammars
4. Constraint-based grammars
    - Linguistic framework: FUG, GPSG, LFG, HPSG, categorial grammar (various), TAG, dependency grammar, construction grammar, ...
    - Formalisms: DCGs (Prolog), (typed) feature structures, TAG . . .
    - Systems: PATR, ANLT parser, XTAG parser, XLE, CLE, LKB, ALE . . .
    - Grammars: ANLT grammar, LinGO ERG, XTAG grammar, PARGRAM, CLE grammars.

What is a constraint-based grammar (CBG)?

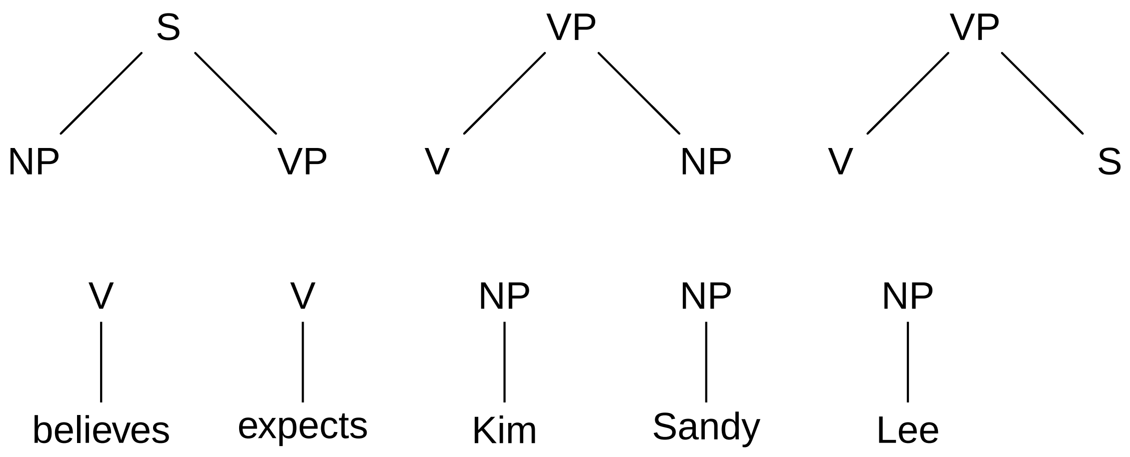
A grammar expressed in a formalism which specifies a natural language using a set of independently specifiable constraints, without imposing any conditions on processing or processing order.

For example, consider a conventional CFG:

```
S -> NP VP
VP -> V S
VP -> V NP
V -> believes
V -> expects
NP -> Kim
NP -> Sandy
NP -> Lee
```

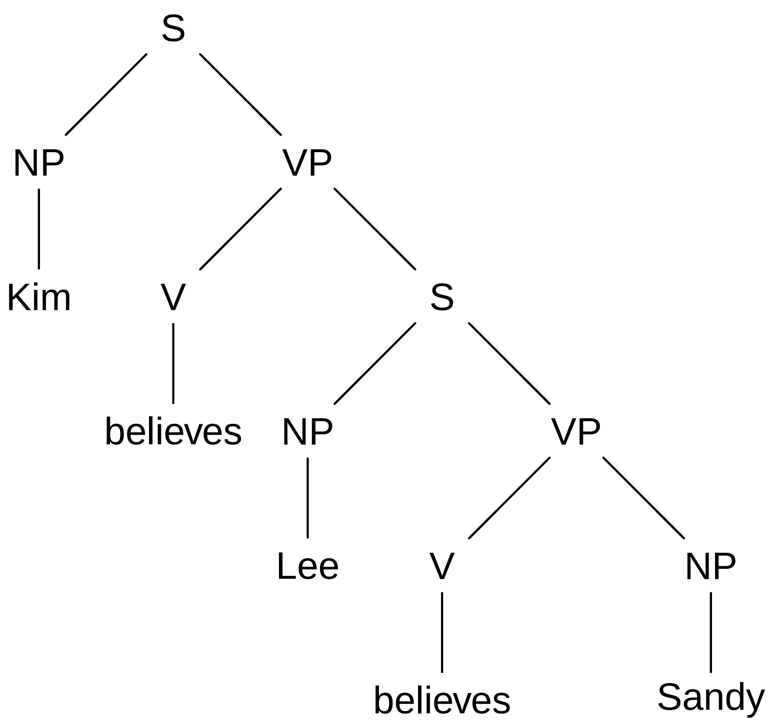
Rule notation suggest a procedural description (production rules).

CFG rules as tree fragments



A tree is licensed by the grammar if it can be put together from the tree fragments in the grammar.

Example of valid tree



Informal definition of CFG as constraints on trees

licensed tree any tree with subtrees that are all made up of trees in the grammar

licensed string any string which can be read off a fully terminated licensed tree

licensed sentence any licensed string with a corresponding tree that is headed by the start symbol (S in this case)

Simple CFGs are not usually taken as examples of constraint-based grammars, however. The unaugmented CFG notation is not powerful enough to usefully represent natural language, so we need richer alternatives (at least notationally richer).

