Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research Report
RR-92-55

Natural Language Semantics and Compiler Technology

**John Nerbonne, Joachim Laubsch,
Abdel Kader Diagne, Stephan Oepen**

December 1992

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
D-6750 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, Philips, SEMA Group Systems, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct *systems with technical knowledge and common sense* which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- Intelligent Engineering Systems
- Intelligent User Interfaces
- Intelligent Communication Networks
- Intelligent Cooperative Systems.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Prof. Dr. Gerhard Barth
Director

Natural Language Semantics and Compiler Technology

John Nerbonne, Joachim Laubsch, Abdel Kader Diagne, Stephan Oepen

DFKI-RR-92-55

This work has been supported by a grant from The Federal Ministry for Research and Technology (FKZ ITW-9002 0).

© Deutsches Forschungszentrum für Künstliche Intelligenz 1992

This work may not be copied or reproduced in whole or in part for any commercial purpose. Permission to copy in whole or in part without payment of fee is granted for nonprofit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslautern, Federal Republic of Germany; an acknowledgement of the authors and individual contributors to the work; all applicable portions of this copyright notice. Copying, reproducing, or republishing for any other purpose shall require a licence with payment of fee to Deutsches Forschungszentrum für Künstliche Intelligenz.

Natural Language Semantics and Compiler Technology*

John Nerbonne†, Joachim Laubsch‡, Abdel Kader Diagne†, Stephan Oepen†

†Deutsches Forschungszentrum für Künstliche Intelligenz
Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11, Germany
{nerbonne,diagne,oe}@dfki.uni-sb.de

‡Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94303-0971, USA
laubsch@hplabs.hp.com

Abstract

This paper recommends an approach to the implementation of semantic representation languages (SRLs) which exploits a parallelism between SRLs and programming languages (PLs). The design requirements of SRLs for natural language are similar to those of PLs in their goals. First, in both cases we seek modules in which both the surface representation (print form) and the underlying data structures are important. This requirement highlights the need for general tools allowing the printing and reading of expressions (data structures). Second, these modules need to cooperate with foreign modules, so that the importance of interface technology (compilation) is paramount; and third, both compilers and semantic modules need "inferential" facilities for transforming (simplifying) complex expressions in order to ease subsequent processing.

But the most important parallel is the need in both fields for tools which are useful in combination with a variety of concrete languages—general purpose parsers, printers, simplifiers (transformation facilities) and compilers. This arises in PL technology from (among other things) the need for experimentation in language design, which is again parallel to the case of SRLs.

Using a compiler-based approach, we have implemented $\mathcal{NLL}$, a public domain software package for computational natural language semantics. Several interfaces exist both for grammar modules and for applications, using a variety of interface technologies, including especially compilation. We review here a variety of $\mathcal{NLL}$ applications, focusing on COSMA, an NL interface to a distributed appointment manager.

Keywords: Computational Linguistics, Semantics, Compiler, Interfaces

*Thanks to Derek Proudian for initial stimulus in the direction of compiler technology and to Masayo Iida, Walter Kasper, Karsten Konrad and Ingo Neis for important experimentation. This work was supported by research grants ITW 9002 0 and ITV 9102 from the German Bundesministerium für Forschung und Technologie to the DFKI DISCO and ASL projects.

Natural Language Semantics and Compiler Technology

Contents

1 Introduction: Design Goals 2

2 Analogy to Programming Languages 3

3 $\mathcal{NLL}$ 5

3.1 Data Structures and Basic Tools 5

3.2 Interfaces and Compilations 6

3.3 Inference Rules 7

3.4 Compiling into $\mathcal{NLL}$ 9

3.5 $\mathcal{NLL}$-Planner Interface in COSMA 9

3.5.1 Initial Sentence Semantics ($\mathcal{NLL}$ Representation of Parsing Result) 10

3.5.2 Equivalence Transformations and Simplification 10

3.5.3 Translation into Target Language 12

4 Conclusions and Prospectus 13

4.1 Additional Benefits—Comparing SRLs Theoretically 13

4.2 Future Directions 14

References 16

1 Introduction: Design Goals

The focus of this paper is the design and IMPLEMENTATION of semantic representation languages (SRLs). Given the need of such modules to represent natural language meanings, we assume that they should apply linguistic semantics, the specialized study of natural language meaning. But because of the focus on design and implementation, we examine quite generally the uses to which such modules may be put, abstracting away from details which distinguish such superficially distinct approaches as generalized quantifier theory (GQT, Barwise and Cooper 1981), discourse representation theory (DRT, Kamp 1981), situation theory (Barwise and Perry 1983), or dynamic logic (Groenendijk and Stokhof 1991).

The appropriate design for any module can only be determined by close analysis of the uses to which it is to be put. It is therefore appropriate to begin our consideration of the design for an NL semantics module with a summary of its foreseen applications. We see the areas of application for semantics modules as divided into three main types. First, and most importantly, there are NL understanding and generation applications—where meaning representation and manipulation is central (Allen 1987). These include NL interfaces, such as database query, software systems interfaces, speech understanding, and device interfaces; information retrieval; message understanding (e.g., of bug reports); and automatic indexing. Second, there is an reciprocal application to linguistics—that of implementing semantic theories in order to enhance one's understanding of them (cf. e.g., Scha 1981). Third and finally, there are applications in which semantics is used primarily for the sake of adding additional constraints to a classification task such as speech recognition or grammar checking (Young et al. 1989). (We have included this category for completeness, but will not try to develop it further since it is still unclear what sorts of semantics information are most successful in constraining recognition, so that it is difficult to extract useful design criteria here.)

The first and numerically most important group of applications, that of understanding and generation, give rise to fairly clear requirements. Independent semantics modules are used with these applications for the following reasons:

meaning representation The SRL must recognize all (or as many as possible) of the semantic distinctions relevant to the natural language used in an application domain. This is the area in which linguistic semantics is essential. Because linguistic semantics is an lively research field, SRL are likely to change frequently, which underscores the need for general specification tools.

extend domain-independent NLP A semantics module decreases the "tailoring" required to fit an NLP system to an application domain, and increases the portability of the NLP system (Martin et al. 1983).

ease grammar–application mapping The use of an SRL simplifies and standardizes the mapping m: grammar $\Leftrightarrow$ domain; in particular it limits the application mapping to m: SRL $\Leftrightarrow$ domain, and the SRL is much less variable than NL syntax. This involves two interfaces: grammar $\Leftrightarrow$ SRL $\Leftrightarrow$ domain.

support inference (not in applications) In general applications have no inferential capability, but databases and some other software systems are exceptions. Even these, however, normally do not support, e.g., plural distributivity inferences (Scha and Stallard 1988), or sensitivity to presupposition vs. assertion (Weischedel 1979), which are implicitly assumed by human language users. It is the need to provide for inference which provides the most convincing argument for using logics as SRLs; these characterize the allowable inferences. (But the use of logic jibes nicely with the application of linguistic semantics, which is also logic-based.

support meaning-related processing In addition to inference, semantic modules generally provide the data structures on which resolution and disambiguation are based. But this may involve communication, and therefore interfaces with various modules:

- context or discourse memory module (anaphoric resolution, reference resolution);
- domain model (predicate disambiguation); and
- dialogue management (speech act recognition).

We note the importance of interfaces in this catalogue and the special need for inference. Applications involving the implementation of linguistic semantics theories share with applications in NLU and generation the need to provide GENERAL TOOLS for work with SRLs—language definition tools, readers (parsers), printers (unparsers), and inference specification tools.

We arrive at the following summary of design goals for work on semantics modules:

- semantic representation
- inference
- support meaning-related processing—disambiguation, resolution, speech act management
- modularity—independence from syntax and application
- modifiability—for experimentation
- interface tools—for mapping into and out of module
- tools for independent use (reader, printer, tracer)

The first three points confirm the good sense of current practice in the field—that of viewing the main task of the semantic module as the implementation of a linguistic semantic theory (with selected AI enhancements for resolution and disambiguation). But we suggest that insufficient attention is paid to the latter four points, and that we may profit from a comparison to programming language technology and compiler construction.

2 Analogy to Programming Languages

It is axiomatic that modern PLs should meet the last four goals listed in the design goals for SRLs. Standard introductions (Aho et al. 1986) detail how a programming language syntax is specified in definitions independent of specific machines and environments (modularity) which are, moreover, easily modifiable given tools for parsing (parser-generators) and printing. The parsers automatically created from language specifications take well-formed strings as input and produce abstract syntax trees (like linguistic parse trees) as output. Modern tools also provide

Goals	PL	SRL
modularity	independent definition (BNF)	
tools	parser, printer	parser (%), printer
modifiability	parser-generator	
mappings in, out	compiler	
inference	program transformation	resolution, backward-chaining ...

