Topics in Computational Linguistics — Parsing and Generation —

Stephan Oepen
Dan Flickinger

Center for the Study of Language and Information
Stanford University — Spring Quarter 2003

{oe|dan}@csli.stanford.edu

http://lingo.stanford.edu/courses/03/pg/
Review: Feature Structure Unification & Copying

Basic Notions
- Typed feature structures encoded as *directed acyclic graphs* (DAGs);
- each node bears a *type* and a set of *arcs* (aka features – value pairs);
- feature structure reentrancy (coreference) corresponds to DAG identity;
- unification creates *equivalence classes*, encoded through *forwarding*.

Basic Operations
- *unify()* — make two DAGs equivalent, check and combine all information;
  → at each node, *glb()* types, forward, recurse over and accumulate arcs;
- *copy()* — create *structurally equivalent* copy (preserving reentrancies);
  → at each node, *copy* slot as short-term memory, reset upon completion.
Feature Structure Reentrancy (AVM)

Parsing and Generation (59)
Feature Structure Reentrancy (DAG)

```
phrase -> verb
  HEAD
  ARGs
  *ne-list*

word -> "chased"
  FIRST
  ORTH
  syn-struc
  noun
  FIRST
  HEAD
  REST
  *null*  
```

ABabcdfghiejkl

stanford — may-03 (oe@csli.stanford.edu)

Parsing and Generation (60)
The Costs of Feature Structure Manipulation

Basic Cost Measure

- Visit each DAG node once (node operations ‘constant’): \textit{full traversal};
- \textit{linear} in the number of nodes $\rightarrow$ upper bound is size of largest DAG.

Naïve Complexity Theory

- Prior to each (destructive) unification, make copies of both input DAGs;
- upon completion of each copy, recursively reset \textit{copy} slot on all nodes.

<table>
<thead>
<tr>
<th></th>
<th>restore()</th>
<th>copy()</th>
<th>unify()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>
# The unify() vs. copy() Trade-Off

## Destructive Unification [Boyer & Moore, 1972]
- Permanently alter both input dags: `setf()` on `forward`, `type`, and `arcs`;
- _over copying_ — two full copies required for only one result structure;
- _early copying_ — majority of unifications fail: many unnecessary copies.

## Non-Destructive Unification [Wroblewski, 1987]
- Incrementally build up result DAG during unification, one node at a time;
- eliminates over copying, reduces early copying more or less effectively.

## Quasi-Destructive Unification [Tomabechi, 1991]
- Alter input DAGs in way that is reversible (at small cost): ‘generations’;
- copy out result only after unification success, no over or early copying.
Generation Counting

- Protect DAG slots with *generation* counter → ‘expiration date’ of value;
- access: require valid generation; assignment: set value *and* generation;

→ implemented through interaction of global counter and ADT functionality.

(defstruct dag
    forward type arcs xcopy (generation 0))
(defparameter *generation* 1)
(defun dag-copy (dag)
    (when (= (dag-generation dag) *generation*) (dag-xcopy dag)))
(defsetf dag-copy dag-set-copy)
(defun dag-set-copy (dag value)
    (setf (dag-xcopy dag) value)
    (setf (dag-generation dag) *generation*))
Unification-Based Parsing

Adaptations to CFG-Based Chart Parser

- Make all elements of $\Sigma$, $C$, and $P$ from the grammar feature structures;
- substitute *unification* and *equivalence test* for category comparison;
- unify category of passive edges with *argument position* of active edges;

$\rightarrow$ *edge* structure  
LHS is DAG, RHS list of paths to argument positions;

$\rightarrow$ *fundamental-rule()*  
result of unification is category for new edge;

$\rightarrow$ *pack-edge()*  
equivalence test: two DAGs contain same information;

- test *spanning* passive edges for compatibility against start symbol $S$.