Feature structures

Grammar ‘rules’ in feature structure grammars can also be seen as describing fragments that can be put together:

$$\begin{bmatrix} \textsc{categ} & \textbf{S} \\ \textsc{dtr1} & \begin{bmatrix} \textsc{categ} & \textbf{NP} \end{bmatrix} \\ \textsc{dtr2} & \begin{bmatrix} \textsc{categ} & \textbf{VP} \end{bmatrix} \end{bmatrix} \begin{bmatrix} \textsc{categ} & \textbf{VP} \\ \textsc{dtr1} & \begin{bmatrix} \textsc{categ} & \textbf{V} \end{bmatrix} \\ \textsc{dtr2} & \begin{bmatrix} \textsc{categ} & \textbf{NP} \end{bmatrix} \end{bmatrix} \begin{bmatrix} \textsc{categ} & \textbf{VP} \\ \textsc{dtr1} & \begin{bmatrix} \textsc{categ} & \textbf{V} \end{bmatrix} \\ \textsc{dtr2} & \begin{bmatrix} \textsc{categ} & \textbf{S} \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} \textsc{orth} & \textbf{expects} \\ \textsc{categ} & \textbf{V} \end{bmatrix} \begin{bmatrix} \textsc{orth} & \textbf{believes} \\ \textsc{categ} & \textbf{V} \end{bmatrix} \begin{bmatrix} \textsc{orth} & \textbf{Kim} \\ \textsc{categ} & \textbf{NP} \end{bmatrix} \begin{bmatrix} \textsc{orth} & \textbf{Lee} \\ \textsc{categ} & \textbf{NP} \end{bmatrix} \begin{bmatrix} \textsc{orth} & \textbf{Sandy} \\ \textsc{categ} & \textbf{NP} \end{bmatrix}$$

For example:

$$\begin{bmatrix} \textsc{categ} & \textbf{S} \\ \textsc{dtr1} & \begin{bmatrix} \textsc{orth} & \textbf{Lee} \\ \textsc{categ} & \textbf{NP} \end{bmatrix} \\ \textsc{dtr2} & \begin{bmatrix} \textsc{categ} & \textbf{VP} \\ \textsc{dtr1} & \begin{bmatrix} \textsc{orth} & \textbf{expects} \\ \textsc{categ} & \textbf{V} \end{bmatrix} \\ \textsc{dtr2} & \begin{bmatrix} \textsc{orth} & \textbf{Sandy} \\ \textsc{categ} & \textbf{NP} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Example grammar with agreement

$$\left[\begin{array}{ll} \text{CATEG} & \mathbf{S} \\ \text{DTR1} & \left[\begin{array}{ll} \text{CATEG} & \mathbf{NP} \\ \text{AGR} & \boxed{1} \end{array}\right] \\ \text{DTR2} & \left[\begin{array}{ll} \text{CATEG} & \mathbf{VP} \\ \text{AGR} & \boxed{1} \end{array}\right] \end{array}\right] \quad \left[\begin{array}{ll} \text{CATEG} & \mathbf{VP} \\ \text{AGR} & \boxed{1} \\ \text{DTR1} & \left[\begin{array}{ll} \text{CATEG} & \mathbf{V} \\ \text{AGR} & \boxed{1} \end{array}\right] \\ \text{DTR2} & \left[\begin{array}{ll} \text{CATEG} & \mathbf{NP} \end{array}\right] \end{array}\right] \quad \left[\begin{array}{ll} \text{ORTH} & \mathbf{expects} \\ \text{CATEG} & \mathbf{V} \\ \text{AGR} & \mathbf{3sg} \end{array}\right] \quad \left[\begin{array}{ll} \text{ORTH} & \mathbf{expect} \\ \text{CATEG} & \mathbf{V} \\ \text{AGR} & \mathbf{not3sg} \end{array}\right]$$

$$\left[\begin{array}{ll} \text{ORTH} & \mathbf{Kim} \\ \text{CATEG} & \mathbf{NP} \\ \text{AGR} & \mathbf{3sg} \end{array}\right] \quad \left[\begin{array}{ll} \text{ORTH} & \mathbf{Lee} \\ \text{CATEG} & \mathbf{NP} \\ \text{AGR} & \mathbf{3sg} \end{array}\right] \quad \left[\begin{array}{ll} \text{ORTH} & \mathbf{they} \\ \text{CATEG} & \mathbf{NP} \\ \text{AGR} & \mathbf{not3sg} \end{array}\right] \quad \left[\begin{array}{ll} \text{ORTH} & \mathbf{fish} \\ \text{CATEG} & \mathbf{NP} \\ \text{AGR} & \mathbf{[\,]} \end{array}\right]$$

For example:

$$\left[\begin{array}{ll} \text{CATEG} & \mathbf{S} \\ \text{DTR1} & \left[\begin{array}{ll} \text{ORTH} & \mathbf{they} \\ \text{CATEG} & \mathbf{NP} \\ \text{AGR} & \boxed{1}\ \mathbf{not3sg} \end{array}\right] \\ \text{DTR2} & \left[\begin{array}{ll} \text{CATEG} & \mathbf{VP} \\ \text{AGR} & \boxed{1} \\ \text{DTR1} & \left[\begin{array}{ll} \text{ORTH} & \mathbf{expect} \\ \text{CATEG} & \mathbf{V} \\ \text{AGR} & \boxed{1} \end{array}\right] \\ \text{DTR2} & \left[\begin{array}{ll} \text{ORTH} & \mathbf{Sandy} \\ \text{CATEG} & \mathbf{NP} \\ \text{AGR} & \mathbf{3sg} \end{array}\right] \end{array}\right] \end{array}\right]$$

Informal definition using constraints expressed as feature structures

licensed structure any structure with substructures that are all made up of structures in the grammar

licensed string any string which can be read off a fully-terminated licensed structure

licensed sentence any licensed string with a corresponding structure that has a start structure root.