Figure 1: Design goals common to PLs and SRLs plotted against "standard" solutions in the two areas. The analogy suggests filling the gaps for standard solutions for SRLs by using PL solutions: language specification tools for definition together with parser-generators to provide the SRL reader, and compiler technology for interfaces to semantics modules. Finally, program transformation techniques suggest a simple implementation for at least some inference rules.

printers (unparsers) which reverse the process: given an abstract syntax tree, they produce a print form (Friedman et al. 1992, 85ff).

Just as a modular SRL must interface to more than one application, a programming language needs to be able to run on different machines. In the latter case, this is accomplished by compiling: the abstract syntax trees produced by the PL's parser are transformed into (the abstract syntax trees representing) expressions of another lower-level language (often a machine-specific assembler language). While it is obvious how this scheme enables generality vis-à-vis translation targets, it may not be as immediately apparent that the level of abstract syntax likewise facilitates generality toward translations sources: in the case of a PL such as C, we not only compile FROM C into various assembler languages on the basis of transformations of abstract syntax, but we likewise compile other (normally more specialized languages) INTO C. The SRL correspondence is the use of compiler technology to translate into SRLs, viz., in syntax/semantics interfaces (in NLU) or application/semantics interfaces (in generation). We provide details of practical experiments on both interfaces below.

Some of the transformations performed by compilers are not simple translations into target languages, but rather transformations to alternative structures in the source language (cf. Aho et al. 1986, 592ff), or immediate evaluation of parse structures (cf. "translation during parsing" Aho et al. 1986, 293-301). The use of these techniques suggests an implementation for some inference facilities for SRLs—those arising from equivalence rules.

A more ambitious, but still relevant example of facilitating modularity and generality in everyday compiler technology is the *register transfer language* (RTL) incorporated in the GNU C and C++ compilers. RTL is an abstraction both from the concrete syntax of several high-level PLs (in addition to the existing C and C++ compilers, Ada, Fortran, Pascal and Modula-2 compilers are under development) and from several concrete assembly languages. The high-level languages are all compiled into RTL and make use of the *same* highly optimizing compiler engine. At the same time GNU compilers have been ported to a large variety of computer architectures relatively easily because it is only the *final* compiler pass that translates into assembly language on the basis of machine description files which declaratively specify the mapping from RTL to the concrete architecture.

Richard M. Stallman describes RTL in the following way in the current gcc manual (Stallman 1992):

> Most of the work of the compiler is done on an intermediate representation called register transfer language. In this language, the instructions to be output are described, pretty much one by one, in an algebraic form that describes what the instruction does.

> RTL is inspired by Lisp lists. It has both an internal form, made up of structures that point at other structures, and a textual form that is used in the machine description and in printed debugging dumps. The textual form uses nested parentheses to indicate the pointers in the internal form.
>
> [Chapter 12: RTL Representation. p. 145]

The SRL analogue is of course deployment in a multi-application scenario, where a single NL system is used to interface to several application systems.[1] A technical prerequisite for any such use of NLP technology is the ability to translate to multiple application languages, enhanced by the translation flexibility compilation offers.

Our core thesis: PL technology may profitably be applied to the design of SRL software. Figure 1 summarizes the points at which immediate borrowings from PL technology seem apt means to SRL goals.

We turn now to a brief description of $\mathcal{NLL}$, an SRL implemented using PL technology. We then illustrate how SRLs profit from this approach using a concrete and fully implemented example.

3 $\mathcal{NLL}$

$\mathcal{NLL}$ is an SRL which borrows heavily from linguistic semantics in order to provide representational adequacy, using, e.g., on the one hand work from generalized quantifier theory and on the other from the logic of plurals. Laubsch and Nerbonne 1991 and Nerbonne 1992 present an overview of $\mathcal{NLL}$ and the background linguistic and model-theoretic ideas, which will not be repeated here. For the sake of understanding examples below, we note that atomic formulas in $\mathcal{NLL}$ are composed of a predicate together with a set of role-argument pairs, e.g.

'Anterist ships to Hamburg' ship(source:a goal:h)

The $\mathcal{NLL}$ formula may also be read: '*a*' plays the role of agent and '*h*' that of goal with respect to some shipping situation. An advantage of identifying arguments via roles rather than positions (as is customary in predicate logic) is that one can sensibly use the same predicate, e.g., 'ship', with various numbers of arguments; thus even though something must also play a 'theme' role in this situation (what is shipped), it need not be expressed in the role-coded set of arguments. Note that this means that unbound variables may be dropped from atomic formulas where they occur only once: there is an implicit existential force attached to unexpressed arguments. Cf. Nerbonne 1992 for formal development.

3.1 Data Structures and Basic Tools

Following the PL lead, we begin with a formal syntactic specification of $\mathcal{NLL}$ in a form usable by a parser-generator.[2] For this purpose we use Zebu (Laubsch 1992b), a public-domain tool in Common Lisp.[3]

Zebu grammar specifications are illustrated with an excerpt from the $\mathcal{NLL}$ grammar in Figure 2. A grammar is primarily a set of RULES, each of which specifies a SYNTAX for a grammatical category and an ACTION to be taken by the parser when the category is found. These specifications are easily modified in case extensions, variations or even substantial modifications of the language become interesting. In addition to syntax rules, Zebu grammars may also contain lexical restrictions (Laubsch 1992b,15) needed for generating a lexical analyzer (which, however, is not used in $\mathcal{NLL}$).

From the $\mathcal{NLL}$ grammar, Zebu generates an LALR(1) parser (Aho et al. 1986,§ 4), which is the $\mathcal{NLL}$ reader. The reader immediately supports experiments with the semantics module by easing the creation of semantic data structures. The Zebu grammar compilation process detects any inconsistencies or ambiguities in the grammar definition.

Zebu goes beyond the capabilities of parser-generators such as UNIX yacc in further optionally generating (automatically) the definition of a DOMAIN, a hierarchy of data structures (LISP

[1] See Bobrow et al. 1990 for an example of this sort of system.

[2] We are concentrating here on the more recent $\mathcal{NLL}$ implementation; an earlier implementation in REFINE (Laubsch and Nerbonne 1991) is no longer the focus of our efforts, even though we continue to maintain it for its usefulness in rapid prototyping. REFINE is a trademark of Reasoning Systems, Palo Alto.

[3] Zebu was originally developed in Scheme by William Wells.

```
(defrule Atomic-Wff
  := (NLL-Predicate "(" Role-Argument-Pairs ")" )
  :build (make-Atomic-Wff :-predicate NLL-Predicate
                          :-Role-Argument-Pairs Role-Argument-Pairs))

(defrule Role-Argument-Pairs
  := ()
  :build zb:EMPTY-SEQ
  := (Role-Argument-Pair Role-Argument-Pairs)
  :build (CONS Role-Argument-Pair Role-Argument-Pairs))

(defrule Role-Argument-Pair
  := (Role-Complex ":" NLL-Term)
  :build (make-Role-Argument-Pair :-Role     Role-Complex
                                  :-Argument NLL-Term))
```

Figure 2: An excerpt of the $\mathcal{NLL}$ definition in Zebu. Each LISP-like rule contains a BNF-like syntax part, introduced by ":=", and an action, introduced by ":build". Whenever the Zebu parser finds an instance of the syntax, it takes the action specified in the rule. In these cases this is always the action of constructing an abstract syntax tree for the expression. A complete BNF for $\mathcal{NLL}$ may be found in Laubsch and Nerbonne, 1991.

structures) for abstract syntax. If this option is chosen, then Zebu defines a structure type for each expression type; the structure for a given expression (e.g., Atomic-Wff—cf. Figure 2) has as many fields as the expression has subexpressions (e.g., Predicate and Role-Argument-Pairs). On the basis of the domain Zebu then also generates an "unparser", in this case the $\mathcal{NLL}$ printer (which in turn may be called by the LISP printer).

At this point we have implemented a representational system with thorough dual-access: we may process it through manipulations of either surface or abstract syntax. For example, $\mathcal{NLL}$ structures are created either from strings or from constructor functions (and occasionally in a mixture of approaches); similarly, one could specify lexical inference rules or substitution rules either as string operations or as operations on LISP structures (or both). Important processing submodules have been implemented using both surface and abstract levels. We examine these now.

