Parsing with unification

Bernd Kiefer

Deutsches Forschungszentrum für künstliche Intelligenz

Introduction to Computational Linguistics

Thanks to Frederik Fouvry for the slides

Outline

1. Motivation
2. Unification
3. Other issues
4. References

Insufficiency of *cfgs*

- Atomic categories:
  No relation between the categories in a *cfg*:
  e.g. NP, N, N′, VP, VP_3sg, Nsg
- Hard to express generalisations in the grammar:
  for every rule that operates on a number of different categories, the rule specification has to be repeated

An example

- NP $\rightarrow$ Det N
- NPsg $\rightarrow$ Detsg Nsg
  NPpl $\rightarrow$ Detpl Npl

  Can we throw away the first instance of the rule?
  No: *sheep* is underspecified, just like *the*, ...

  We need to add the cross-product:
- NPsg $\rightarrow$ Detsg N
  NPpl $\rightarrow$ Detpl N
  NPsg $\rightarrow$ Det Nsg
  NPpl $\rightarrow$ Det Npl

An example

- Alternatively, words like *sheep* and *the* could be associated with several lexical entries.
  $\longrightarrow$ only reduces the number of rules somewhat
  $\longrightarrow$ increases the lexical ambiguity considerably

More problems

- The grammar cannot rule out yet: *Those sheep runs*
  → subject-verb agreement is not encoded yet
- Subcategorisation frames in their different stages of saturation are to be done as well.
- However: the expansion could be done automatically from feature structure descriptions: e.g.

$$\begin{bmatrix} \text{CATEGORY} & \textit{noun} \\ \text{SUBCAT} & \langle\rangle \\ \text{NUMBER} & \textit{sing} \\ \text{PERSON} & \textit{3} \end{bmatrix} \rightarrow \text{NP\_3sg}$$

More problems

- The grammar cannot rule out yet: *Those sheep runs*
  $\rightarrow$ subject-verb agreement is not encoded yet
- Subcategorisation frames in their different stages of saturation are to be done as well.
- However: the expansion could be done automatically from feature structure descriptions: e.g.

$$\begin{bmatrix} \text{CATEGORY} & \textit{noun} \\ \text{SUBCAT} & \langle\rangle \\ \text{NUMBER} & \textit{sing} \\ \text{PERSON} & \textit{3} \end{bmatrix} \rightarrow \text{NP\_3sg}$$

More problems

- The formalism does not leave any room for generalisations like the following:
  - “All verbs have to agree in number and person with their subject.”
    S $\rightarrow$ NP_(*) VP_(*) \1 $=$ \2
  - “In a headed phrase, the head daughter has the same category as the mother.”
    XP $\rightarrow$ Y X
- Feature structures can do that.
- When a feature structure stands for an infinite set of categories, the grammar cannot be compiled out into a *cfg*.

Part II

Definitions

Outline

2 Definitions
- What is a feature structure?
- What is unification?

3 Parsing

4 Efficiency techniques

Outline

2 Definitions
- What is a feature structure?
- What is unification?

3 Parsing

4 Efficiency techniques

Definition

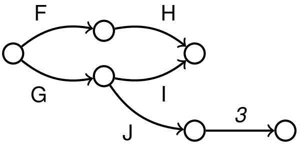
A feature structure is a directed graph, consisting of nodes and labelled edges. One node is special: the *root node*, from which every node can be reached by following edges.

A feature structure is a tuple $\langle Q, \overline{q}, \delta \rangle$:

- Q is a finite set of nodes, rooted at $\overline{q}$
- $\overline{q} \in Q$ is the root node
- $\delta : \text{Feat} \times Q \rightarrow Q$: a partial feature value function

Notation

- As a graph



- As an *avm*

$$\begin{bmatrix} \text{F} \mid \text{H} & \boxed{1} \\ \text{G} & \begin{bmatrix} \text{I} & \boxed{1} \\ \text{J} & \mathit{3} \end{bmatrix} \end{bmatrix}$$

