An introduction to practical NLP using constraint-based grammars

Ann Copestake

University of Cambridge Computer Laboratory

aac@cl.cam.ac.uk

Overview

- NLP applications
- Deep vs. shallow processing
- Architecture of deep processing systems
- Constraint-based grammar formalisms in NLP applications: why and how?
- LinGO
- Large-scale grammar development
- Semantic representation
- 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 email question answering

Computer ordering domain:

- Has my order number 4291 been shipped yet?
- Is FD5 compatible with a 505G?
- What is the speed of the 505G?

Very similar strings can mean very different things, very different strings can mean much the same thing.

- How fast is the 505G?
- How fast will my 505G arrive?
- Please tell me when I can expect the 505G I ordered.

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

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?

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:

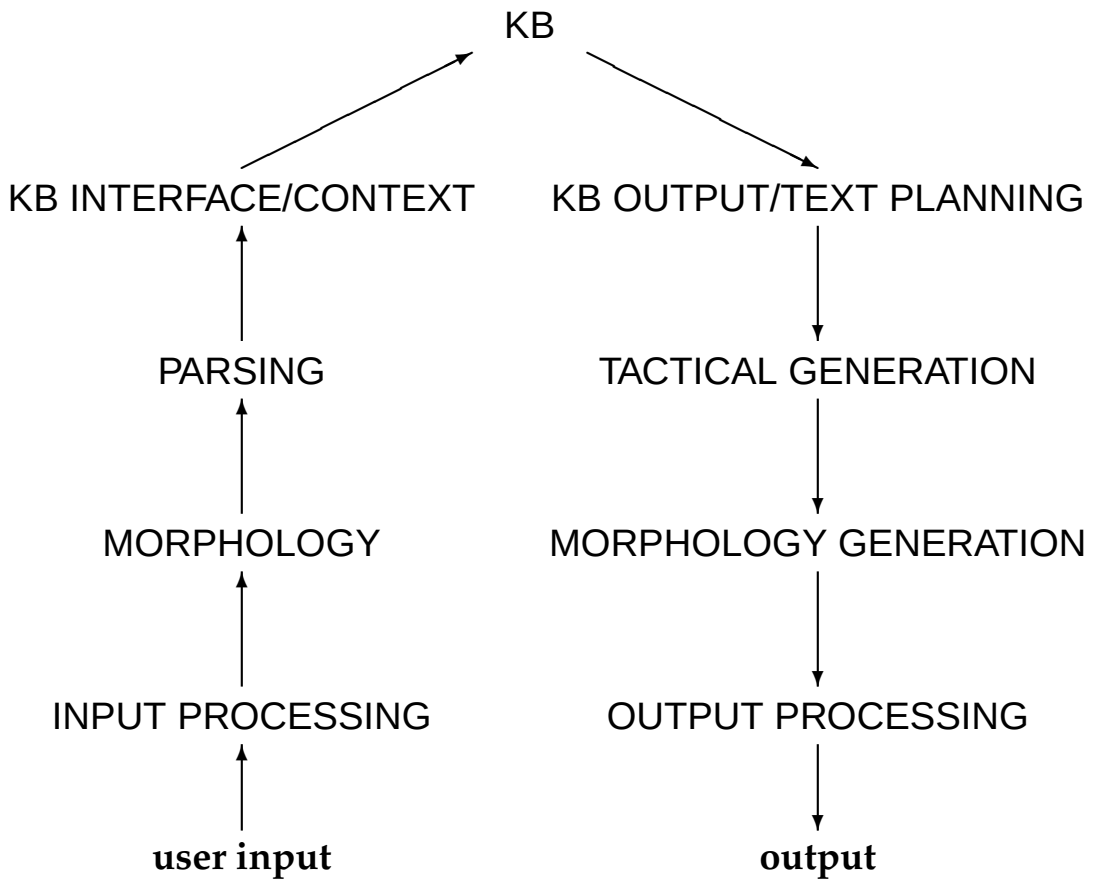
1. USER QUESTION: Have you shipped 4291?
2. SEMANTIC REP: ynq(2pers(y) and def(x, id(x,4291), ship(e,y,x) and past(e)))
3. DB QUERY: order(number=4291,date_shipped=?)