Here the start structure was simply: $\left[\text{CATEG } \mathbf{s}\right]$

Non-constraint-based approaches to deep parsing

- Augmented transition networks
- PNLP (Jensen and Heidorn)
- LSP (Sager)
- Dynamic syntax (left-to-right processing built-in)
- Optimality theory (at least as usually formulated)
- Charniak-style probabilistic CFGs: automatically learned CFGs with many thousands of rules, some with very low probability. Declarative, but doesn't constrain the language much. (Really intermediate, rather than deep.)

Why constraint-based grammar for NLP?

Explicit formalization

Declarativity

Bidirectionality generate as well as parse (in principle, though not for all large-scale grammars and systems)

Linguistic motivation

Linguistic coverage

Semantic representation

Problems with CBG approaches for NLP

Toy systems

Too many variants

Expense to construct

Training grammar developers

Efficiency

Coverage on real corpora

Ambiguity

Development of good CBG technology is not practical for a single academic site.

LinGO

- Informal collaboration of researchers
- CSLI, Saarbrücken, Tokyo, Sussex, Edinburgh, NTT, Cambridge, NTNU, CELI and others are actively collaborating
- LinGO English Resource Grammar: large scale HPSG for English
- Other grammars: other frameworks, other languages, teaching.
- LKB system: grammar development environment
- PET system: fast runtime environment
- [incr tsdb()] : test-suite machinery
- MRS semantics
- Redwoods treebank (under development)
- Lexicons, especially multiword expressions (under development)

Why LinGO?

Standardized formalization framework independent

Bidirectionality generate as well as parse

Linguistic coverage test suites include wide range of phenomena

Semantic representation tools for manipulating semantics are included in the LKB

Availability Open source (http://lingo.stanford.edu), multiple OS (Windows, Linux, Solaris), LKB is documented (Copestake, 2002).

Scale and Efficiency

Compatible set of tools

Some of the problems mentioned above are still issues, but we have partially solved some and have approaches to the others.

Demo of LKB and ERG

Overview of lecture 2

Implementing and using constraint-based grammars:

- Formalism issues (inheritance, type constraints)
- Semantic representation
- Generation

The main point of this lecture is to provide some idea of grammar engineering with constraint-based systems: the tools available and the challenges involved.

Why grammar engineering?

Building grammars for real NLP system requires a mixture of linguistic intuition and coding skills:

- Adapting existing analyses
- New phenomena
- Efficiency: analyses should not be multiplied beyond necessity (underspecification may help)
  But cute tricks cause later problems . . .
- Generation as well as parsing: no overgeneration
  (But for robustness: allow common errors)
- Grammars are complicated pieces of code: documentation, source control, testing etc.
- ‘Avoid redundancy’ $\equiv$ ‘Capture generalizations’

Inheritance

Organization of lexical items by inheritance is common in CBG formalisms. In lexicalist approaches, complexity is in the lexicon rather than grammar rules, but then the lexicon must be organized.

For example, consider the CFG:

```
VP -> Vsimple-trans NP
VP -> Vvping VPing
VP -> Vpp PP
```

A lexicalist grammar can replace these (and others) with a single rule, schematically:

```
VP -> Vx X*
```

where the compatibility between Vx and X* (i.e., zero or more constituents) is guaranteed by unification.

Inheritance, continued (1)

‘`VP -> Vx X`’ rule in TDL description language (only one X, because of parser):

```
head-complement-rule-1 := phrase &
[ HEAD #head,
  SPR #spr,
  COMPS < >,
  ARGS < word &
         [ HEAD #head,
           SPR #spr,
           COMPS < #nonhddtr > ],
         #nonhddtr >  ].
```

Inheritance, continued (2)

Entry for *chase* without inheritance:

```
chase := word &
[ ORTH "chase",
  HEAD verb,
  SPR < phrase &
       [ HEAD noun,
         SPR <>] >,
  COMPS < phrase &
          [HEAD noun,
           SPR <>] > ].
```

Use inheritance to avoid redundancy:

```
chase := trans-verb &
[ ORTH.LIST.FIRST "chase" ].
```

The type **trans-verb** contains the rest of the information needed.