3.2 Interfaces and Compilations

As we noted in Section 1, an important task of SRLs is the communication with a variety of modules in NLP systems, including at least syntax, context (resolution), dialogue management, and application system. How can the PL approach to SRL design support the construction of interfaces?

The dual access provided by the PL approach already allows an interesting degree of freedom. For example, the opportunity to create $\mathcal{NLL}$ via constructor functions allows the implementation of a syntax-semantics interface of the sort suggested by Johnson and Kay 1990—in which the syntax/semantics interface is constituted by a set of generic constructor functions attached to syntactic rules (and therefore nonterminal nodes). $\mathcal{NLL}$ has been employed this way in a syntax/semantics interface in an extensive NLP system (Nerbonne and Proudian 1987). This is appropriate when relatively complex structures are created in a series of simple increments. Alternatively, one may invoke the $\mathcal{NLL}$ reader to create $\mathcal{NLL}$, and an interface from the COSMA appointment manager (cf. below) to $\mathcal{NLL}$ (for generation) invokes the reader extensively. This made the single-step creation of complex structures much simpler.

But given the relatively easy access to abstract syntax trees provided by the Zebu reader, the construction of interfaces through genuine compilation (translation of expressions based on abstract syntax) is also feasible. $\mathcal{NLL}$'s basic scheme of compilation is TREE REWRITING (Aho et al. 1986,572ff). An abstract syntax tree is traversed (different traversal disciplines may be

specified), and at each node, each of a sequence of REWRITE RULES is applied. A rewrite rule abstractly takes the form:[4]

meta-syntactic-pattern $\Rightarrow$ *replacement-node*

A rewrite rule checks whether a *meta-syntactic-pattern* is satisfied at the current node, and returns the replacement node together with a boolean flag indicating whether the rule has fired: (*replacement-node*,δ?:bool). In case we are translating from one abstract syntax to another, then the meta-syntactic-pattern describes a node in the source language, while the replacement node belongs to the translation target language. The traversal routine replaces the current node with the replacement-node in case the rule has fired.

The top-down tree-rewriting algorithm PREORDER-TRANSFORM inputs a tree t and a sequence $\langle r_1 \ldots r_n \rangle$ of rewrite rules. It then traverses the tree in preorder (Aho et al. 1983,78-9), and at each node, attempts to rewrite using each of the rules r_i. If any rule in the sequence fires, then the entire sequence is tried again, until no rules fire. Then the traversal continues, until the leaf nodes of the tree. The algorithm is attractive because it reduces the tree-transformation problem to the specification of transformations on local subtrees. We specify the algorithm here in pseudo-code:

```
procedure preorder-transform (t:tree, ⟨r_1 ... r_n⟩:rules);
        begin
            Δ? ← 1;
            do while Δ? = 1
                Δ? ← 0;
                for each r_i from i = 1 to i = n do
                    if r_i(t) = (t', 1) then
                        begin
                            t ← t';
                            Δ? ← 1;
                        end;
                for each child c of t do
                    preorder-transform(c, ⟨r_1 ... r_n⟩)
        end preorder-transform
```

An analogous POSTORDER-TRANSFORM invokes sequences of rewrite rules in a bottom-up traversal of the abstract syntax tree.

In addition to optimized control routines for tree-transformation $\mathcal{NLL}$ provides a library of access and manipulation functions (for substitution, construction, simplification) to support the transformation process. Laubsch 1992a reports on the required transformations for $\mathcal{NLL}$ compilers to SQL and to the New Wave task language, and we present below several transformations needed in COSMA, a distributed system for appointment management.

Cf. Figure 3 for a further example of a compilation from $\mathcal{NLL}$, this time to a feature description language.

Compilation is normally an effective translation technique because it abstracts away from irrelevant details of the concrete syntaxes of target and source languages. It is especially appropriate: (i) when communication between modules is limited (e.g., when modules run on separate machines or in separate processes, so that communication is limited to strings); (ii) when the nature of target data structures is unknown or unspecified; or (iii) when there is minor variability in targets (e.g., different versions of the same programming language or query language).

3.3 Inference Rules

In Subsection 3.2 above, we examined the use of rewrite rules of the form:

meta-syntactic-pattern $\Rightarrow$ *replacement-node*

[4] In the REFINE implementation of $\mathcal{NLL}$, rules are also concretely of this form (cf. Rea 1990,§§ 3.7.5–3.7.7). In current work we are trying to incorporate abstract specifications of $\mathcal{NLL}$ rewrite rules in the Zebu-based version. In the current implementation, rewrite rules are LISP functions.

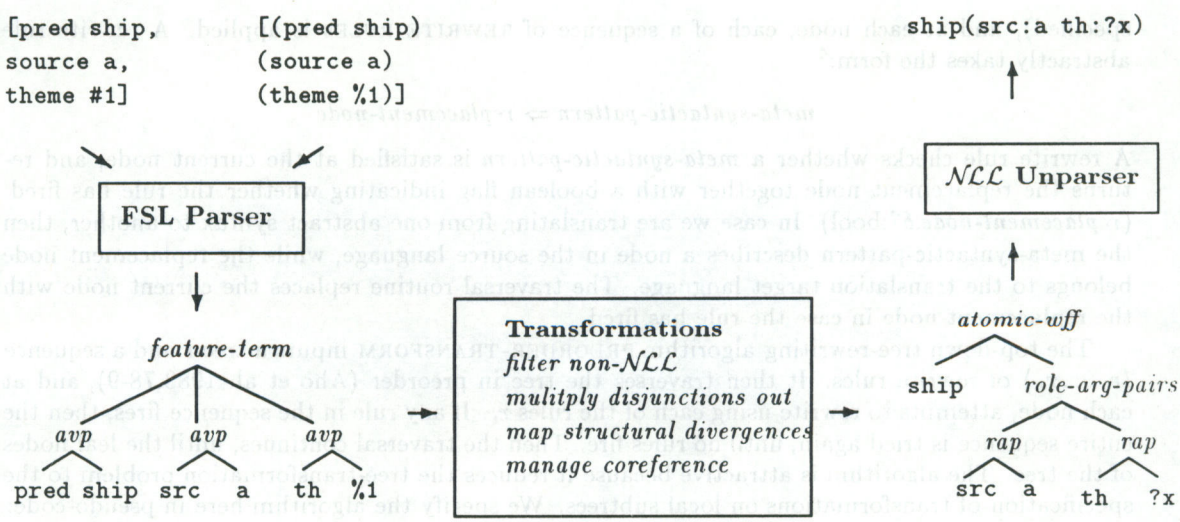


Figure 3: Compiler-based approach to the problem of semantics modularity. In order to employ $\mathcal{NLL}$ not only in the ASL speech understanding project (cf. Görz 1992) but also in the COSMA dialogue application (cf. below), involving both different parsers and different feature description languages (ASL and UDiNe), FS2NLL, a compiler from feature structure (fs) languages to $\mathcal{NLL}$ was developed. The two fs languages were parsed into the same abstract syntax, which was filtered and transformed into $\mathcal{NLL}$ abstract syntax, which is "unparsed" in a final step. A further borrowing from compiler technology, a symbol table, was needed to manage coreferences in the fs language ('`%1,#1`'). Diagne and Nerbonne 1992 report on the ASL interface. A final advantage of the compiler-based approach is the fact that several components of the translation are useful for reverse translation. NLL2FS was implemented to provide semantic feedback to parsing and search in the ASL speech understanding system and for generation in the COSMA dialogue system. NLL2FS has just completed implementation.

where the meta-syntactic-pattern describes a node in the source language, while the replacement node belongs to the translation target language. This is appropriate for compiling one language into another, but it is not forced by the technology. In particular, we may formulate rewrite rules where both the metasyntactic pattern and replacement node refer to $\mathcal{NLL}$ structures, and this is exactly what we have done in formulating some inference rules for $\mathcal{NLL}$.

For example n-ary conjunctions are normalized according to the following patterns.

flatten	AND{p AND{q r}} → {p q r}
identity element	AND{p true} → AND{p}
constant	AND{p false} → false
single element	AND{p} → p

Of course, as we indicated in Section 3.2 above, these are not specified declaratively in the Zebu implementation, but instead implemented as LISP functions. A declarative (and easier to use) specification is the subject of ongoing work. Laubsch December 1989 presents many more inferences rules as these were used in a database interface from $\mathcal{NLL}$. There is no further inference mechanism in $\mathcal{NLL}$, so that the inference system is weak. Eisinger et al. 1992 sketches more powerful rewrite systems which proceed from the same basis. This may be a direction for future work.

We turn now to illustrations of compiling techniques applied to concrete interface and inference problems.