Outline

2 Definitions
- What is a feature structure?
- What is unification?

3 Parsing

4 Efficiency techniques

Subsumption

- An order relation between elements of a set:
  $\sqsubseteq: P \times P$ $\quad\quad\quad\quad \langle P, \sqsubseteq \rangle$
- It is an information ordering:
  a subsumes b iff a contains less information than b,
  alternatively iff a is more general than b.
- Special cases
  - There may be elements a, b such that $a \not\sqsubseteq b$ and $b \not\sqsubseteq a$ (*incomparable*)
  - Each element subsumes itself
    $a \sqsubseteq b \wedge b \sqsubseteq a \Leftrightarrow a = b$
  - In an anti-chain, no two elements are comparable

- Unification is the operation of merging information-bearing structures, without loss of information *if* the unificands are consistent (monotonicity).

Feature structure unification

- Here, $\sqsubseteq$ is a relation in the set of feature structures
- Feature structure unification ($\sqcup$) is the operation of combining two feature structures so that the result is the most general feature structure that is subsumed by the two unificands (*the least upper bound*). If there is no such structure, then the unification *fails*.
- Two feature structures that can be unified are compatible (or consistent). Comparability entails compatibility, but not the other way round.
- There is untyped feature structure unification and typed feature structure unification.

Untyped feature structure unification

- Token-identity: two feature structures are token-identical iff they *are* the same object.
- Consistent/compatible: two feature structures are consistent if they
  - have the same value,
  - the values of their common features are consistent.

Untyped unification: examples

See also Shieber (1986)

- $\begin{bmatrix}\text{CATEGORY} & \textit{noun}\end{bmatrix} \sqcup \begin{bmatrix}\text{NUMBER} & \textit{singular}\end{bmatrix} = \begin{bmatrix}\text{CATEGORY} & \textit{noun} \\ \text{NUMBER} & \textit{singular}\end{bmatrix}$
- $\begin{bmatrix}\text{CAT} & []\end{bmatrix} \sqcup \begin{bmatrix}\text{CAT} \mid \text{CASE} & \textit{accusative}\end{bmatrix} = \begin{bmatrix}\text{CAT} \mid \text{CASE} & \textit{accusative}\end{bmatrix}$
- $\begin{bmatrix}\text{F} & \boxed{1} \\ \text{H} & \boxed{1}\end{bmatrix} \sqcup \begin{bmatrix}\text{F} & [] \\ \text{H} \mid \text{G} & []\end{bmatrix} = \begin{bmatrix}\text{F} & \boxed{1}\begin{bmatrix}\text{G} & []\end{bmatrix} \\ \text{H} & \boxed{1}\end{bmatrix}$
- $\begin{bmatrix}\text{CATEGORY} & \textit{noun}\end{bmatrix} \sqcup \begin{bmatrix}\text{CATEGORY} & \textit{verb}\end{bmatrix} = \text{fail}$

Untyped unification: examples

- $$\begin{bmatrix} \text{AGR} & \boxed{1}\begin{bmatrix} \text{NUM} & \textit{sg} \end{bmatrix} \\ \text{SUBJ} & \begin{bmatrix} \text{AGR:} \ \boxed{1} \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \text{SUBJ} & \begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{PERS} & \textit{third} \end{bmatrix} \end{bmatrix} \end{bmatrix} =$$
$$\begin{bmatrix} \text{AGR} & \boxed{1}\begin{bmatrix} \text{NUM} & \textit{sg} \\ \text{PERS} & \textit{third} \end{bmatrix} \\ \text{SUBJ} & \begin{bmatrix} \text{AGR} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

Destructive and non-destructive unification

In implementations, there are two ways to perform unification:

- Destructive unification: in the process of unifying two structures, one is modified and will contain the result
- Non-destructive unification: the unificands are not changed, and the result is a totally new structure.

The former is faster, but gives undesirable effects in some cases. For instance, when you apply a grammar rule, you do not want the rule to be different after the application. Non-destructive unification is easier to keep track of, but requires copying. Because it does not change the feature structures, the latter is used in implementations.

Typed unification

- Type-identity: two object are type-identical iff they are of the same type.
- Consistent: two feature structures are consistent if
  - their type values are consistent
  - their features have consistent values.

