October 28, 2004

Unification

Midterm Review

Overview

- Where we are (with unification)
- Unification algorithm (lisp and pseudocode)
- Integrating unification into the Earley parser
- Some other issues with unification
- Mid-quarter review: What we’ve covered

Where we are with unification

- Feature structures, types
- Feature structures are good for capturing generalizations (e.g., agreement, subcategorization)
- Feature structures can be useful for harder things: long distance dependencies, compositional semantics
- Unification: Fundamental operation with feature structures
- Check for computability, return either fail or the combined set of constraints.

Unification algorithm

- Recursive. Why?
- What are the base cases?
- Destructive. What does this mean? What is the benefit?

Pointers/Contents/Dereferencing

- Complicate feature structures by adding pointer/content arcs at every level.
- If pointer is null, value at end of content arc is actual contents.
- Otherwise, follow pointer.content.
- The process of finding the actual contents is called ‘dereferencing’.

Unification algorithm: outline (1/2)

- Input is two dags, output is one dag
- For two complex, non-identical feature structures:
    - Set pointer of F1 to F2.
    - Recursively check whether value of each feature in F1 unifies with the value of that feature in F2.
    - If the feature isn’t found in F2, create it there, with null value.
    - What about features in F2 but not in F1?

Unification algorithm: outline (2/2)

- Base cases:
    - If value in one case (f1) is null, set the pointer of that case to the other case, and return other (f2).
    - If the values are identical (not just compatible!), set pointer of f1 to f2 and return f2.
    - If the values are atomic, nonnull, and not identical, return failure.

Unification in Lisp

- How does the Lisp code on the following slides differ from the pseudocode given in the book?

Unification in Lisp: representing DAGs

```
(defstruct dag
  forward type arcs copy)

(defstruct arc
  feature value)
```

Unification in Lisp (1/3)

```
(defun unify (dag1 dag2)
  (catch :fail (unify1 dag1 dag2)))
```

Unification in Lisp (2/3))

```
(defun unify1 (dag1 dag2)
  (let* ((dag1 (deref dag1))
         (dag2 (deref dag2)))
    (unless (eq dag1 dag2)
      (let ((glb (glb (dag-type dag1) (dag-type dag2))))
        (when (null glb) (throw :fail nil))
        (setf (dag-forward dag1) dag2)
        (setf (dag-type dag2) glb)
        (loop ... [see next slide])
    dag2))
```

Unification in Lisp (3/3)

```
(loop
    for arc1 in (dag-arcs dag1)
    for arc2 = (loop
                   for foo in (dag-arcs dag2)
                   when (eq (arc-feature foo)
                            (arc-feature arc1))
                   return foo)
    when arc2 do
      (unify1 (arc-value arc1) (arc-value arc2))
    else do
      (setf (dag-arcs dag2) (cons arc1 (dag-arcs dag2))))))
```

Integrating unification into the Earley parser

- Three changes:
    - Add DAGs to edge representations
    - Make COMPLETER check whether the fs of the completed edge is compatible with the daughter it (apparently) matches in each incomplete edge.
    - Make ENQUEUE check for subsumption, not equality.
- Why are there no changes to PREDICTOR or SCANNER?

A more radical approach to parsing with unification

- Replace category labels completely with feature structures (this is what's done in the LKB).
- Allows rules to UNDERSPECIFY the information that would have been in the category labels, and instead constraint only semantics, or only identify of category, or ...

A still more radical approach to parsing with unification

- Do away with CFG ‘backbone’ altogether.
- Take advantage of a recursive feature like ARGS (cf assignments 2 and 3).
- Parsing can be seen as successively resolving the ARGS values to fully specified feature structures.

Packed charts and unification

- Check for subsumption rather than equality.
- Leave ‘accumlator’ features (RELS, HCONS) out of the comparison.
- Unpacking becomes a bit more complicated.

Summary

- Unification algorithm
- Unification in Earley parser
- More radical approaches to unification-based parsing
- Unification and packed charts
- Now: Review

Notes on the exam

- Open book, open notes, closed web
- Covers all material discussed so far

Synthesis

- What is computational linguistics?
- How does it differ from other subfields of linguistics/CS?
- How is it similar to other subfields of linguistics/CS?

Topics covered so far

- Regular expressions
- Finite state automata
- Finite state transducers
- Morphology & morphological parsing
- CFG
- Syntactic parsing
- Feature structures
- Unification
- Parsing with unification

Formal languages

- A formal language is a set of strings
- Things you can do with a formal language:
  - Recognize it
  - Parse it
  - Generate it

Knowledge bases

- Knowledge bases are encodings of (linguistic) information.
- What kinds have we seen in this class?
  - What formal systems to they use?
  - What do they encode?

Using knowledge bases

- Knowledge bases can be used by various algorithms to:
  - recognize
  - parse
  - generate
- ... sets of strings.
- Which algorithms have we seen for each, and what knowledge bases do they use?

Formal devices

- What kinds of formal devices have we seen so far?
- What are the relationships among them?
- What kinds of operations are appropriate for each?

Topics covered so far

- Regular expressions
- Finite state automata
- Finite state transducers
- Morphology & morphological parsing
- CFG
- Syntactic parsing
- Feature structures
- Unification
- Parsing with unification