3.4 Compiling into $\mathcal{NLL}$

The further examples of compilation below all involve compiling FROM $\mathcal{NLL}$ into another language; but it is natural to apply the same techniques when translating INTO $\mathcal{NLL}$, and similar advantages accrue to these applications as well. This section reports briefly on a syntax/semantics interface based on the compiler approach.

As noted above $\mathcal{NLL}$ supports a variety of creation methods: besides invoking constructor functions, one may also invoke a reader, and there is a specialized compiler for interfacing to feature-based grammars.

Compilation is particularly cost-effective when one wishes to construct interfaces to several modules with similar representations. In this case relatively minor modifications in grammar specifications may be all that is required to obtain further interfaces. Cf. Figure 3 for an example in which two feature description languages were compiled into $\mathcal{NLL}$. These were used with different front ends (different grammars, lexicons, parsers) in different systems with no application-specific modification.

We turn to a recent application of these compiling techniques in COSMA, both of the inferential and of the translating types.

3.5 $\mathcal{NLL}$-Planner Interface in COSMA

In building the COSMA (Cooperative Schedule Management Agent) system we connected a computational linguistics core engine for natural language understanding and generation—the DISCO system developed at DFKI—to an appointment planner that was provided by another in-house project. The product of our collaboration is a distributed system in which each participant in the appointment planning process may or may not use (an instance of) COSMA; the practical *raison d'être* of NL here is that it enables COSMA users to automate their appointment management even when users are involved whose appointment management is not done automatically—for these users an NL channel is maintained. For further details see Neumann et al. 1993.

When interfacing the DISCO meaning processing modules (mainly the $\mathcal{NLL}$ semantics module with basic resolution and speech act recognition built on top of it) to the planner internal representation language (henceforth IR), the compiler approach described above has been proven to be not only powerful enough to support rather complex semantics processing and translation into and out of the interface language, but at the same time flexible enough to allow for rapid prototyping in a development environment in which the target language has undergone syntactic change several times.

The $\mathcal{NLL}$ transformations implemented so far in the COSMA prototype can be classified according to theory vs. domain dependence and target language ($\mathcal{NLL}$ vs. IR) as follows:

- **core $\mathcal{NLL}$ inference**: purely logical (theory-independent) equivalence transformation, e.g. flattening of nested conjunctions (example below)
- **simplification**: equivalence transformation dependent on extralogical axioms and generating a 'canonical' form, e.g., grouping of multiple restrictions on the same variable (example below)
- **disambiguation**: resolution of ambiguity, e.g. quantifier scoping
- **domain-specific inference**: transformation based on domain knowledge, e.g. anchoring of underspecified time expressions, non-conventional speech act processing
- **translation**: mapping into *different* abstract syntax — mediation between different expressive power on either side, e.g. translation to SQL or the COSMA planner internal representation (example below)

Based on a concrete example from the appointment domain, viz. the sentence

(1) Ich möchte mit Ihnen einen Termin für den 23. Oktober um 13:30 h im DFKI vereinbaren.
(*I want to arrange a meeting with you for October 23 at 1:30 PM in the DFKI.*)

the following sections will present in some detail the major transformation steps that are taken

```
request(theme:^vereinbar(agent:(?a | and{has-card-leq(theme:?a goal:1)
                                         speak(source:?a)})
                         goal:(?p | and{has-card-gt(theme:?p goal:1)
                                        speak(goal:?p)})
                         instance:?e
                         theme:(?x |
                                 and{and{and{and{has-card-leq(theme:?x goal:1)
                                                 termin(instance:?x)}
                                             temp-um(theme:?x
                                                     goal:(?t2 | time(instance:?t2
                                                                      hour:13
                                                                      min:30))))}
                                         temp-fuer(theme:?x
                                                   goal:(?t1 | time(instance:?t1
                                                                    mon:(?m |
                                                                         oktober(instance:?m))
                                                                    day:23))))}
                                     space-in(theme:?x
                                              goal:(?y | identity(source:?y theme:'DFKI')))))))
```

Figure 4: Result of parsing the sentence 'Ich möchte mit Ihnen einen Termin für den 23. Oktober um 13:30 h im DFKI vereinbaren.' after simple translation into $\mathcal{NLL}$. The nesting and order of conjuncts in the restriction of the variable ?x is semantically arbitrary, reflecting a (convenient) grammatical treatment in which adjunct semantics are introduced conjunctively into conditions on semantic arguments.

in mapping the initial sentence semantics into an appropriate IR expression.[5] Because the disambiguation and domain-specific inference components are still under development in DISCO (rudimentary parts have been implemented using the $\mathcal{NLL}$ compiler approach), we will focus on the other three areas, core inference, simplification, and translation.

3.5.1 Initial Sentence Semantics ($\mathcal{NLL}$ Representation of Parsing Result)

The initial semantics as input to the semantics module in feature structures is heavily determined by purely syntactic conditions, i.e. the structure derived from the underlying grammar. Because sentence 3.5 is very rich in prepositional phrases (free adjuncts in the HPSG-style DISCO grammar), the attachment problem becomes obvious in the grouping of restrictions to the restricted parameter ?x, the theme of the predicate vereinbar() (cf. Figure 4). Basically the expression given in Figure 4 is an *atomic formula* representing a request() (a primitive from the DISCO syntactic speech act recognition module which is integrated into the feature structure formalism) whose theme is vereinbar() (arrange). The agent role ?a of the vereinbar() event ?e is filled by a *restricted parameter* that is constrained to be singular and the source of the speak() predicate, i.e. the speaker (*I*) — likewise the goal of vereinbar() is restricted to the plural hearers (the goal role of the speak() predicate, i.e. *you*). The main part of the formula restricts the theme role of what is to be arranged: the variable ?x is constrained to be a single appointment (termin()) and to stand in various temporal and spatial relations to expressions originating from the prepositional phrases *for October 23, at 1:30 PM* and *in the DFKI*. The grouping of conjuncts in the restricting formula is determined by the nesting of adjuncts in the syntactical structure of the sentence.

3.5.2 Equivalence Transformations and Simplification

This section illustrates the application of inference rules which are purely logical in nature (i.e., independent of the underlying semantic theory and the actual application domain) to the parse semantics shown in Figure 4. We have already seen one example of an applicable logical equivalence

[5] We have also implemented the reverse translation, from the planner IR into $\mathcal{NLL}$, but it the result has yet to be integrated into the verbalization part of generation, so that it is untested.

transformation, viz., the 'flattening' of nested conjunctions (see § 3.3 above). Other relevant rewrite rules are the following:

- **elimination of non-constraining (unbound) variables:**

  $p(\ldots \text{role}_i\text{:?y} \ldots) \rightarrow p(\ldots)$ (p is an $\mathcal{NLL}$ predicate; ?y does not ocur in '...')

- **identity substitution:**

  (?x | identity(arg1:?x arg2:c)) $\rightarrow c$ (c is an $\mathcal{NLL}$ constant)

Applied to the $\mathcal{NLL}$ expression in Figure 4 the variable ?e (the arrange event) will be removed from the formula because it is a single unbound occurrence (cf. remarks on anadic predicates above). One might ask why the variable was ever introduced, if it is only to be eliminated. But the introduction of an event variable ?e is well motivated in the grammatical system: it serves as a peg on which to hang the restrictions imposed, e.g., by free temporal adjuncts. Likewise for grammatical reasons the goal role of the spatial restriction to the appointment variable ?x is itself a restricted parameter requiring ?y to stand in the identity() relation to the $\mathcal{NLL}$ constant 'DFKI'. The variable ?y might have been used to allow further restriction (e.g., from appositive adjuncts) during the process of compositional semantic construction. The *identity substitution* rewrite rule as stated above will fire for the current example and substitute the constant 'DFKI' for occurrences of the variable ?y.

While the core logical rewrite rules come built-in with the $\mathcal{NLL}$ semantics module (viewed as a base representational formalism), the second class of transformations listed above, viz., the simplification to a 'canonical' form, depends on the concrete linguistic theory of semantics assumed. With respect to example 3.5 we will exclusively look at the representation of temporal constraints in the DISCO framework.

The basic format of temporal expressions in the DISCO grammar is given by the atomic predicate time() with roles instance, year, month, day etc. down to second—additional roles for units of time could be easily added. The relation is intended to hold whenever the time playing the instance role falls within the time specifications provided by the various other role-argument pairs. However, as Figure 4 shows, the actual distribution of temporal data from the example sentence is determined by the number of prepositional phrases and the syntactical nesting of adjuncts. Moreover, the two chunks of temporal restrictions on the appointment stand in the scope of two distinct $\mathcal{NLL}$ predicates (viz. temp-fuer() and temp-um())[6] as they are lexically introduced by the corresponding German prepositions.