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

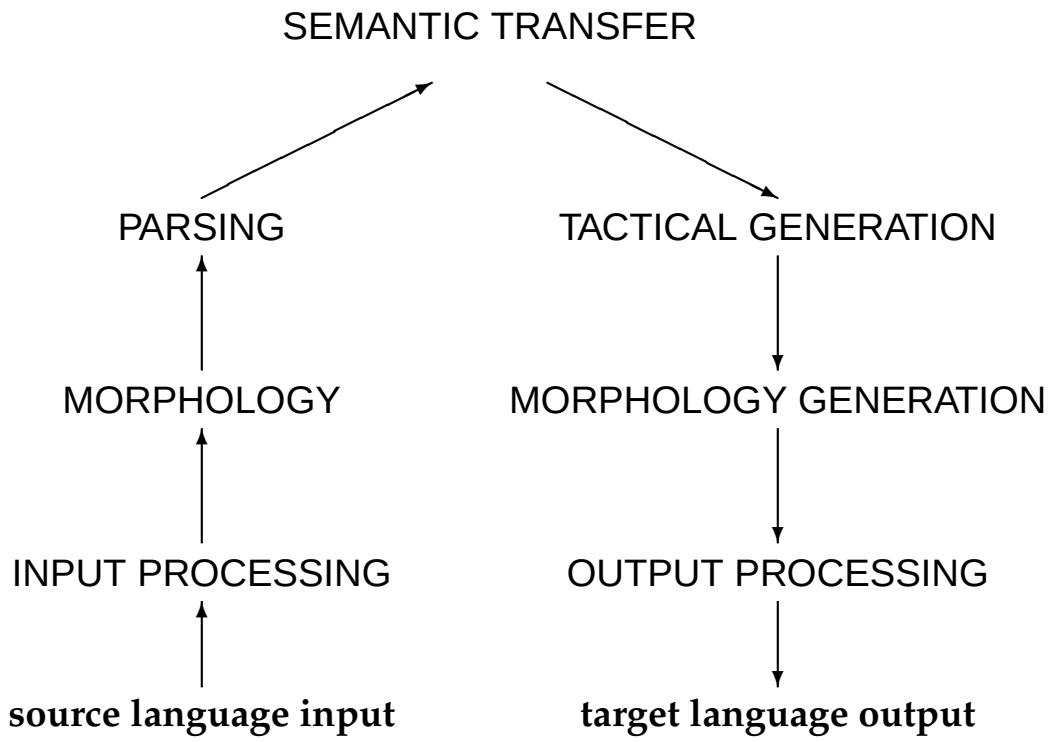
Generic NLP application architecture

- 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. Induced probabilistic grammars
3. Non-constraint-based ‘deep’ grammars: Augmented transition networks, PNLP (Jensen and Heidorn), LSP (Sager)
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.

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 (though compared with what?)

Training grammar developers

Efficiency

Coverage on real corpora 'peripheral' phenomena, lexicons

Ambiguity

Development of good broad-coverage 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)

Core LinGO-related projects

1. Verbmobil: German research project on spoken language machine translation (1994–2000)
2. Natural language generation for a speech prosthesis: NSF research project (1997–2001)
3. YY Software: commercial email response
4. Multiword expressions. NTT and NSF funding (2001–2005)
5. ROSIE: incorporating statistical information. Scottish Enterprise (2002–2004)
6. Deep Thought: linking deep processing with robust, shallow processing. EU (2002–2004)

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

But also note: ANLT (Cambridge, Lancaster, Edinburgh, Sussex), CLE (SRI, no longer ongoing), ParGram (Xerox), XTAG (UPenn and others).

Open Source for deep language technology

- Long-term collaboration between small groups in many countries is essential
- Incremental development is required
- Collaboration is between individuals, not organisations
- Standards emerge based on distributed technology
- Teaching, research and commercial applications sharing same tools
- BUT: only really works if there’s a serious commitment to software maintenance etc

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'

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.
  - email data
- 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

Semantics for Computational Grammars

Formal semantics and computational semantics: shared concerns.

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

Computational needs:

1. Application needs
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
  $\mathsf{a}'(x, \mathsf{cat}'(x), \mathsf{a}'(y, \mathsf{animal}'(y), \mathsf{chase}'(y, x))$
  $\mathsf{a}'(y, \mathsf{animal}'(y), \mathsf{a}'(x, \mathsf{cat}'(x), \mathsf{chase}'(y, x))$
  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)

MRS

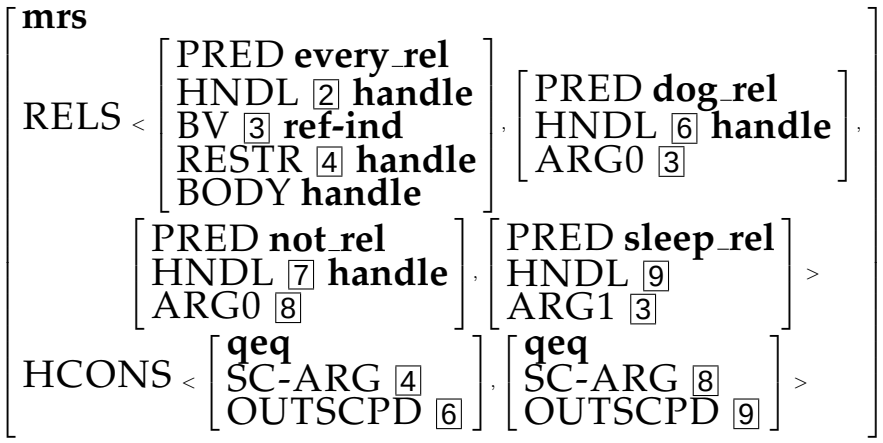
l1:not(h2), l5:sleep(x), l3:every(x,h7,h4), l6:dog(x), h7=l6

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

The current MRS 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



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

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
$\text{the}(y), \text{beginning}(y, x), \text{def}(x), \text{spring}(x), \text{arrive}(e, y)$

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

Demo of LKB and ERG

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 . . .