Type constraints

The expanded type constraint for **trans-verb**:

$$\left[\begin{array}{l}
\textbf{trans-verb} \\
\text{ORTH } \textbf{string} \\
\text{HEAD } \textbf{verb} \\
\text{SPR} < \left[\begin{array}{l}
\textbf{phrase} \\
\text{HEAD} \left[\begin{array}{l} \textbf{noun} \\ \text{NUMAGR } \textbf{agr} \end{array}\right] \\
\text{SPR} < > \\
\text{COMPS} < >
\end{array}\right] > \\
\text{COMPS} < \left[\begin{array}{l}
\textbf{phrase} \\
\text{HEAD} \left[\begin{array}{l} \textbf{noun} \\ \text{NUMAGR } \textbf{agr} \end{array}\right] \\
\text{SPR} < > \\
\text{COMPS} < >
\end{array}\right] > \\
\text{ARGS } \textbf{*list*}
\end{array}\right]$$

Constraint specifications

```
trans-verb := verb-lxm &
[ COMPS < phrase & [HEAD noun, SPR <>] > ].

intrans-verb := verb-lxm &
[ COMPS < > ].

verb-lxm := lexeme &
[ HEAD verb,
  SPR < phrase & [HEAD noun, SPR <>] > ].
```

Well-formedness conditions

Constraint Each substructure of a well-formed TFS must be subsumed by the constraint corresponding to the type on the substructure’s root node.

Appropriate features The top-level features for each substructure of a well-formed TFS must be the appropriate features of the type on the substructure’s root node.

Consistent inheritance The constraint on a type must be subsumed by the constraints on all its parents.

Types vs simple inheritance

- Inheritance operates on all levels of a TFS — hierarchies of rules, lexical signs and subparts of sign.
- Some errors are picked up automatically:

  ```
  trans-verb := verb-lxm &
  [ COMPS < phrase & [HED noun, SPR <>] > ].
  ```

  Moving the ERG from a somewhat typed system (DISCO/PAGE) to the LKB (strong typing) was time-consuming, but identified many grammar bugs.

Formal semantics and computational semantics: shared concerns.

1. Adequacy / coverage
2. Link to syntax: compositionality
3. Formalizability / declarativity

Computational needs:

1. Application needs (mentioned last lecture)
2. Computational tractability: construction, equivalence checking, inference, support for generation. ‘monotonicity’ (i.e., never lose semantic information during composition)
3. Portability and flexibility — ideally want parser/generator as a module that can be hooked up to multiple systems.
4. Breadth — computational semantics cannot ignore any frequent phenomena!

Scope ambiguity

Every cat was chased by an animal

$\forall x[\text{cat}'(x) \Rightarrow \exists y[\text{animal}'(y) \wedge \text{chase}'(y,x)]]$

$\exists y[\text{animal}'(y) \wedge \forall x[\text{cat}'(x) \Rightarrow \text{chase}'(y,x)]]$
i.e., every cat was chased by a specific animal

Generalized quantifier notation:

$\text{every}'(x, \text{cat}'(x), \text{a}'(y, \text{animal}'(y), \text{chase}'(y,x)))$

$\text{a}'(y, \text{animal}'(y), \text{every}'(x, \text{cat}'(x), \text{chase}'(y,x)))$

Every cat doesn’t sleep

$\neg[\forall x[\text{cat}'(x) \Rightarrow \text{sleep}'(x)]]$
$\text{not}'(\text{every}'(x, \text{cat}'(x), \text{sleep}'(x)))$

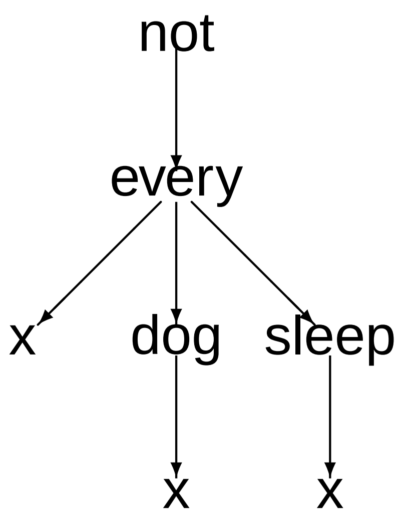
$\forall x[\text{cat}'(x) \Rightarrow \neg\text{sleep}'(x)]]$
$\text{every}'(x, \text{cat}'(x), \text{not}'(\text{sleep}'(x)))$

Scope ambiguity and underspecification

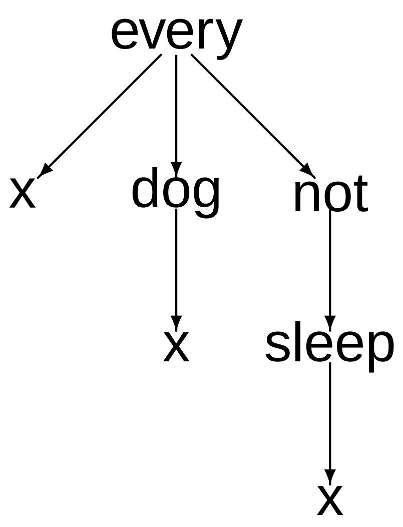
- The composition problem: on the rule-to-rule hypothesis, how do we generate multiple scopes without syntactic ambiguity?
- What do we do with all the possible readings if we generate all scopes?
  Generally a sentence with n quantifiers will have $n!$ readings.