Two very similar examples from the DISCO grammar yield structurally quite different results (where the italicized German phrases in the formulas abbreviate further argument specifications—for fuller specifications, cf. Figure 4):

- 'am Freitag um 13:30 h' (*on Friday at 1:30 PM*)

```
temp-am(theme:?e
        goal:(?t1 | and{time(instance:?t1 'Freitag')
                        temp-um(theme:?t1 goal:(?t2 | time(instance:?t2 '13:30 h')))
```

- 'am Nachmittag des 23.10.' (*on the afternoon of October 23* [genitive case])

```
temp-am(theme:?e
        goal:(?t1 | and{time(instance:?t1 'Nachmittag')
                        poss(possessed:?t1 possessor:(?t2 | time(instance:?t2 '23.10.')))
```

In the first case the second prepositional phrase is (syntactically) subordinate to the noun 'Freitag' which itself is the complement to the first preposition 'am'. The $\mathcal{NLL}$ expression derived clearly reflects this surface structure and, hence, gives rise to the interpretation that *Friday* is (temporally) restricted to 1:30 PM. Almost analogously the second example exhibits the semantic treatment of genitive attributes in the DISCO grammar thus making *October 23* be the possessor of *afternoon*.

[6] The temp- prefix results from sortal disambiguation based on the type of complements (both temporal in this case) as it is integrated into the feature structure (meta-) representation of $\mathcal{NLL}$ expressions and evaluated at parse time. Cf. Kasper 1992.

```
request(theme:^vereinbar(theme:(?x | and{has-card-leq(theme:?x goal:1)
                                        termin(instance:?x)
                                        temp-in(theme:?x
                                                goal:(?t | time(instance:?t
                                                                mon:(?m | oktober(instance:?m))
                                                                day:23
                                                                hour:13
                                                                min:30)))
                                        space-in(theme:?x
                                                 goal:'DFKI')}))))
```

Figure 5: $\mathcal{NLL}$ representation of example sentence after equivalence transformation and simplification (speaker and hearer have been omitted for the sake of perspicuity).

All three example configurations explicitly show the need for simplification into a canonical representation that supports uniform further processing and, finally, the translation into an appropriate application directive. The DISCO general time semantics legitimates the grouping of multiple (and possibly nested) temporal restrictions on the same variable into the scope of a single time() predicate by 'unification' of all existing information. For all the given examples $\mathcal{NLL}$ transformations have been implemented that fire on appropriate *meta-syntactic* configurations (see above) and rewrite to a unified structure. Because the hand-coding of complex rewrite rules is vulnerable to errors, we see clearly here how a declarative rule compiler would facilitate simplification (cf. § 3.3,§ 4).

Figure 5 shows the $\mathcal{NLL}$ representation of the example after core equivalence transformation and simplification: nested binary conjunctions have been flattened, all temporal restrictions on the appointment variable have been unified and the general temp-in() relation has been substituted for lexically determined predicates.

3.5.3 Translation into Target Language

The internal representation of the appointment planner as the interface language to the semantics module in the COSMA system (besides the interface to feature structure descriptions) is far more restricted in expressive power than $\mathcal{NLL}$. The IR language is syntactically close to Common-Lisp (all data structures are represented as nested association lists) — it basically, provides a fixed set of negotiation primitives (arrange(), accept(), modify() et al.) that take one or more appointment descriptions as arguments. Temporal expressions (from natural language input) constrain appointment descriptions according to the slots time or duration.

Despite the obvious simplicity of the target language in the COSMA prototype we have chosen to follow the general $\mathcal{NLL}$ compiler approach in mapping into IR. Just as inside the $\mathcal{NLL}$ module expressions of the target language are represented as (hierarchically organized) Common-Lisp structures thus allowing us to view the process of translation as a series of tree transformations again and to make use of the optimized $\mathcal{NLL}$ tree traversal drivers (see above). The Common-Lisp surface syntax of IR expressions is generated from a (still hand-coded) pretty printer (unparser) that makes IR abstract syntax trees print as Lisp association lists.

Applying the compiler approach to the $\mathcal{NLL}$ to IR interface, we construe the translation module as a set of rewrite rules that are applied in post-oder (bottom-up) tree traversal[7] just like the other transformations and simplifications we have seen so far. Though the example sentence has been chosen because it maps nicely into an appropriate IR expression (see Figure 6) by strictly local rewrite rules the tree transformation mechanism has turned out to be suitable in handling more

[7]Bottom-up processing is the control regime of choice because several high-level rewrite rules depend on the argument type of embedded structures. Generally a mixed control strategy is achieved by splitting transformations into (not necessarily disjoint) sets and applying these in order.

```
((arrange . ((type . :appointment)
             (time . ((month . :october)
                      (day-of-month . 23)
                      (hour . 13)
                      (minute . 30)))
             (place . "DFKI"))))
```

- termin(instance:?x) → (type . :appointment)
- (?t | time(instance:?t
  mon:(?m | oktober(instance:?m))
  day:23
  hour:13
  min:30))
  → (time . ((month . :october) (day-of-month . 23)
  (hour . 13) (minute . 30)))
- 'DFKI' → (place . "DFKI")
- (?x | and{termin(...) temp-in(...) space-in(...)})
  → *appointment structure*
- request(theme:^vereinbar(...))
  → (arrange . (...))

Figure 6: Representation of the example semantics in the COSMA internal representation language (left side). The structure as shown in Figure 4 (after some very basic domain-specific inference that is not discussed here) has been transformed according to the rewrite rules given on the right.

complex transformation rules and mediation of differences in expressive power as well.[8] For details on rewrite rules in the given example see Figure 6.

4 Conclusions and Prospectus

The purpose of this paper has been to argue for an approach to the software design of semantic representation languages (SRLs) modeled after standard design in programming language (PL) and compiler technology. The approach proceeds from an analysis of the goals of SRLs, noting especially the following:

- there is a need for experimentation in SRLs design because of many unsolved problems
- there is a need for general tools such as readers and printers
- SRLs must participate in a variety of interfaces
- SRLs must support inference

These design goals are largely satisfied by implementing SRLs using standard PL technology. A BNF-like specification provides a modifiable basis from which experimentation may proceed. The use of this in a (un)parser-generator provides a useful reader (and printer). Given parser and unparser, interfaces may be constructed in a variety of ways, including especially compilation: the transformation of abstract syntax trees (parser output and unparser input). Finally, some inference is implementable using the same transformations employed for interfaces.

We have illustrated the approach and its benefits by reporting on extensive experimentation with $\mathcal{NLL}$, a public-domain SRL implemented in Zebu using compiler technology. $\mathcal{NLL}$has been interfaced to (several versions of) three different grammar systems, four different application systems, and a variety of other modules. The interfaces are exactly defined and easily implemented and modified; several are done in compiler-style (through tree transformations on abstract syntax trees), and these are quite flexible, allowing sensitive manipulations of representations. Finally, a variety of semantic inference rules have been implemented and used.

4.1 Additional Benefits—Comparing SRLs Theoretically

Several further benefits have accrued to this approach. Reverse translations, which are needed between components in dialogue systems, and which are useful in architectures emphasizing component feedback (Görz 1992), share a good deal of code with translations, facilitating their implementation.

[8] E.g. when translating a meeting *event* into an appointment structure (in sentences like *I want to meet you on October 23 at 1:30 PM in the DFKI.*) all slots except the type of the appointment object are determined from restrictions on the event variable. Rewrite rules of this type therefore have an extended domain of locality.

The ability to define interfaces on the basis of strings allows not only the tighter definition of interfaces noted above, but also flexibility in architectures—e.g., no assumptions need to be made about whether SRL modules run in the same address space or even on the same machine as the modules they communicate with. (It even proven useful in communicating with modules in the same LISP image because it eliminated the need for coordination in packaging.)

