Robust Semantic Construction

Michael Schiehlen[*]
Institute for Computational Linguistics, University of Stuttgart,
Azenbergstr. 12, 70174 Stuttgart
mike@adler.ims.uni-stuttgart.de

1 Introduction

Recent years have seen a surge in interest for robust flat analysis, i.e. NLP systems with fairly limited supply of linguistic knowledge but with vast coverage. The paper describes a module that serves as a back-end to such flat analysis methods and transforms their output into full semantic representations as constructed by deep analysis methods. In particular, the module has been designed so as to process input from

- tree banks
- a statistic context-free parser trained on these tree banks
- a finite-state parser
- a traditional feature-structure parser

The semantic representations which the module constructs are so-called Verb*mobil* Interface Terms (VITs) (Bos et al., 1998), (building on Reyle's Underspecified Discourse Representation Structures (1993), see an example in Figure 1). Although in principle other representations could be constructed as well, VITs seem to be a particularly good choice: They can be implemented as sets of constraints so that semantic construction (SC) reduces to collecting the constraints and unifying some variables in these constraints. Furthermore VITs are supported by an abstract data type (Dorna, 2000). Several daunting problems had to be faced in the design of the module.

[*] This work was funded by the German Federal Ministry of Education, Science, Research and Technology (BMBF) in the framework of the Verb*mobil* Project under Grant 01 IV 101 U. Many thanks are due to M. Emele and the colleagues in Verb*mobil*.

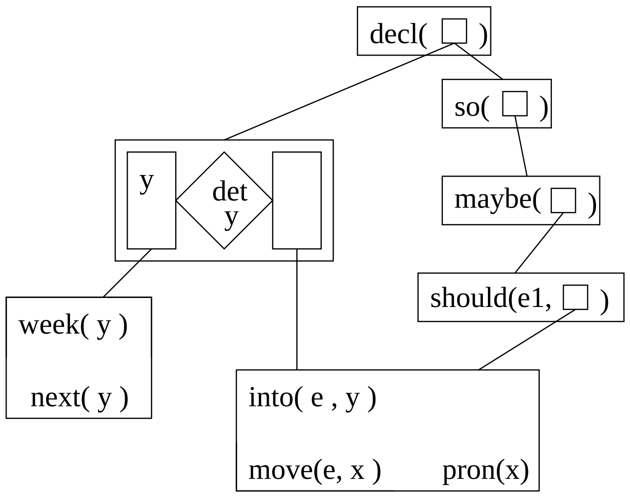

Figure 1: VIT for *So maybe we should move into the next week.*

Context-Free Input. The tree banks providing the input structures (which have been built in the Verb*mobil* project) only encode context-free trees to facilitate the training of a statistical parser. This means that non-local dependencies are either left out (e.g. topicalization in English) or treated by flattening out subtrees into rules (e.g. head-movement in German). The latter strategy can create a vast amount of rules: Since German head-movement connects a clause-initial and a clause-final position, every clause frame gives rise to a new rule. To face this challenge some adjustments had to be made:

(1) Predicate-argument structure is indispensable for SC but presupposes reconstruction of long distance dependencies ("movement"). If syntax cannot supply it, SC has to retrieve it on its own (see Section 5.2).

(2) The sheer bulk of rules prohibits manual tagging of syntactic rules with semantic rules. In-

stead, syntax has to provide pertinent information in its rules so that SC can determine the semantic operations required.

Robustness. Since the tree banks have been constructed by hand, errors are prone to crop up. Likewise, flat analysis methods cannot be expected to deliver input of the same quality as deep traditional parsers. Finally, grammars and semantic formalism will often differ in their subcategorization assumptions: The verb *move* e.g. subcategorizes for *into* in the tree bank (see Figure 2) but not in the VIT formalism (see Figure 3).
To handle this problem, the syntax-semantics-interface should be dismantled as far as possible: Only the most indispensable information should be taken over from syntax. By neglecting all the rest the system stands a good chance of skipping syntax errors. Furthermore in many cases decisions made in syntax need to be overturned in semantics (e.g. the complement/adjunct specifications). Important semantic information is often determined only in SC or in subsequent disambiguation modules that have access to larger stores of context. This approach eases the burden on syntactic analysis and potentially yields more reliable results.

Diverse Input. A SC module should be able to handle input from a variety of grammars and convert it into an independent format of semantic representation. Thus, a common syntax-semantics-interface (or more precisely an interface between syntax and SC) must be defined onto which every type of input is mapped.

2 Design Principles

To cope with the problems mentioned, traditional SC techniques (Montague, 1973) (Pereira and Shieber, 1987) (Bos et al., 1996) cannot be used. Instead, the following ideas were exploited.