  Not all these readings will be semantically distinct:
  A cat was chased by an animal
  a$'(x,$ cat$'(x),$ a$'(y,$ animal$'(y),$ chase$'(y,x))$
  a$'(y,$ animal$'(y),$ a$'(x,$ cat$'(x),$ chase$'(y,x))$
  but there's no way of generating one reading and not the other compositionally.
- The solution generally adopted is to underspecify scope so a single representation covers all valid readings.
  (LUNAR, QLF in the CLE, UDRT, VIT, MRS)

LFs as trees

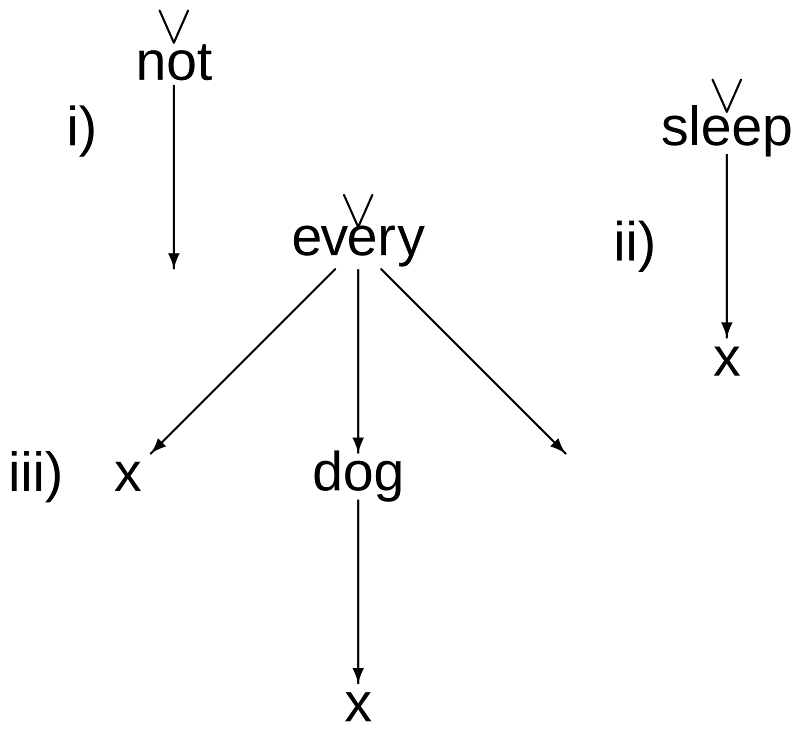


not(every(x,dog(x),sleep(x)))



every(x,dog(x),not(sleep(x)))

Underspecification as partial description of trees



Holes and labels

$$\begin{tikzcd}
\text{i)} & \begin{array}{c}\text{l1}\\ \text{not}\end{array} \arrow[d] \\
& \text{h2}
\end{tikzcd}$$

$$\begin{tikzcd}
& \begin{array}{c}\text{l5}\\ \text{sleep}\end{array} \arrow[d] \\
\text{ii)} & \text{x}
\end{tikzcd}$$

$$\begin{tikzcd}
& & \begin{array}{c}\text{l3}\\ \text{every}\end{array} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{iii)} & \text{x} & \text{dog} \arrow[d] & \text{h4} \\
& & \text{x} &
\end{tikzcd}$$

Elementary predications

$$\begin{tikzcd}
\substack{\text{l1}\\ \text{not}} \arrow[d, swap, "\text{i)}"] & & & \substack{\text{l5}\\ \text{sleep}} \arrow[d, swap, "\text{ii)}"] \\
\text{h2} & \substack{\text{l3}\\ \text{every}} \arrow[dl, swap, "\text{iiia)}"] \arrow[d] \arrow[dr] & & \text{x} \\
\text{x} & \text{h7} & \text{h4} & \\
 & \text{h7=l6} & & \substack{\text{l6}\\ \text{dog}} \arrow[d, swap, "\text{iiib)}"] \\
 & & & \text{x}
\end{tikzcd}$$

MRS

l1:not(h2), l5:sleep(x), l3:every(x,h7,h4), l6:dog(x), h7=l6

Two valid possible sets of equations, shown here with the equivalent scoped structures:
l1:not(h2), l5:sleep(x), l3:every(x,h7,h4), l6:dog(x), h7=l6, h4=l1, h2=l5
every(x,dog(x),not(sleep(x)))
top label is l3

l1:not(h2), l5:sleep(x), l3:every(x,h7,h4), l6:dog(x), h7=l6, h4=l5, h2=l3
not(every(x,dog(x),sleep(x)))
top label is l1

qeq constraints

In general, we can’t equate the restriction of a generalized quantifier with the Nbar because other quantifiers may intervene.

every nephew of a dragon snores

every(x, a(y, dragon(y), nephew(x,y)) snore(y))
i.e., the arbitrary dragon reading

a(y, dragon(y), every(x, nephew(x,y), snore(y)))
i.e., the specific dragon reading

Solution adopted in MRS is to introduce qeq constraints:

l1:every(x,h2,h3), l4:nephew(x,y), l5:a(y,h6,h7), l8:dragon(y), l9:snore(x), h6 $=_q$ l8, h2 $=_q$ l4

qeq constraints, continued

If a hole h is $=_q$ a label l, then one of the following must be true:

- $h = l$
- there is an intervening quantifier, quant, such that quant has a label l' where $l' = h$ and the body of quant is h' (i.e., quant(var,h_r,h')) and $h' = l$
- there is a chain of such intervening quantifiers, all linked via their bodies.

MRS in constraint-based grammars

Labels and holes have to be unifiable: generically referred to as *handles*. An MRS contains:

RELS the collection of elementary predications (cf RESTR in Sag and Wasow etc)

HCONS handle constraints (i.e., the collection of qeqs)

Every EP has a feature HNDL which introduces its label.