Perhaps the most interesting benefit has come in the ability to implement and therefore compare alternative SRLs relatively easily. To take one example, we added an expression type RESTRICTED PARAMETER to the language in an effort to explore situation semantics (Gawron and Peters 1990,20). In a further extension, in order to represent the two-part structure of DRT (Kamp 1981), a slot was added to all formula structures for untrapped parameters (those whose scope extended beyond operators in the formula). This allowed representations such as the following:

A consultant was not hired by a manager.

```
?y ; ~hire(agt:(?x|manager(inst:?x))
          thm:(?y|consultant(inst:?y)))
```

Thus the formula here indicates that '?y' scopes past all operators in the formula, in particular past the negation operator '~' (which otherwise nonselectively binds everything within its scope in DRT). Although none of this goes beyond well-known implementation strategies for DRT (Johnson and Klein 1986), the compiler approach allows us to write conversion routines to the language of generalized quantifiers (from GQT, cf. Barwise and Cooper 1981) form, which were implemented to convert, e.g. to the following form:

```
(exists ?y consultant(inst:?y)
        ~(exists ?x manager(inst:?x)
                 hire(agt:?x thm:?y)))
```

This facilitates practical experimentation with the different theories of semantics—LGQ and DRT, surely a useful effect in computational semantics.

4.2 Future Directions

The focus of our current work is allowing a declarative (and easier) specification for transformation rules, as discussed in Section 3.3 above. We are employing a variant of a typed feature description language for this purpose.[9] The advantage of the specification language over direct coding is that it relieves a user from the burden of producing the LISP functions which simplify in the desired way, and it allows us, the designers, much finer control the sorts of simplifications we support. For example, we can perform type-checking to ensure that type-inconsistent specifications are disallowed, and we can examine the properties of sets of rules to check for likely problems in termination, etc.

The following is a specification of the "single element specification rule" (cf. § 3.3) for conjunctions (and disjunctions):

```
n-ary-connective-wff: [connective 'and|'or
                       sub-wffs {nll-wff: p}]
--> p
```

The left-hand side of this rule specifies a type of logical expression, namely a formula with an n-ary connective, which is furthermore constrained so that its connective must be `'and` or `'or`, and whose subformula set consists of the single formula `p`. Given a formula of this type, the rule licenses a simplification to `p`. Thus this is a rendering of the simplification specified in § 3.3, which we repeat here for convenience:

$$\text{single element} \quad \mathsf{AND}\{p\} \rightarrow p$$

The left side of the specification is interpreted as a pattern which an input expression is matched against—e.g.AND{walk(agt:s)}—and a pair $<$ **match?** : *bool*, **bindings** $>$ is returned, where

[9]This idea of using a typed feature description langauge for the transformation specification language is due to Karsten Konrad.

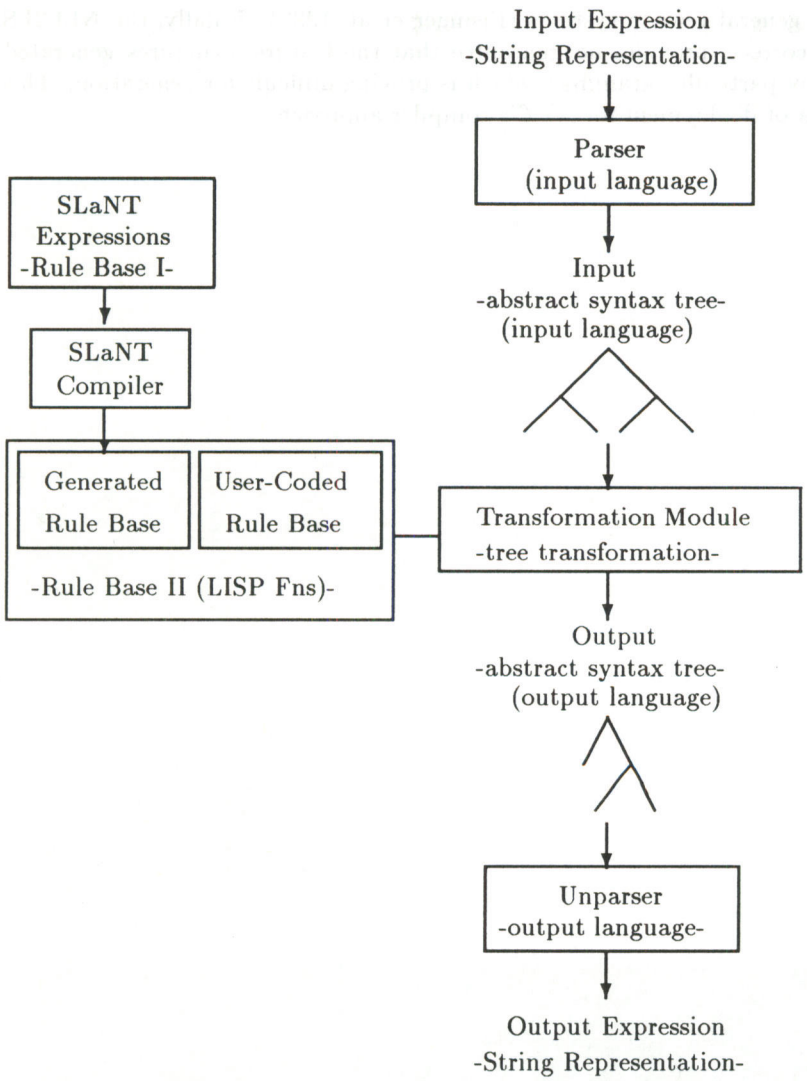
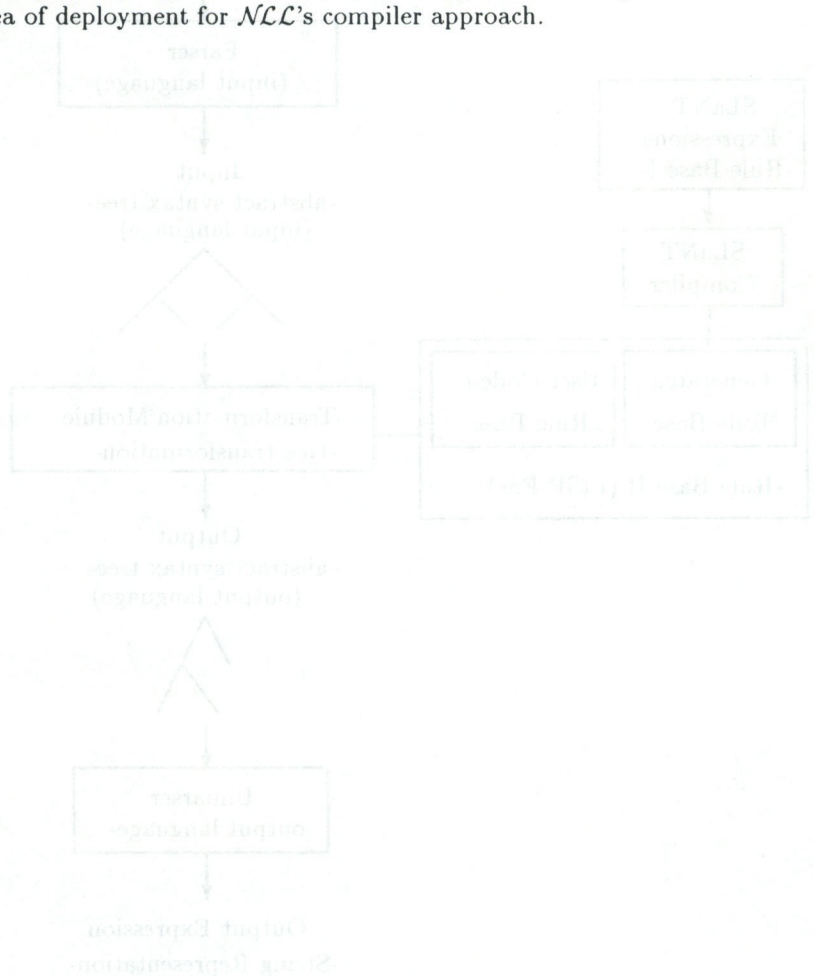


Figure 7: Architecture of the transformation module. User-specified transformations are compiled into LISP functions, which—together with hand-coded transformations—provide a rule base. The transform component inputs a logical expression and (a selection from) the rule group and processes in preorder or postorder fashion (cf. § 3.2 above, Inferences and Compilations).

bindings simply keeps track of metavariables (e.g., `p`), and what they were matched with—here, `(p.walk(agt:s))`. In cases such as these where there is a positive match, the bindings are used in the construction of a replacement node—in this case just the value of the metavariable `p` itself—`walk(agt:s)`. In the current architecture, the entire specification is compiled to a LISP function, which may then be combined with hand-coded specifications in designing a transformation module. This allows for flexibility in case the specification language is too weak, and it is clear that the specification language would benefit from further flexibility, e.g., the possibility of specifying that OPERATIONS—e.g., substitutions—be perform on logical expressions. Figure 7 sketches the planned architecture.