Type hierarchies

- A type hierarchy is a partially ordered set $\langle \text{Type}, \sqsubseteq \rangle$
- Often type hierarchies have to obey the *bounded complete partial order* requirement:
  "For every set of elements with an upper bound, there is a least upper bound."
  It ensures that every unification is unique
- Every feature structure node q has a typed value: $\theta(q)$
- In a type hierarchy, the more specific types inherit all properties from their supertypes. It is not possible to remove a property.

Typed feature structures

- A typed feature structure is a tuple $\langle Q, \overline{q}, \delta, \theta \rangle$:
  - Q is a finite set of nodes, rooted at $\overline{q}$
  - $\overline{q} \in Q$ is the root node
  - δ : Feat $\times$ $Q \rightarrow Q$: a partial feature value function
  - θ : $Q \rightarrow$ Type: a total type assignment function
- Typed feature structures stand in a subsumption hierarchy, the shape of which is determined by the type hierarchy and feature reentrancies. Even though the type hierarchy is finite, the feature structure hierarchy is not necessarily finite.
- It may not be immediately clear a reentrancy contains more information than a structure without. After all: the latter structure has more nodes. A reentrancy adds the knowledge that two things do not only look the same, they *are* the same.

Typed feature structure unification

Let $F, F' \in \mathcal{F}$ and $F = \langle Q, \overline{q}, \theta, \delta \rangle$, $F' = \langle Q', \overline{q}', \theta', \delta' \rangle$. It is required that $Q \cap Q' = \emptyset$. A least equivalence relation $\bowtie$ is defined on $Q \cup Q'$ such that

- $\overline{q} \bowtie \overline{q}'$
- $\delta(f, q) \bowtie \delta(f, q')$ if both are defined and $q \bowtie q'$

Then $F \sqcup F' = \langle (Q \cup Q')/_{\bowtie}, [\overline{q}]_{\bowtie}, \theta^{\bowtie}, \delta^{\bowtie} \rangle$
with

$$\theta^{\bowtie}([q]_{\bowtie}) = \bigsqcup \{ (\theta \cup \theta')(q') | q \bowtie q' \}$$

$$\delta^{\bowtie}(f, [q]_{\bowtie}) = \begin{cases} [(\delta \cup \delta')(f, q)]_{\bowtie} & \text{if } (\delta \cup \delta')(f, q) \text{ is defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

if all joins in $\theta^{\bowtie}$ exist. It is undefined otherwise.
(Carpenter, 1992)

$$\begin{bmatrix} \mathsf{F} & \boxed{1} \\ \mathsf{G} & \boxed{1} \end{bmatrix} \sqcup \begin{bmatrix} \mathsf{F} & a \\ \mathsf{G} & b \end{bmatrix} = \begin{bmatrix} \mathsf{F} & \boxed{1}\ a/b \\ \mathsf{G} & \boxed{1} \end{bmatrix}$$

Feature Appropriateness

- In an untyped framework, feature may be added anytime anywhere: there are no restrictions.
- In typed feature structures, the occurrence of features is limited by the type hierarchy:
  - Each feature is *introduced* on a unique, most general type
  - Only that type and its subtypes can carry that feature
  - Each feature is introduced with a value, and all valid values have to be subsumed by this value.
- These requirements ensure monotonicity in feature structure unification

Parsing with unification-based grammars

- In most implementations, the rules have a context-free backbone, but feature structures in the categories. Information can be shared between the categories in the rule.
- $\begin{bmatrix} \text{CATEGORY} & \textit{noun} \\ \text{SUBCAT} & \langle\rangle \end{bmatrix} \longrightarrow \boxed{1}\begin{bmatrix} \text{CATEGORY} & \textit{det} \end{bmatrix} \begin{bmatrix} \text{CATEGORY} & \textit{noun} \\ \text{SUBCAT} & \langle \boxed{1} \rangle \end{bmatrix}$
- Sometimes the rules are written in a *cfg*-like format, sometimes feature structures whereby a feature identifies the daughters.