The composition rules guarantee that the result of parsing a sentence is a valid MRS with a reasonable set of scopes.

An example MRS in TFSs

$$\left[\begin{array}{l}
\textbf{mrs} \\
\text{RELS} < \left[\begin{array}{l} \text{PRED } \textbf{every\_rel} \\ \text{HNDL } \boxed{2} \textbf{ handle} \\ \text{BV } \boxed{3} \textbf{ ref-ind} \\ \text{RESTR } \boxed{4} \textbf{ handle} \\ \text{BODY } \textbf{handle} \end{array}\right], \left[\begin{array}{l} \text{PRED } \textbf{dog\_rel} \\ \text{HNDL } \boxed{6} \textbf{ handle} \\ \text{ARG0 } \boxed{3} \end{array}\right], \\
\qquad \left[\begin{array}{l} \text{PRED } \textbf{not\_rel} \\ \text{HNDL } \boxed{7} \textbf{ handle} \\ \text{ARG0 } \boxed{8} \end{array}\right], \left[\begin{array}{l} \text{PRED } \textbf{sleep\_rel} \\ \text{HNDL } \boxed{9} \\ \text{ARG1 } \boxed{3} \end{array}\right] > \\
\text{HCONS} < \left[\begin{array}{l} \textbf{qeq} \\ \text{SC-ARG } \boxed{4} \\ \text{OUTSCPD } \boxed{6} \end{array}\right], \left[\begin{array}{l} \textbf{qeq} \\ \text{SC-ARG } \boxed{8} \\ \text{OUTSCPD } \boxed{9} \end{array}\right] >
\end{array}\right]$$

$$\{h2{:}\,\mathsf{every}(x, h4, h5), h6{:}\,\mathsf{dog}(x),$$
$$h7{:}\,\mathsf{not}(h8), h9{:}\,\mathsf{sleep}(x)\},$$
$$\{h4 =_q h6, h8 =_q h9\}$$

Flat semantics for MT

Flat semantic representation (Phillips (1993), Trujillo (1995)) introduced to make writing semantic transfer rules simpler.

For example:
beginning of spring is a translation of the German *Frühlingsanfang*

If we're using conventional scoped representations, it's difficult to write the transfer rule, because the structure depends on the scope.

the beginning of spring arrived
def(x, spring(x), the(y, beginning(y,x), arrive(y)))
the(y, def(x, spring(x), beginning(y,x)), arrive(y))

Phillips' flat semantics ignores the quantifiers:
the beginning of spring arrives
the$(y),$ beginning$(y, x),$ def$(x),$ spring$(x),$ arrive(e, y)

MRS has the same advantages for transfer, because we can drop the handles, but scope can be represented when needed.

Demo of MRS

Generation from logical form

To recap:

licensed structure any structure with substructures that are all made up of structures in the grammar

licensed string any string which can be read off a fully-terminated licensed structure

licensed sentence any licensed string with a corresponding structure that has a start structure root.

For generation, we regard the start structure as including the input semantics, expressed in TFS.

Logical form equivalence

Problem: two different LFs can be logically equivalent.
$\neg[\forall x[\text{cat}'(x) \Rightarrow \text{sleep}'(x)]]$ is logically equivalent to
$\exists x[\text{cat}'(x) \wedge \neg\text{sleep}'(x)]$

Even for first order predicate calculus, the logical form equivalence problem is undecidable. So generation cannot be on the basis of truth-conditions: the actual form of the input LF matters.

BUT, why logical equivalence? Do we want to generate both *it is not the case that every cat sleeps* and *some cat doesn't sleep* from the same input?

The main thing is to avoid bracketing and ordering effects, which can arise because of differences in grammars.

fierce(x) $\wedge$ black(x) $\wedge$ cat(x) $\equiv$ (fierce(x) $\wedge$ black(x)) $\wedge$ cat(x) $\equiv$ cat(x) $\wedge$ (fierce(x) $\wedge$ black(x))

Naive lexicalist generation

1. From the LF, construct a bag of instantiated lexical signs.
2. List the signs in all possible orders.
3. Parse each order.

- Highly independent of syntax
- Requires lexical entries to be recoverable
- Not exactly efficient . . .
- Shake and Bake generation is part of an approach to MT in which transfer operates across instantiated lexical signs
- Chart generation is a similar approach with better practical efficiency

Generation from MRS

- chart generation with some tweaks for additional efficiency
- MRS input makes construction of the bag of signs fairly easy
- still somewhat experimental compared with the parser
- overgeneration is sometimes an issue with the LinGO ERG, mainly with respect to modifier order e.g., *big red box*, ?*red big box*.
- stochastic ordering constraints (under development)
- also machine learning techniques for instantiating underspecified input: e.g., guessing determiners (*a(n)*, *the*, no determiner)

Demo of generation

Overview of lecture 3

- Test suites
- Ambiguity and efficiency

Some research issues:

- Stochastic HPSG and the Redwoods treebank
- Multiword expressions
- Combining deep and shallow processing

Test suites

- Realistic data and constructed data
  - CSLI test suite: constructed by linguists to cover major phenomena (originated at HP). Coverage by phenomenon, regression testing.
  - VerbMobil data: realistic data. Coverage for VerbMobil, realistic efficiency measurements.
- Regression testing: make sure you don't break anything
- Number of parses: avoid spurious readings
- Number of edges: efficiency

