SYSTEM DEMONSTRATION
NATURAL LANGUAGE GENERATION WITH ABSTRACT MACHINE

Evgeniy Gabrilovich and Nissim Francez
Computer Science Department
Technion, Israel Institute of Technology
32000 Haifa, Israel
{gabr,francez}@cs.technion.ac.il

Shuly Wintner
Seminar für Sprachwissenschaft
Universität Tübingen
72074 Tübingen, Germany
shuly@sfs.nphil.uni-tuebingen.de

Abstract

We present a system for Natural Language Generation based on an Abstract Machine approach. Our abstract machine operates on grammars encoded in a unification-based Typed Feature Structure formalism, and is capable of both generation and parsing. For efficient generation, grammars are first inverted to a suitable form, and then compiled into abstract machine instructions. A dual compiler translates the same input grammar into an abstract machine program for parsing. Both generation and parsing programs are executed under the same (chart-based) evaluation strategy. This results in an efficient, bidirectional (parsing/generation) system for Natural Language Processing. Moreover, the system possesses ample debugging features, and thus can serve as a user-friendly environment for bidirectional grammar design and development.

1 Overview

An input for the generation[1] task is a logical form which represents a meaning, and a grammar to govern the generation process. The output consists of one or more phrases in the language of the grammar whose meaning is (up to logical equivalence) the given logical form.

The system to be demonstrated applies an Abstract Machine (AM) approach for Natural Language Generation, within the framework of Typed Feature Structures (Carpenter, 1992b). Such a machine is an abstraction over an ordinary computer, lying somewhere between regular high-level languages and common hardware architectures. Programming an Abstract Machine has proved fruitful in previous research, reaching a peak as a highly efficient technique to build Prolog compilers (Ait-Kaci, 1991).

AMALIA[2] has two compilers of grammars into Abstract Machine instructions; the outputs of compilation are AM programs which perform either chart generation or chart parsing, both according to the given grammar. Both tasks use an auxiliary table (chart) to store intermediate processing results. AMALIA has a uniform core engine for bottom-up chart processing, which interprets the given (abstract machine) program, and realizes the generation or parsing task. In the case of generation it is the given semantic meaning whose components are consumed in the process. The only differences between the two processing directions are in the nature of chart items and interpretation of the final results. Thereby, AMALIA makes dual use of its chart and forms a complete bidirectional natural language system, which is considered an advantage in the literature (Strzalkowski, 1994). The system is capable of very efficient processing, since grammars are precompiled directly into abstract machine instructions, which are subsequently executed over and over.

[1] In this work we mean by "generation" what is sometimes known also as "syntactic generation". Thus, no text planning, speaker intentions and the like are considered here.

[2] The acronym stands for "Abstract MAchine for LInguistic Applications".

Logical forms specified as meanings by input grammars are given in a so-called predicate-argument structure[3]. Thus, meanings are built from basic units (feature structures), each having a predicate and (optionally) a number of arguments. Our approach also allows λ-abstractions over predicate-argument constructs, as well as systematic encoding of second- and higher-order functions.

Grammars are usually designed in a form oriented towards the analysis of a string and not towards generation from a (usually nested) semantic form. In other words, rules reflect the phrase structure and not the predicate-argument structure. It is therefore useful to transform the grammar in order to enable systematic reflection of any given logical form in the productions. For this purpose, we apply to the input grammar an inversion procedure, based upon[4] (Samuelsson, 1995), to render the rules with the nested predicate-argument structure, corresponding to that of input logical forms. The resultant "inverted" grammar is thus more suitable for performing the generation task. Once the grammar is inverted, the generation process can be directed by the input semantic form; elements of the input are consumed during generation just like words are consumed during parsing. Grammars must satisfy certain requirements in order for them to be invertible. However, the requirements are not overly restrictive and allow encoding of a variety of natural language grammars.

Grammar inversion is performed prior to compilation for generation. The given grammar is enhanced in a way that will ultimately enable to reconstruct the words spanned by the semantic forms. To achieve this aim, each rule constituent is extended by an additional special-purpose feature. The value of this feature for the rule's head is set to the concatenation of its values in the body constituents, to reflect the original phrase structure of the rule.

Figure 1 delineates an overview of AM-based generation. After the grammar is inverted, it is compiled into the abstract machine code. At run time, the given logical form is decomposed into meaning components, which initialize the AM chart, and then the generation program is invoked. If generation terminates, it yields a (possibly empty) set of feature structures; a grammar-independent post-processing routine analyzes these structures and retrieves the generated phrases per se.