Parsing

- Is there any difference in parsing?
- No. All known techniques can be used, and you will obtain a working parser, provided that you use non-destructive unification.
- *But* it will be (much) slower: the categories are much bigger, and the unification is non-destructive. A lot of copying is done.

Techniques to improve efficiency

- Packing (subsumption packing)
- Rule filter: not all rules can feed into all other rules
- Quick check: some paths are more likely to fail than others
- Sharing and deleting of daughters: do not keep information that can easily be (re)computed or retrieved
- Delayed copying (Tomabechi): only copy when you are sure that it will be used

Subsumption packing

- With *cfgs* and chart parsing, every category is only stored once for a given pair of indices to avoid recomputation. The criterion is a simple identity/equality check.
- Suppose we have (among others) the following feature structure in the chart:

$$\begin{bmatrix} \text{CAT} & \textit{noun} \\ \text{AGR} & \begin{bmatrix} \text{PER} & \textit{3} \end{bmatrix} \end{bmatrix}$$

Subsumption packing

- After a rule application, we want to add one of the following feature structures:

  1. $\begin{bmatrix} \text{CAT} & \textit{noun} \\ \text{AGR} & \begin{bmatrix} \text{PER} & \textit{1} \end{bmatrix} \end{bmatrix}$
  2. $\begin{bmatrix} \text{CAT} & \textit{noun} \\ \text{AGR} & \begin{bmatrix} \text{PER} & \textit{3} \\ \text{NUM} & \textit{sg} \end{bmatrix} \end{bmatrix}$
  3. $\begin{bmatrix} \text{CAT} & \textit{noun} \\ \text{AGR} & \begin{bmatrix} \text{NUM} & \textit{sg} \end{bmatrix} \end{bmatrix}$
  4. $\begin{bmatrix} \text{CAT} & \textit{noun} \\ \text{AGR} & [\,] \end{bmatrix}$

Subsumption packing

- Which one the two should we take?
  - all: too many solutions (spurious ambiguity)
  - the first, most recent, . . . : may give over/undergeneration

    e.g. with (4) a solution with $\begin{bmatrix} \text{CAT} & \textit{noun} \\ \text{AGR} & \begin{bmatrix} \text{PER} & \textit{1} \end{bmatrix} \end{bmatrix}$ is also possible, although that does not correspond with the original situation
  - in general: when the newer category is more specific, using it may invalidate older analyses (which were based on a more general feature structure; see (2)), and vice versa

Subsumption packing

- In *cfgs* with atomic catgories, we use an equality check
- With feature structures, we want to *be able* to use unification (it is the operation we use in rule applications), but unification should not be used to perform the check.
- A subsumption check will tell us what is the most general feature structure, and that one should be stored in the chart:
  - if new $\sqsubseteq$ old, then the set of solutions from new will be a superset of the set of solutions from old, so replace old by new.
  - if old $\sqsubseteq$ new, then new should be discarded (it is already implied by old)
  - otherwise, add new.

In this way, no solutions are invalidated.

Part III

Other issues

Statistical processing with feature structures

- Applying statistical techniques to feature structures is very hard, mainly because of the presence of reentrancies (Abney, 1997, See e.g.).
- Very often the following technique is applied: simplify the feature structure, even to the type of the root node only. That way, the categories can be made sufficiently simple. Examples: Bouma et al. (2001); Toutanova et al. (2002)

Default unification

- Credulous default unification: the default FS adds as much information as possible that is not conflicting with the strict FS. It is non-deterministic.
- Sceptical default unification: the default FS adds the information that is common between each variant of credulous default unification.
  (Carpenter, 1993)
- Sensitive to order of processing
- Persistent associative default unification (Lascarides et al., 1996)
- Mainly used for lexical specification