Efficiency and grammar engineering

System and formalism choice:

- Formalism choice: richer formalisms can lead to slower systems, but a formalism that is too impoverished can lead to slow development
  - Theoretical efficiency
  - Practical efficiency: replacing feature structure disjunction in the ERG with use of types led to a big increase in efficiency of unification
- Parser choice: e.g., assumptions about word order make parsers faster, but complicate grammars for Japanese etc

Grammar efficiency with LKB and similar systems

- Bigger feature structures mean slower processing
- More edges means slower processing (i.e., ambiguity, even if only local, is bad)
  - Underspecify
  - Only have multiple lexical entries if there's some chance of discriminating between them.
  - Analyses which cut down alternatives 'low down' in the tree, or generate alternatives 'high up', are preferred.
- Small grammars can be much slower than big ones (e.g., textbook grammar compared to ERG)
- ERG is relatively better for parsing than generation

Demo of [incr tsdb()] (‘t’ ‘s’ ‘d’ ‘b’ plus plus) with PET

The ambiguity problem

Lexical ambiguity: not too bad if it can be resolved quickly, exponential increase if it doesn’t get resolved.

Syntactic ambiguity, e.g., PP attachment:

1. he saw the boy with the binoculars. (2 readings)
2. he saw the boy with the binoculars in the forest. (5 readings)
3. he saw the boy with the binoculars in the forest on the mountain. (14 readings)

Catalan series

‘silly’ ambiguities:
Fred Smith is interviewing today

Packing

- Chart parsing with CFGs is n^3 theoretical complexity, despite exponential parses. No internal structure, so e.g., all NPs are equivalent, regardless of embedded PPs
- Much more complex to implement packing in unification based systems (Oepen and Carroll, 2000)
- Requirement to unpack: system using the grammar output still has to disambiguate
- Underspecification is better if the structures don't need to be resolved (e.g., scope ambiguity generally irrelevant for MT), but underspecification is problematic if there's a real syntactic ambiguity

Weights

- disprefer particular rules and lexical items (or classes of lexical item) via manually assigned weights (cf., XLE: ‘optimality theory’?)
- interaction of weights cannot be predicted by humans in practice
- weights are domain-specific
- weights are a temporary hack: stochastic CBG (i.e., learned weights or, preferably, probabilities) is the long-term aim

Towards stochastic HPSG

- Probabilistic CFGs are formally understood
- But CBGs are not simple CFGs: Abney (1997) discusses difficulties with defining a probabilistic approach to CBGs
- Several possible solutions, but a Treebank is a prerequisite for experimentation
- Projects: ROSIE, Deep Thought

Redwoods Treebank

- Manual selection between the parses admitted by the grammar (unlike Penn Treebank) on a realistic corpus
- Selected analyses are stored and used for training of alternative models
- Tools in the LKB to manage treebank building and ease parse selection
- Dynamic: i.e., (somewhat) robust to changes in the grammar
- Existing work reported in LREC 2002 and Coling 2002 workshops, planned work involves larger Treebank.

Multiword expressions

- MWEs: phrases we can't account for by normal generative mechanisms
  - idiosyncratic syntax or semantics (lexicalised phrases)
  - unexpected frequency (institutionalised phrases or collocations)
- Representational issues: especially idioms, collocations.
- Productivity and semi-productivity
- Acquiring MWEs is more difficult than acquiring simplex words
- Funding from NTT and NSF

Types of lexicalised phrase

- idioms
- compound nouns
- verb particles: *look up, wash up*
- syntactically irregular phrases: *on top of*
- words with spaces: *ad hoc*, *of course*
- support verb constructions: *have a shower*

Lexical entries

- Simplex entries consist of orth, syntactic type, semantic predicate
- MWEs are generally more complex (sometimes coded in terms of relationships between simplex entries)
- Acquisition, databases, type hierarchies are all more complex
- Attempting to build reusable MWE databases
- Data sparseness is big issue in automatic acquisition

Combining deep and shallow processing

- Deep and shallow processing aren’t generally in competition but complement each other
- Integration is an issue
- Using MRS-style semantics to integrate looks promising
- EU project ‘Deep Thought’, expected to start October 1 2002

Deep processing: advantages and problems

- Proper coverage of large proportion of phenomena is possible
- Detailed semantic representation
- Generation
- Not as slow as it used to be!

But:

- Not robust to unknown words, missing subcategorization information, ungrammatical input etc
- Not fast enough to process large volumes of text
- High level of ambiguity in longer sentences

Shallow processing

e.g., POS tagging, NP chunking, simple dependency structures.

- Fast
- Robust

But:

- Lack of lexical subcategorization information for open-class words precludes extraction of conventional semantic representations
- Long-distance dependencies
- Insufficient information to recover scope information
- Allows ungrammatical strings

Combining deep and shallow processing

- Patching up failed deep parses
- Shallow parse complete text, use deep parsing on selected sentences
- Shallow parser as preprocessor to guide deep parsing (especially because of punctuation, not covered in most deep grammars)
- Question answering: deep parse questions, shallow parse answer texts

Fine-grained integration

- Integrated shallow and deep parsing, identify specific parts of sentences to deep parse:

  We show that to substantially improve verb sense disambiguation it will be necessary to extract subcategorization information.