We are also investigating more dedicated support for other "back-end" modules and communications, especially deindexation in the context or discourse memory module, predicate disambiguation in the application interface, support for (or an interface to) a dialogue management module (speech act recognition), and a lexical semantics interface. We also continue to investigate stronger

inferential systems, such as general rewrite systems (Eisinger et al. 1992). Finally, the NLL2FS interface, while technically correct, provides no guarantee that the feature structures generated have anything to do with any particular grammar, which is proving difficult for generation. This too may be a promising area of deployment for $\mathcal{NLL}$'s compiler approach.



References

Aho, A., J. Hopcroft, and J. Ullman. 1983. *Data Structures and Algorithms.* Addison Wesley.

Aho, A., R. Sethi, and J. Ullman. 1986. *Compilers: Principles, Techniques and Tools.* Addison Wesley.

Allen, J. 1987. *Natural Language Understanding.* Menlo Park: Benjamin/Cummings.

Barwise, J., and R. Cooper. 1981. Generalized Quantifiers and Natural Language. *Linguistics and Philosophy* 4(2):159–219.

Barwise, J., and J. Perry. 1983. *Situations and Attitudes.* Cambridge: MIT Press.

Bobrow, R. J., P. Resnick, and R. M. Weischedel. 1990. Multiple Underlying Systems: Translating User Requests into Programs to Produce Answers. In *Proceedings of the 28th Annual Meeting of the ACL*, 227–234. Association for Computational Linguistics.

Eisinger, N., A. Nonnengart, and A. Präcklein. 1992. Termersetzungssysteme. In *Deduktionssysteme*, ed. K.H.Bläsius and H.-J.Bürckert, chapter 3.4, 126–149. München: Oldenbourg. 2nd, Revised Edition.

Friedman, D., M. Wand, and C. Haynes. 1992. *Essentials of Programming Languages.* New York: McGraw-Hill.

Gawron, J. M., and S. Peters. 1990. *Anaphora and Quantification in Situation Semantics.* Stanford University: CSLI Lecture Notes.

Görz, G. 1992. Kognitiv orientierte Architekturen für die Sprachverarbeitung. Technical Report ASL-TR-39-92, Fachbereich Informatik, Universität Erlangen-Nürnberg.

Groenendijk, J., and M. Stokhof. 1991. Dynamic Predicate Logic. *Linguistics and Philosophy* 14(1):39–100.

Johnson, M., and M. Kay. 1990. Semantic Abstraction and Anaphora. In *Proceedings of COLING-90*, ed. H. Karlgren, 17–27. Helsinki. COLING.

Johnson, M., and E. Klein. 1986. Discourse, Anaphora and Parsing. Report CSLI-86-63, CSLI, Stanford, Oct.

Kamp, H. 1981. A Theory of Truth and Semantic Representation. In *Formal Methods in the Study of Language*, ed. J.Groenendijk, T. Janssen, and M. Stokhof. Amsterdam: Mathematical Centre.

Kasper, W. 1992. Integration of Syntax and Semantics in Feature Structures. DFKI-Workshop *Natürlichsprachliche Systeme*, Saarbrücken, 23. Okt. Vortrag.

Laubsch, J. 1992a. The Semantics Application Interface. In *Applied Natural Language Processing*, ed. H. Haugeneder. ??address: ??publisher.

Laubsch, J. 1992b. **Zebu**: A Tool for Specifying Reversible LALR(1) Parsers. Technical report, Hewlett-Packard Laboratories, Palo Alto, CA, July.

Laubsch, J. December 1989. Logical Form Simplification. STL report, Hewlett-Packard.

Laubsch, J., and J. Nerbonne. 1991. An Overview of $\mathcal{NLL}$. Technical report, Hewlett-Packard Laboratories, Palo Alto, July.

Martin, P., D. Appelt, and F. Pereira. 1983. Transportability and Generality in a Natural-Language Interface System. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 573–581. Los Altos. IJCAI, Morgan Kaufmann.

Nerbonne, J. 1992. $\mathcal{NLL}$ Models. Research report, DFKI, Saarbrücken.

Nerbonne, J., and D. Proudian. 1987. The HP-NL System. Technical report, Hewlett-Packard Labs.

Neumann, G., S. Oepen, and S. P. Spackman. 1993. Design and Implementation of the COSMA System. Technical report, Deutsches Forschungszentrum für Künstliche Intelligenz, Saarbrücken, Germany.

Reasoning Systems Inc., Palo Alto. 1990. REFINE *User's Guide*, May.

Scha, R. 1981. Distributive, Collective and Cumulative Quantification. In *Truth, Interpretation and Information*, ed. J. Groenendijk, T. Janssen, and M. Stokhof, 131–158. Dordrecht: Foris.

Scha, R., and D. Stallard. 1988. Multi-Level Plurals and Distributivity. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, 17–24. ACL.

Stallman, R. M. 1992. *Using and Porting GNU CC.* Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA, preliminary 2nd edition, May. Remember: GNU's not Unix!

Weischedel, R. 1979. A New Semantic Computation while Parsing: Presupposition and Entailment. In *Syntax and Semantics II: Presupposition*, ed. C. Oh and D. Dineen, 155–182. New York: Academic Press.

Young, S., A. Hauptmann, W. Ward, E. Smith, and P. Werner. 1989. High Level Knowledge Sources in Usable Speech Recognition Systems. *Communications of the ACM* 32(2):183–194.

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
D-6750 Kaiserslautern
FRG

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse bezogen werden.
Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all publisched papers so far can be ordered from the above address.
The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

RR-92-02
Andreas Dengel, Rainer Bleisinger, Rainer Hoch, Frank Hönes, Frank Fein, Michael Malburg:
Π_{ODA}: The Paper Interface to ODA
53 pages

RR-92-03
Harold Boley:
Extended Logic-plus-Functional Programming
28 pages

RR-92-04
John Nerbonne: Feature-Based Lexicons:
An Example and a Comparison to DATR
15 pages

RR-92-05
Ansgar Bernardi, Christoph Klauck, Ralf Legleitner, Michael Schulte, Rainer Stark:
Feature based Integration of CAD and CAPP
19 pages

RR-92-06
Achim Schupetea: Main Topics of DAI: A Review
38 pages

RR-92-07
Michael Beetz:
Decision-theoretic Transformational Planning
22 pages

RR-92-08
Gabriele Merziger: Approaches to Abductive Reasoning - An Overview -
46 pages

RR-92-09
Winfried Graf, Markus A. Thies: Perspektiven zur Kombination von automatischem Animationsdesign und planbasierter Hilfe
15 Seiten

RR-92-10
M. Bauer: An Interval-based Temporal Logic in a Multivalued Setting
17 pages

RR-92-11
Susane Biundo, Dietmar Dengler, Jana Koehler:
Deductive Planning and Plan Reuse in a Command Language Environment
13 pages

RR-92-13
Markus A. Thies, Frank Berger:
Planbasierte graphische Hilfe in objektorientierten Benutzungsoberflächen
13 Seiten

RR-92-14
Intelligent User Support in Graphical User Interfaces:
1. InCome: A System to Navigate through Interactions and Plans
   Thomas Fehrle, Markus A. Thies
2. Plan-Based Graphical Help in Object-Oriented User Interfaces
   Markus A. Thies, Frank Berger

22 pages

RR-92-15
Winfried Graf: Constraint-Based Graphical Layout of Multimodal Presentations
23 pages

RR-92-16
Jochen Heinsohn, Daniel Kudenko, Berhard Nebel, Hans-Jürgen Profitlich: An Empirical Analysis of Terminological Representation Systems
38 pages

RR-92-17
Hassan Aït-Kaci, Andreas Podelski, Gert Smolka:
A Feature-based Constraint System for Logic Programming with Entailment
23 pages

RR-92-18
John Nerbonne: Constraint-Based Semantics
21 pages

RR-92-19
Ralf Legleitner, Ansgar Bernardi, Christoph Klauck: PIM: Planning In Manufacturing using Skeletal Plans and Features
17 pages

RR-92-20
John Nerbonne: Representing Grammar, Meaning and Knowledge
18 pages

RR-92-21
Jörg-Peter Mohren, Jürgen Müller
Representing Spatial Relations (Part II) -The Geometrical Approach
25 pages

RR-92-22
Jörg Würtz: Unifying Cycles
24 pages

RR-92-23
Gert Smolka, Ralf Treinen:
Records for Logic Programming
38 pages

RR-92-24
Gabriele Schmidt: Knowledge Acquisition from Text in a Complex Domain
20 pages