Modularity and Underspecification. A major problem in SC is the treatment of ambiguity. Often the local rule context available in SC does not give enough information to resolve such ambiguities. In these cases, underspecification should be used to defer the resolution of choices. Thus, the described module builds a lexically and scopally underspecified representation. Subsequently the lexical ambiguities are resolved by disambiguation modules.

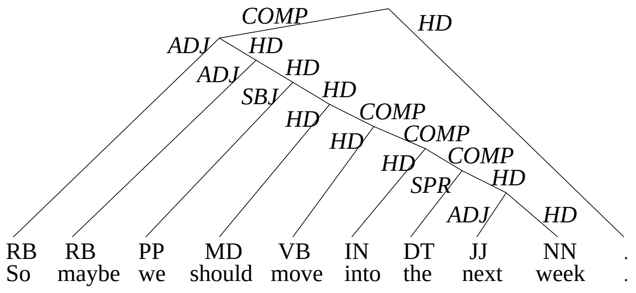

Figure 2: Example for an application tree.

Modularity and Syntax-Semantics-Interface. To facilitate modularity a syntax-semantics-interface is explicitly defined. The input of every parser is mapped onto an interface structure called *application tree*, see Figure 2. In this way input from various sources can be processed with a minimum of effort.

Semantic Database. Great emphasis is laid on an external database of semantic predicates (Heinecke and Worm, 1996). This database associates lemmas with predicate names, semantic classes and subcategorization frames (see the entry in Figure 3).

3 System Overview

The process of SC can be split into two phases (see Figure 4). In the first phase an application tree is traversed and simultaneously an underspecified semantic representation is built (*compositional semantic construction*, see section 5). In the second phase the semantic representation is partially disambiguated (see section 6). The two phases are preceded by a step which mediates between the actual output of the syntax and the syntax-semantics-interface.

4 Syntax-Semantics-Interface

Traditionally, the content of the syntax-semantics-interface is somewhat contentious. While syntax-oriented approaches try to integrate a good part of SC already into the parsing process (cf. the construction of f-structure in LFG), other approaches put the main focus on semantics (e.g. Montague Grammar). To achieve a high degree of flexibility, a modular SC system has to settle for the lowest common denominator of all input sources. The following information seems to be minimally required from the syntax.

Lemma	PredName	SemClass	SyntFrame	Sort	ArgSorts
move	move	v13	arg1:subj,arg3:obj	move_sit	agentive,entity

Figure 3: Entry in the semantic database.

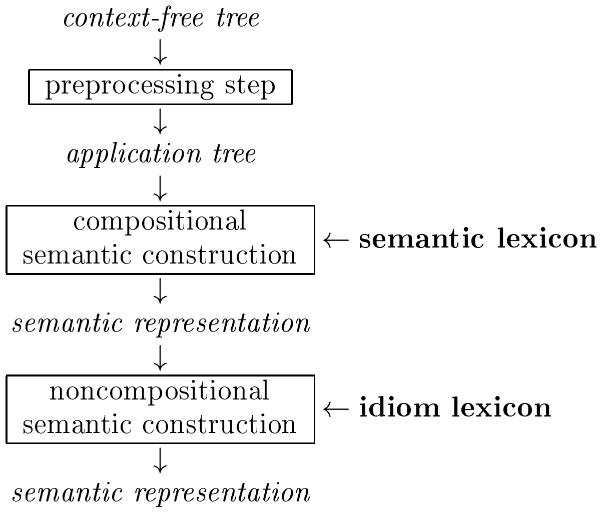

Figure 4: System overview.

(1) The parser should deliver a *tree* for the parsed string which the SC system then can convert into a hierarchical structure of semantic operations (an *application tree*).
(2) Every word in the input string should be syntactically classified, i.e. assigned a *syntactic category* or part of speech tag. We will assume that the parser assigns every word exactly one category. (Lexical underspecification could conceivably be used to deal with multiple categories.) Then morphological analysis (either in syntax or SC) maps the word–category pair to a morphological lemma and a set of morphological features. SC records the features in the VIT while it uses the lemma as a key to the semantic lexicon. In case the lemma is unknown in the semantic lexicon, the system uses the syntactic category to automatically associate a new predicate and semantic class with the lemma.
(3) Every rule used in the tree should specify for each of the categories on its right-hand side exactly one *grammatical role* (GR). If the grammar does not do this, GRs must be determined in the preprocessing step (e.g. determiners in NPs are specifiers). GRs are used to control the choice of semantic operations. The set of GRs employed is inspired by HPSG (Pollard and Sag, 1994): *Head*, *Complement*, *Adjunct*, *Conjunct*, *Specifier*, *Part of a Multi-Word Lexeme*. The corresponding semantic operations are *Complementation*, *Adjunction*, *Coordination*, *Specification*, and *Predicate Formation*. Except for Coordination and Predicate Formation all operations are binary. A rule without a head is considered elliptical and an abstract predicate for the missing head is inserted in semantics.