- Regeneration:

  Extraction of subcategorization information
  is a prerequisite for substantial improvement in verb sense disambiguation.

Interfacing via semantics

If shallow parsing could return underspecified semantic representations:

- Integrated parsing: shallow parsed phrases could be incorporated into deep parsed structures
- Deep parsing could be invoked incrementally in response to information needs
- Reuse of knowledge sources: information for further processing of shallow parsing might also be used on deep parser output (e.g., recognition of named entities, transfer rules in MT)
- Formal properties should be clearer, representations might be more generally usable

Semantic output from a POS tagger

POS tagger can be viewed as providing some semantic information:

- Lemmatization and morphology, partial disambiguation, BUT no relational information

every_DT0 fat_AJ0 cat_NN1 sat_VVD on_PRP some_DT0 mat_NN1

every_DT$(x1)$, fat_AJ$(x2)$, cat_N$(x3)$, sit_V$(e4)$, past$(e4)$,
on_PRP$(e5)$, some_DT$(x6)$, mat_N$(x7)$

Tag ‘lexicon’ (entries for tags, not lexemes):

DT0	lexrel_DT(x)	AJ0	lexrel_AJ(x)
NN1	lexrel_N(x)	PRP	lexrel_PRP(e)
VVD	lexrel_V(e), past(e)		

Modified syntax for deep representation

Flat semantics, underspecified scope (MRS):

$$h0 : \text{every}(x, h1, h2), h1 : \text{fat}(x), h1 : \text{cat1}(x),$$
$$h4 : \text{sit1}(e, x), h4 : \text{spast}(e), h4 : \text{on2}(e', e, y),$$
$$h5 : \text{some}(y, h6, h7), h6 : \text{mat1}(y)$$

Ignoring scope for now:

$$\text{every}(x), \text{fat}(x), \text{cat1}(x), \text{sit1}(e, x), \text{spast}(e), \text{on2}(e', e, y),$$
$$\text{some}(y), \text{mat1}(y)$$

Parsons' style representation:

$$\text{every}(x), \text{fat}(x), \text{cat1}(x), \text{sit1}(e), \text{spast}(e), \text{on2}(e'),$$
$$\text{some}(y), \text{mat1}(y),$$
$$ARG1(e, x), ARG1(e', e), ARG2(e', y)$$

Modified syntax for deep representation (continued)

Distinct variable names, plus equalities:

every$(x1)$, fat$(x2)$, cat1$(x3)$, sit1$(e4)$, spast$(e4)$, on2$(e5)$,
some$(x6)$, mat1$(x7)$,
$ARG1(e4, x1), ARG1(e5, e4), ARG2(e5, x6)$,
$x1 = x2, x2 = x3, x6 = x7$

Predicate naming convention:

every_DT$(x1)$, fat_AJ$(x2)$, cat_N$(x3)$, sit_V1$(e4)$, spast$(e4)$, on_PRP2$(e5)$,
some_DT$(x6)$, mat_N1$(x7)$,
$ARG1(e4, x1), ARG1(e5, e4), ARG2(e5, x6)$,
$x1 = x2, x2 = x3, x6 = x7$

Where sit_V1 $\sqsubset$ sit_V, spast $\sqsubset$ past, on_PRP2 $\sqsubset$ on_PRP,
mat_N1 $\sqsubset$ mat_N

Underspecification

Tagger output is an underspecified form of the deep representation.

Tagger output was:

> every_DT$(x1)$, fat_AJ$(x2)$, cat_N$(x3)$, sit_V$(e4)$, past$(e4)$,
> on_PRP$(e5)$, some_DT$(x6)$, mat_N$(x7)$

This can be converted into the deep representation by adding ARGn relations and equalities and specialising predicates.

Output from slightly deeper forms of shallow processing is intermediate in specificity.

Specificity ordering

- Defined in terms of the syntax of the semantic representation, not the denotation
- Preliminary informal definition: a semantic representation A is an underspecified form of B if A can be converted to B by:
  1. adding argument-relations (e.g., ARG1)
  2. adding equalities between variables
  3. specialising predicates

Formal properties:

- Treat representations as made up of minimal units of semantic information
- Specificity as a partial order (cf (typed) feature structures)

Combining deep and shallow semantics: conclusions

- Main idea: manipulate the syntax of the semantic representation so it can be split into minimal composition units suitable for a variety of processors
- So far this looks promising, but considerably more work needed to see how plausible it really is
- Don’t want to change hand-built grammars substantially (at least short-term), so MRS to RMRS converter is necessary, although Parsons-style representation might be better for Japanese.
- RMRS might also be useful as an output format for parser evaluation (Briscoe et al, LREC 2002)
- Default composition rules make RMRS attractive as a way of robustly building semantics for automatically-acquired grammars.

Final comments

- Deep processing with hand-built grammars is a sensible approach for some NLP applications but needs good infrastructure: long-term projects, open source, active user community
- Constraint-based grammars can be efficient, reusable, bidirectional
- Lexical acquisition, parse/realization choice and robustness remain problems
- Integrating constraint-based grammars into full applications is difficult (especially for generation): better consensus on semantics is needed
- Deep and shallow processing are not mutually exclusive
- There are many interesting open research issues