RR-92-25
Franz Schmalhofer, Ralf Bergmann, Otto Kühn, Gabriele Schmidt: Using integrated knowledge acquisition to prepare sophisticated expert plans for their re-use in novel situations
12 pages

RR-92-26
Franz Schmalhofer, Thomas Reinartz, Bidjan Tschaitschian: Intelligent documentation as a catalyst for developing cooperative knowledge-based systems
16 pages

RR-92-27
Franz Schmalhofer, Jörg Thoben: The model-based construction of a case-oriented expert system
18 pages

RR-92-29
Zhaohui Wu, Ansgar Bernardi, Christoph Klauck:
Skeletel Plans Reuse: A Restricted Conceptual Graph Classification Approach
13 pages

RR-92-30
Rolf Backofen, Gert Smolka
A Complete and Recursive Feature Theory
32 pages

RR-92-31
Wolfgang Wahlster
Automatic Design of Multimodal Presentations
17 pages

RR-92-33
Franz Baader: Unification Theory
22 pages

RR-92-34
Philipp Hanschke: Terminological Reasoning and Partial Inductive Definitions
23 pages

RR-92-35
Manfred Meyer:
Using Hierarchical Constraint Satisfaction for Lathe-Tool Selection in a CIM Environment
18 pages

RR-92-36
Franz Baader, Philipp Hanschke:
Extensions of Concept Languages for a Mechanical Engineering Application
15 pages

RR-92-37
Philipp Hanschke: Specifying Role Interaction in Concept Languages
26 pages

RR-92-38
Philipp Hanschke, Manfred Meyer:
An Alternative to Θ-Subsumption Based on Terminological Reasoning
9 pages

RR-92-40
Philipp Hanschke, Knut Hinkelmann: Combining Terminological and Rule-based Reasoning for Abstraction Processes
17 pages

RR-92-41
Andreas Lux: A Multi-Agent Approach towards Group Scheduling
32 pages

RR-92-42
John Nerbonne:
A Feature-Based Syntax/Semantics Interface
19 pages

RR-92-43
Christoph Klauck, Jakob Mauss: A Heuristic driven Parser for Attributed Node Labeled Graph Grammars and its Application to Feature Recognition in CIM
17 pages

RR-92-44
Thomas Rist, Elisabeth André: Incorporating Graphics Design and Realization into the Multimodal Presentation System WIP
15 pages

RR-92-45
Elisabeth André, Thomas Rist: The Design of Illustrated Documents as a Planning Task
21 pages

RR-92-46
Elisabeth André, Wolfgang Finkler, Winfried Graf, Thomas Rist, Anne Schauder, Wolfgang Wahlster: WIP: The Automatic Synthesis of Multimodal Presentations
19 pages

RR-92-47
Frank Bomarius: A Multi-Agent Approach towards Modeling Urban Traffic Scenarios
24 pages

RR-92-48
Bernhard Nebel, Jana Koehler:
Plan Modifications versus Plan Generation: A Complexity-Theoretic Perspective
15 pages

RR-92-51
Hans-Jürgen Bürckert, Werner Nutt:
On Abduction and Answer Generation through Constrained Resolution
20 pages

RR-92-52
Mathias Bauer, Susanne Biundo, Dietmar Dengler, Jana Koehler, Gabriele Paul: PHI - A Logic-Based Tool for Intelligent Help Systems
14 pages

RR-92-54
Harold Boley: A Direkt Semantic Characterization of RELFUN
30 pages

RR-92-55
John Nerbonne, Joachim Laubsch, Abdel Kader Diagne, Stephan Oepen: Natural Language Semantics and Compiler Technology
17 pages

DFKI Technical Memos

TM-91-12
Klaus Becker, Christoph Klauck, Johannes Schwagereit: FEAT-PATR: Eine Erweiterung des D-PATR zur Feature-Erkennung in CAD/CAM
33 Seiten

TM-91-13
Knut Hinkelmann: Forward Logic Evaluation: Developing a Compiler from a Partially Evaluated Meta Interpreter
16 pages

TM-91-14
Rainer Bleisinger, Rainer Hoch, Andreas Dengel: ODA-based modeling for document analysis
14 pages

TM-91-15
Stefan Bussmann: Prototypical Concept Formation An Alternative Approach to Knowledge Representation
28 pages

TM-92-01
Lijuan Zhang: Entwurf und Implementierung eines Compilers zur Transformation von Werkstückrepräsentationen
34 Seiten

TM-92-02
Achim Schupeta: Organizing Communication and Introspection in a Multi-Agent Blocksworld
32 pages

TM-92-03
Mona Singh:
A Cognitiv Analysis of Event Structure
21 pages

TM-92-04
Jürgen Müller, Jörg Müller, Markus Pischel, Ralf Scheidhauer:
On the Representation of Temporal Knowledge
61 pages

TM-92-05
Franz Schmalhofer, Christoph Globig, Jörg Thoben: The refitting of plans by a human expert
10 pages

TM-92-06
Otto Kühn, Franz Schmalhofer: Hierarchical skeletal plan refinement: Task- and inference structures
14 pages

TM-92-08
Anne Kilger: Realization of Tree Adjoining Grammars with Unification
27 pages

DFKI Documents

D-92-06
Hans Werner Höper: Systematik zur Beschreibung von Werkstücken in der Terminologie der Featuresprache
392 Seiten

D-92-07
Susanne Biundo, Franz Schmalhofer (Eds.): Proceedings of the DFKI Workshop on Planning
65 pages

D-92-08
Jochen Heinsohn, Bernhard Hollunder (Eds.): DFKI Workshop on Taxonomic Reasoning Proceedings
56 pages

D-92-09
Gernod P. Laufkötter: Implementierungsmöglichkeiten der integrativen Wissensakquisitionsmethode des ARC-TEC-Projektes
86 Seiten

D-92-10
Jakob Mauss: Ein heuristisch gesteuerter Chart-Parser für attributierte Graph-Grammatiken
87 Seiten

D-92-11
Kerstin Becker: Möglichkeiten der Wissensmodellierung für technische Diagnose-Expertensysteme
92 Seiten

D-92-12
Otto Kühn, Franz Schmalhofer, Gabriele Schmidt: Integrated Knowledge Acquisition for Lathe Production Planning: a Picture Gallery (Integrierte Wissensakquisition zur Fertigungsplanung für Drehteile: eine Bildergalerie)
27 pages

D-92-13
Holger Peine: An Investigation of the Applicability of Terminological Reasoning to Application-Independent Software-Analysis
55 pages

D-92-14
Johannes Schwagereit: Integration von Graph-Grammatiken und Taxonomien zur Repräsentation von Features in CIM
98 Seiten

D-92-15
DFKI Wissenschaftlich-Technischer Jahresbericht 1991
130 Seiten

D-92-16
Judith Engelkamp (Hrsg.): Verzeichnis von Softwarekomponenten für natürlichsprachliche Systeme
189 Seiten

D-92-17
Elisabeth André, Robin Cohen, Winfried Graf, Bob Kass, Cécile Paris, Wolfgang Wahlster (Eds.): UM92: Third International Workshop on User Modeling, Proceedings
254 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-92-18
Klaus Becker: Verfahren der automatisierten Diagnose technischer Systeme
109 Seiten

D-92-19
Stefan Dittrich, Rainer Hoch: Automatische, Deskriptor-basierte Unterstützung der Dokumentanalyse zur Fokussierung und Klassifizierung von Geschäftsbriefen
107 Seiten

D-92-21
Anne Schauder: Incremental Syntactic Generation of Natural Language with Tree Adjoining Grammars
57 pages

D-92-23
Michael Herfert: Parsen und Generieren der Prolog-artigen Syntax von RELFUN
51 Seiten

D-92-24
Jürgen Müller, Donald Steiner (Hrsg.): Kooperierende Agenten
78 Seiten

D-92-25
Martin Buchheit: Klassische Kommunikations- und Koordinationsmodelle
31 Seiten

D-92-26
Enno Tolzmann:
Realisierung eines Werkzeugauswahlmoduls mit Hilfe des Constraint-Systems CONTAX
28 Seiten

D-92-27
Martin Harm, Knut Hinkelmann, Thomas Labisch: Integrating Top-down and Bottom-up Reasoning in COLAB
40 pages

D-92-28
Klaus-Peter Gores, Rainer Bleisinger: Ein Modell zur Repräsentation von Nachrichtentypen
56 Seiten

Natural Language Semantics and Compiler Technology

John Nerbonne, Joachim Laubsch, Abdel Kader Diagne, Stephan Oepen

RR-92-55
Research Report