Grammar Engineering

March 28, 2005

Introduction, overview

Overview

- The BIG Picture
- The LinGO Grammar Matrix
- Other approaches
- Course requirements/workflow
- Pick a language, any language
- Goals (of grammar engineering, this course)
- Best practices

The BIG Picture: Precision Grammars

- relate surface strings to semantic representations
- distinguish grammatical from ungrammatical sentences
- knowledge engineering approach to parsing
- can be used for both parsing and generation

The BIG Picture: Applications

- language documentation/linguistic hypothesis testing
- machine translation
- automated email response
- augmentative and assistive communication
- computer assisted language learning
- IR (from structured or unstructured data)
- …

The BIG Picture: Hybrid approaches (1/2)

- Naturally occurring language is noisy
  - Typos
  - “mark-up”
  - Addresses & other non-linguistic strings
  - False starts
  - Hesitations
  - …
- Allowing for the noise within the grammar would reduce its precision
- And then there’s ambiguity, unknown words, ...

The BIG Picture: Hybrid approaches (2/2)

- Combine symbolic (aka deep) and stochastic (aka shallow) approaches:
  - Statistical parse selection
  - (Statistical) named entity recognition and POS tagging in a preprocessing step (for unknown word handling)
  - Tiered systems with a shallow parser as a fall back for the precision parser
- Coming the other direction, deep grammars can provide richer linguistic resources for training statistical systems (e.g., MT systems).

The LinGO Grammar Matrix (1/3)

- One of the primary impediments to deploying precision grammars is that they are expensive to build.
- The Grammar Matrix aims to address this by providing a starter-kit which allows for quick initial development while supporting long-term expansion.
- The Grammar Matrix also represents a set of hypotheses about cross-linguistic universals.

The LinGO Grammar Matrix (2/3)

- A sampling of hypotheses:
  - Words and phrases combine to make larger phrases.
  - The semantics of a phrase is determined by the words in the phrase and how they are put together.
  - Some rules for phrases add semantics, and some don’t.
  - Most phrases have an identifiable head daughter.

The LinGO Grammar Matrix (3/3)

- More hypotheses:
  - Heads determine which types of arguments they require, and how they combine semantically with those arguments.
  - Modifiers determine which kinds of heads they modify, and how they combine semantically with those heads.
  - No lexical or syntactic rule can remove semantic information.

Course requirements/workflow (1/2)

- Over 9 weekly lab exercises, each student will build a Matrix-based grammar of a different language.
- On Mondays, I'll announce what you need to have prepared in order to do the Wednesday lab.
- Class time on Wednesdays will be lab time, to start each exercise.
- Labs are due (submitted via E-Submit) notionally on Fridays, effectively by midnight Sunday night.

Course requirements/workflow (2/2)

- Make use of EPost!
- There are no required readings, but if you do not have a strong background in syntax, I strongly recommend Sag et al 2003.
- Copestake 2002 provides an extensive introduction to the LKB.

http://courses.washington.edu/ling471

Pick a language, any language (1/2)

- Each student must pick a different language.
- No English.
- Undergrads have priority for languages they already know.

Pick a language, any language (2/2)

- Languages with non-Latin alphabets will need to be done in translation (sorry)
- Languages with complex morphophonology might require some fudging (sorry again)
- If you aren't working on a language you already know, pick a language with a good descriptive or teaching grammar available.

Other approaches

- The LinGO consortium specializes in large HPSG grammars.
- Other broad-coverage precision grammars have been built in/by/with:
  - LFG (ParGram: Butt et al 1999)
  - F/XTAG (Doran et al 1994)
  - ALE/Controll (Götz & Meurers 1997)
- Proprietary formalisms at Microsoft and Boeing.

Goals: of Grammar Engineering

- Build useful, usable resources
- Test linguistic hypotheses
- Represent grammaticality/minimize ambiguity
- Build modular systems: maintenance, reuse

Goals: of this course

- Mastery of tfs formalism
- Hands-on experience with grammar engineering
- A different perspective on natural language syntax
- Practice building (and debugging!) extensible system
- Contribute to on-going research on multilingual grammar engineering

Best practices

- Incremental development
- Regression testing (testsuites, target corpora)
- Documenting code
- Minimizing redundancy

Incremental development

- You’ve gotta start somewhere
- Labs will guide this process, covering a small number of phenomena each week
- Even still, there may be a need to pare back, and not implement everything you can find
- Go ahead and add things to your testsuite that you don’t expect to be able to cover
- Document what you are and aren’t covering
- Run testsuites frequently in order to check whether any coverage from earlier labs is lost (it shouldn’t be)

Constructing testsuites

- Purpose: Catch (and diagnose) bad interactions between analyses of different phenomena
- Positive and negative examples
- More negative examples than positive ones: map out ungrammatical terrain around each grammatical example
- Organized by phenomenon, annotated by phenomenon
- As the testsuite gets more elaborate, represent combinations of phenomena
- Build incrementally as coverage expands
- Have one general, relatively static test suite to track development over time

Documenting code

- Lab write-ups will provide one kind of documentation
- In-line documentation is also essential.
- Comment character: `;`
- Include a date with every comment.
- Indicate what you decided to do, and why, including example sentences as appropriate.
- Don’t erase old comments, even if you have since decided to change the piece they were documenting. You might just have to come back to it again...

Minimizing redundancy

- Type hierarchies allow us to state constraints in just one place and have them be inherited by many objects.
- Take advantage of this: Constraints stated only once need to be fixed in only one place.
- This is often motivation for the creation of subtypes

Debugging Best Practices (1/2)

- Bugs may reside in: your grammar, the Matrix, the LKB
- When you have a working system, test (either individual sentences or testsuites) after each change
- If you have to make a lot of changes all at once, save a copy of the working system to back out to
- If you encounter a bug, make sure you can reproduce it.
- Explore the conditions that produce the bug

Debugging Best Practices (2/2)

- Keep track of what you've tried in fixing the bug
- Consider backing out of 'fixes' that don't help
- Check the FAQ
- Check that your numlock key is off
- Post to EPost

Technical note: Types v. instances

- The LKB distinguishes between *types* and *instances*.
- Instances are the maximally specific items in the hierarchy which the parser/generator can use in processing sentences.
- Types are used in the definition of instances.
- Types can have multiple parents.
- Instances can only have one parent.

Overview

- The BIG Picture
- The LinGO Grammar Matrix
- Other approaches
- Course requirements/workflow
- Pick a language, any language
- Goals (of grammar engineering, this course)
- Best practices