Credulous default unification

- $F \overset{<}{\sqcup}_c G = \{F \sqcup G' | G' \sqsubseteq G$ is maximal such that $F \sqcup G'$ is defined$\}$
- $\begin{bmatrix} \text{F} & a \end{bmatrix} \overset{<}{\sqcup}_c \begin{bmatrix} \text{F} & \boxed{1}\, b \\ \text{G} & \boxed{1} \\ \text{H} & c \end{bmatrix} = \left\{ \begin{bmatrix} \text{F} & a \\ \text{G} & b \\ \text{H} & c \end{bmatrix}, \begin{bmatrix} \text{F} & \boxed{1} a \\ \text{G} & \boxed{1} \\ \text{H} & c \end{bmatrix} \right\}$

Sceptical default unification

- $F \overset{<}{\sqcup}_s G = \sqcap(F \overset{<}{\sqcup}_c G)$
- $\begin{bmatrix}\text{F} & a\end{bmatrix} \overset{<}{\sqcup}_s \begin{bmatrix}\text{F} & \boxed{1}\, b \\ \text{G} & \boxed{1} \\ \text{H} & c\end{bmatrix} = \sqcap\left\{ \begin{bmatrix}\text{F} & a \\ \text{G} & b \\ \text{H} & c\end{bmatrix}, \begin{bmatrix}\text{F} & \boxed{1}\, a \\ \text{G} & \boxed{1} \\ \text{H} & c\end{bmatrix} \right\} = \begin{bmatrix}\text{F} & a \\ \text{G} & \bot \\ \text{H} & c\end{bmatrix}$

Desirable properties of default unification

- Always well-defined
- All strict information is preserved
- If F and G are consistent, it should give the same result as strict unification
- It is finite

Part IV

References

References

(Abney, 1997) Steven P. Abney. Stochastic attribute-value grammars. *Computational Linguistics*, 23(4):597–618, December 1997.

(Bouma et al., 2001) Gosse Bouma, Gertjan van Noord, and Robert Malouf. Alpino: Wide coverage computational analysis of Dutch. In Walter Daelemans, Khalil Sima'an, Jorn Veenstra, and Jakub Zavrel, editors, *Computational Linguistics in the Netherlands 2000. Selected Papers from the Eleventh CLIN Meeting*, number 37 in Language and Computers: Studies in Practical Linguistics, pages 45–59, Amsterdam/New York, NY, 2001. Selection of the papers presented at CLIN '00 in Tilburg.

(Carpenter, 1992) Bob Carpenter. *The logic of typed feature structures: With applications to unification grammars, logic programs and constraint resolution*. Number 32 in Cambridge Tracts in Computer Science. Cambridge–New York–Melbourne, 1992.

(Carpenter, 1993) Bob Carpenter. Skeptical and credulous default unification with application to templates and inheritance. In Ted Briscoe, Ann Copestake, and Valeria de Paiva, editors, *Inheritance, defaults and the lexicon*, Studies in Natural Language Processing, pages 13–37. Cambridge, 1993.

References

(Davey and Priestley, 2002) B. A. Davey and H. A. Priestley. *Introduction to lattices and order*. Cambridge, second edition, 2002.

(Lascarides et al., 1996) Alex Lascarides, Ted Briscoe, Nicholas Asher, and Ann Copestake. Order independent and persistent typed default unification. *Linguistics and Philosophy*, 19(1):1–90, February 1996. `http://www.cl.cam.ac.uk/Research/NL/acquilex/papers.html` (23 January 1998). Revised version of ACQUILEX II WP 34 (August 1994/March 1995).

(Shieber, 1986) Stuart M. Shieber. *An introduction to unification-based approaches to grammar*. Number 5 in CSLI Lecture Notes. Stanford, California, January 1986.

(Toutanova et al., 2002) Kristina Toutanova, Christopher D. Manning, Stuart M. Shieber, Dan Flickinger, and Stephan Oepen. Parse disambiguation for a rich HPSG grammar. In *Proceedings of the First Workshop on Treebanks and Linguistic Theories*, pages 253–263, Sozopol, Bulgaria, 20–21 September 2002. `http://www.bultreebank.org/Proceedings.html` (14 October 2003).

References

(Carroll & Oepen, 2005) John Carroll and Stephan Oepen High Efficiency Realization for a Wide-Coverage Unification Grammar. In Natural Language Processing – IJCNLP 2005, pages 165–176. Springer, Berlin, 1993.