#E[ id: \((i-j)\) dag --> edge_1 \ldots edge_i . \ path_{i+1} \ldots \ path_n \{ \text{alternates} \}^* ]

#E[42: (0-8) head-specifier-rule --> 13 . (ARGS REST FIRST)]
Reminder: Categories in a Unification Grammar

- All (constituent) categories in the grammar are typed feature structures;
- specific TFS configurations may correspond to ‘traditional’ categories;
- labels like ‘S’ or ‘NP’ are mere abbreviations, not elements of the theory.

\[
\begin{align*}
\text{word} & : \begin{cases}
\text{HEAD} & \text{noun} \\
\text{SPR} & \langle \rangle \\
\text{COMPS} & \langle \rangle 
\end{cases} \\
\text{phrase} & : \begin{cases}
\text{HEAD} & \text{verb} \\
\text{SPR} & \langle \rangle \\
\text{COMPS} & \langle \rangle 
\end{cases} \\
\text{phrase} & : \begin{cases}
\text{HEAD} & \text{verb} \\
\text{SPR} & \langle \rangle \\
\text{COMPS} & \langle \rangle 
\end{cases}
\end{align*}
\]

‘N’  ‘VP’  ‘S’
‘lexical’  ‘intermediate’  ‘maximal’
The Format of Grammar Rules in the LKB

```
mother
[HEAD 1
SPR 2
COMPS ⟨⟩
   · ·] → daughter₁
[HEAD 1
SPR 2
COMPS ⟨3⟩, 3
   · ·]
daughter₂
```

```
mother
[HEAD 1
SPR 2
COMPS ⟨⟩
   · ·]
   ARGS ⟨
daughter₁
[HEAD 1
SPR 2
COMPS ⟨3⟩, 3
   · ·]
daughter₂
```
Additional DAG Manipulation Functionality

Unification into Argument Position

- Additional parameter to `unify()`: unify `dag_2` into `dag_1` under `path`:

  ```lisp
  (defun unify (dag1 dag2 &optional path)
    ...)  
  ```

  - empty `path`: regular unification; otherwise find first `path` element in `dag_1`, recurse with corresponding `arc` value from `dag_1`, `dag_2`, and rest of `path`.

Equivalence Test

- Similar to `unify()`: traverse two dags in parallel, but no modifications;

- reentrancies: for each node, record corresponding node from second dag in `copy` slot; non-empty `copy` values need to match current nodes.
Unification-Based Parsing—Practical Concerns

Observations

- Typical systems: 90+ per cent of parsing time go to DAG manipulation;
- most unifications fail: predict unification failure cheaply, where possible;
  → rule filter: rule feeding relations; quick check: most likely failure paths;
- lexicalisation: argument positions in rules may be highly underspecified;
  → head-driven parsing: instantiate RHS bidirectionally, starting from head;
- many unifications fail very early: copy() more expensive than unify();
  → memory is expensive: redo a couple of unifications instead of one copy.

Several orders of magnitude average speed-up by reducing constants
Unification-Based Parsing—Optimizations

Rule Filter
- ‘Specifier – Head’ cannot feed into first argument position of ‘Head – Complement’ (COMPS);
  → precompute rule filter relation;
- fundamental rule checks filter before attempting a unification.

Head-Driven Parsing
- First argument position of ‘Specifier – Head’ cannot fail: large number of active edges;
  → bi-directional rule instantiation: head argument position first.
Suggested Background Activities

- Retrieve the model solution for the fifth exercise from the course site;
- compare our solution to your submission; how is ours better (or not)?
- apologize our tardiness in returning feedback on earlier exercises;
- read [Wroblewski, 1987], be sure to understand over and early copying;
- investigate a call counting scheme for the DAG manipulation routines;
- identify the parts of our earlier parser that require modifications now;
- [Oepen, Flickinger, Tsujii, & Uszkoreit, 2002], Chapters Five and Nine — see whether you can enjoy reading contemporary parsing literature.