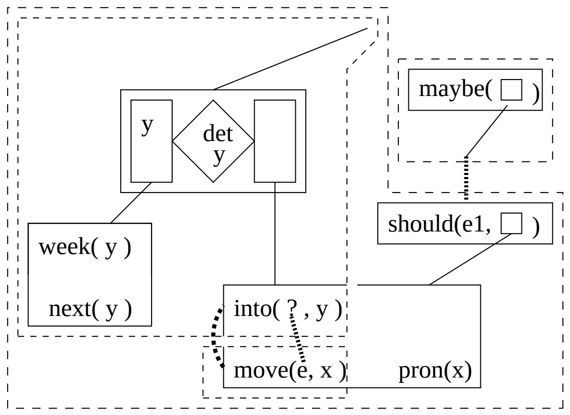

Figure 5: Operation of adjunction.

5 Compositional Semantic Construction Process

Compositional SC follows the application tree (the context-free backbone) and determines the predicate-argument structure (the subcategorization paths).

5.1 Semantic Construction on the Constituent Structure

Figure 5 shows two adjunction operations: In the first one, the intersective adjunct *into the next week* is adjoined to *move*. In the second one, *maybe* is adjoined to the clause. The picture makes clear what the data structure for a partial result should look like: a set of constraints and some pointers to variables in these constraints (e.g. the partial result for *maybe* would be $\{\ \text{maybe}(l_1, h_1), l_2 \leq h_1, l_1 \in l_3\ \}$[1] and $\langle l_2, l_3 \rangle$). Since only finitely many pointers

[1]In a VIT, every predicate is referenced over a base label (e.g. l_1 for maybe). The constraint $l_2 \leq h_1$ says that the box l_2 is subordinated to box h_1, while $l_1 \in l_3$ states that predicate l_1 is in box l_3.

are involved, they can be collected in a record. All partial results are classified into six *semantic types* according to the pointers they allow for: **nhead** (nominal head, for nouns), **vhead** (verbal head, for verbs), **adj** (adjuncts[2], for adverbs, adjectives, subclauses, PPs, also prepositions and subordinating conjunctions), **ncomp** (nominal complements, for pronouns, NPs, also determiners), **vcomp** (verbal complements, for sentences and complement clauses, also sentence moods and complementizers), **cnj** (for coordinating conjunctions).

Semantic operations expect arguments of specific semantic types: Complementation combines heads with complements, Adjunction combines heads with adjuncts. Specification converts an incomplete **ncomp** (i.e. a determiner) and a **nhead** into a complete **ncomp**. Coordination combines a **cnj** with a series of partial results of equal type.

If a type clash occurs, a "type raising" operation is invoked. Such operations usually insert specific abstract predicates that represent phonetically empty words or elided material still to be retrieved by ellipsis resolution in a later step. Figure 6 gives a concise description of these operations[3]. Consider some type-raising examples:

(1) I will be here *Monday*.
udef (**nhead** → **ncomp**)
unspec_ mod (**ncomp** → **adj**)

(2) I will come if *necessary*.
stat (**adj** → **vhead**)

(3) Afternoon might work or *early morning*.
abstr_ rel (**ncomp** → **vhead**)

5.2 Semantic Construction on the Predicate-Argument Structure

While the application tree (Figure 2) states that the pronoun *we* is the subject of *should*, in the

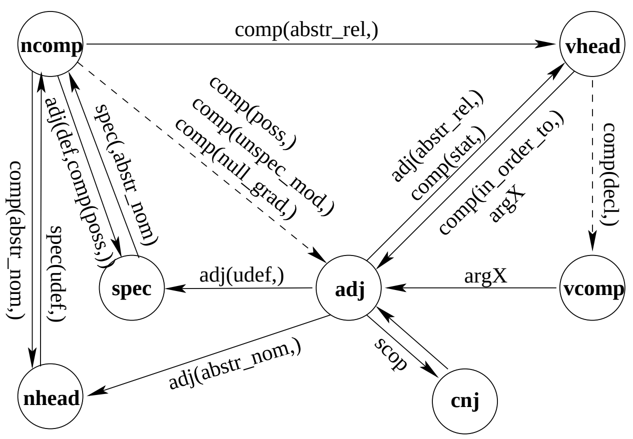

Figure 6: Type-raising operations.

semantic representation (Figure 1) *we* is the subject of *move*. So in this case head and semantic subject are not in the same local rule. To retrace such non-local dependencies, a slash device is used to store the pertinent information (the argument variable and the box label of the head) and propagate it through the application tree in search for a licenser. If a subcategorized element occurs without a subcategorizing head (as occurs often in fragmentary input), an elliptical element is assumed:

(4) I mean if you → *abstr_ rel* with subject *you*

6 Noncompositional Semantic Construction

In noncompositional SC idioms are recognized and a higher level of abstraction is achieved. Technically, noncompositional SC is about transforming VITs. Thus, for implementation the VIT transfer package of Dorna and Emele ((1996)) is used. Linguistically, the component performs the following tasks:

- recognition of multi-word lexemes that are not designated as such by syntax (e.g. greeting expressions *good night*, comparatives *more comfortable*)
- recognition and computation of clock times (e.g. *a quarter to ten*) and date expressions
- recognition of titles (e.g. *Frau Müller*)
- partial disambiguation of sentence mood (e.g. *who did it* is recognized as a question)
- distribution of conjoined material, if required by the level of abstraction aimed for

[2]VITs provide a lexical underspecification class for intersective (e.g. *into the next week* and scopal adjuncts (e.g. *maybe*). Thus, SC has to handle intersective and scopal adjunction in parallel.

[3]In Figure 6 the following names are used for newly inserted predicates: *udef* (null determiner), *unspec_ mod* (null preposition), *stat* (auxiliary verb *be*), *abstr_ nom* (nominal ellipsis), *abstr_ rel* (verbal ellipsis), *decl* (declarative sentence mood), *poss* (relation expressed by genitive), *def* (definite quantifier).

(e.g. clock times *between a quarter to and half past ten*, date expressions *Monday the third and tenth*)

- compositional morphology for German (e.g. *Stiftmuseum* = *museum* with the name *Stift*)

7 Summary

The paper has presented a module[4] capable of handling input from flat analysis methods and transforming them into full-fledged semantic representations. The module works robustly and currently has a throughput of about 98% on Verb*mobil* tree bank input (i.e. it generates 21,222 English and 26,789 German VITs.) The remaining 2% are due to errors in the SC module, errors in the tree bank, or coordination problems between SC and tree bank.

Evaluation of the module is complicated by the effort involved in manually constructing a sizable set of input structures and corresponding semantic representations. Furthermore, the VIT formalism has been in constant flux over the last years with the correct output representations changing almost monthly. It is, however, envisaged to perform an evaluation once dust has settled.

The approach described adds in two respects to the robustness of the overall system. First, the flat analysis parsers used are very robust as concerns low-level inconsistencies such as agreement failures or missing function words (prepositions, determiners, complementizers, etc.). Second, the data analysed in the Verb*mobil* tree banks are exclusively spoken language. Hence, the tree banks encode analyses for phenomena such as fragmentary input, truncated or elliptical sentences, etc. The described module gives semantic analyses for all of these constructions. (Usually an abstract predicate is incorporated which gives a hint to subsequent modules that aim to piece together partial utterances.)

Another perspective of this work is that it provides a first step towards a real corpus semantics by converting large sets of data into semantic representations. Due to the abstraction they embody, semantic representations are a valuable tool for content queries to the processed corpora. More immediately, the semantic representations generated by the described module have been used as test and training data for applications requiring abstract input, such as transfer in machine translation and generation.

[4]More information can be found in Schiehlen (1999).

References

Johan Bos, Björn Gambäck, Christian Lieske, Yoshiki Mori, Manfred Pinkal, and Karsten L. Worm. 1996. Compositional Semantics in Verbmobil. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING '96)*, Copenhagen, Denmark.

Johan Bos, Bianka Buschbeck-Wolf, Michael Dorna, and C.J. Rupp. 1998. Managing information at linguistic interfaces. In *Proceedings of the 17th International Conference on Computational Linguistics (COLING '98)*, Montreal, Canada.

Michael Dorna and Martin C. Emele. 1996. Semantic-Based Transfer. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING '96)*, Copenhagen, Denmark.

Michael Dorna. 2000. A Library Package for the Verbmobil Interface Term. Verb*mobil* Report 238, Institut für maschinelle Sprachverarbeitung.

Johannes Heinecke and Karsten Worm. 1996. A Lexical Semantic Database for Verbmobil. In *Proceedings of the 4th International Conference on Computational Linguistics (COMPLEX '96)*, Budapest, Hungary.

Richard Montague. 1973. The Proper Treatment of Quantification. In Jaako Hintikka, Julius Moravcsik, and Patrick Suppes, editors, *Approaches to Natural Language*, pages 221–242. Reidel, Dordrecht.

Fernando C.N. Pereira and Stuart M. Shieber. 1987. *Prolog and Natural-Language Analysis.* CSLI Lecture Notes. Center for the Study of Language and Information, Stanford, California.

Carl Pollard and Ivan Sag. 1994. *Head-Driven Phrase Structure Grammar.* University of Chicago Press, Chicago.

Uwe Reyle. 1993. Dealing with Ambiguities by Underspecification: Construction, Representation and Deduction. *Journal of Semantics*, 10(2):123–179.

Michael Schiehlen. 1999. *Semantikkonstruktion.* Ph.D. thesis, Universität Stuttgart.