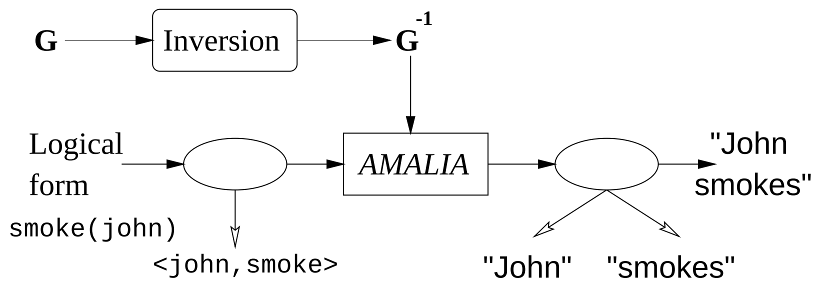

Figure 1: An overview of generation with Abstract Machine.

This outline only contains a brief description of software features and capabilities. For a theoretical background on using an abstract machine for Natural Language Parsing refer to (Wintner and Francez, 1995; Wintner, 1997). (Gabrilovich, 1998) gives more details on Natural Language Generation with Abstract Machine. (Wintner, Gabrilovich, and Francez, 1997a) describes AMALIA

[3] The predicate-argument structure is analogous to the familiar representation of semantic logical forms with first-order terms.

[4] Samuelsson's inversion algorithm was originally developed for definite clause grammars. We adapted it to the Typed Feature Structure formalism.

as a unified platform for parsing and generation, elaborating more on the way the two directions are integrated into a single system.

2 $\mathcal{A}$MALIA functionality

$\mathcal{A}$MALIA operates on input grammars encoded in a subset of the ALE specification language (Carpenter, 1992a). In particular, $\mathcal{A}$MALIA supports the same type hierarchies as ALE does, with exactly the same specification syntax. This means that the user can specify any bounded-complete partial order as the type hierarchy. In contrast to ALE, $\mathcal{A}$MALIA allows appropriateness loops in the type hierarchy. On the other hand, $\mathcal{A}$MALIA does not support type constraints and relational extensions.

$\mathcal{A}$MALIA uses a subset of ALE's syntax for describing totally well-typed, possibly cyclic, non-disjunctive feature structures. Set values, as in ALE, are not supported, but list values are. $\mathcal{A}$MALIA does not respect the distinction between *intensional* and *extensional* types (Carpenter, 1992b, Chapter 8). Also, feature structures cannot incorporate inequality constraints.

$\mathcal{A}$MALIA supports macros in a similar way to ALE. The syntax is the same, and macros can have parameters or call other macros (though not recursively, of course). ALE's special macros for lists are supported by $\mathcal{A}$MALIA. Lexical rules are not supported in this version of $\mathcal{A}$MALIA. $\mathcal{A}$MALIA's syntax for phrase structure rules is similar to ALE's, with the exception of the `cats>` specification (permitting a list of categories in the body of a rule) which is not supported. $\mathcal{A}$MALIA uses ALE's syntax in describing lexical entries, and allows disjunctive lexical entries, separated by semicolons.

$\mathcal{A}$MALIA is implemented in ANSI-C, augmented by *lex* and *yacc* to implement the input acquisition module, and *Tcl/Tk* to build the graphical user interface. The application is compatible with a variety of platforms, such as SUN and SILICON GRAPHICS workstations running UNIX operating system, as well as IBM PC running WINDOWS'95 and LINUX. For a detailed description and a complete user's guide of $\mathcal{A}$MALIA refer to (Wintner, Gabrilovich, and Francez, 1997b).

There are two versions of $\mathcal{A}$MALIA: an interactive, user-friendly program with a graphical user interface, and a non-interactive but more efficient version for batch processing. The former program provides extensive debugging capabilities, and is ideally suited for developing *reversible grammars*.

Figure 2 presents a sample snapshot of the program screen. In the case of generation, the "Input string" field specifies the name of the query file, which contains (an ALE description of) a feature structure representing the input semantic form. In this example, the query file encodes the logical form $\forall x(\mathtt{man}(x) \rightarrow \mathtt{dream}(x))$; the feature structure for this query is shown in the figure over the main program screen. The "Messages" window displays the phrases generated (if any). The feature structures that encode these phrases are also displayed graphically, in separate windows (not shown in the figure). In the case of parsing, the "Input string" field contains the word string to be parsed, and the program eventually displays feature structures assigned to this string by the parser (if any).

References

Ait-Kaci, Hassan. 1991. *Warren's Abstract Machine: A Tutorial Reconstruction.* The MIT Press.

Carpenter, Bob. 1992a. ALE - the attribute logic engine: User's guide. Technical report, Laboratory for Computational Linguistics, Philosophy Department, Carnegie Mellon University, Pittsburgh, PA.

Carpenter, Bob. 1992b. *The Logic of Typed Feature Structures. With Applications to Unification Grammars, Logic Programs and Constraint Resolution.* Cambridge University Press.

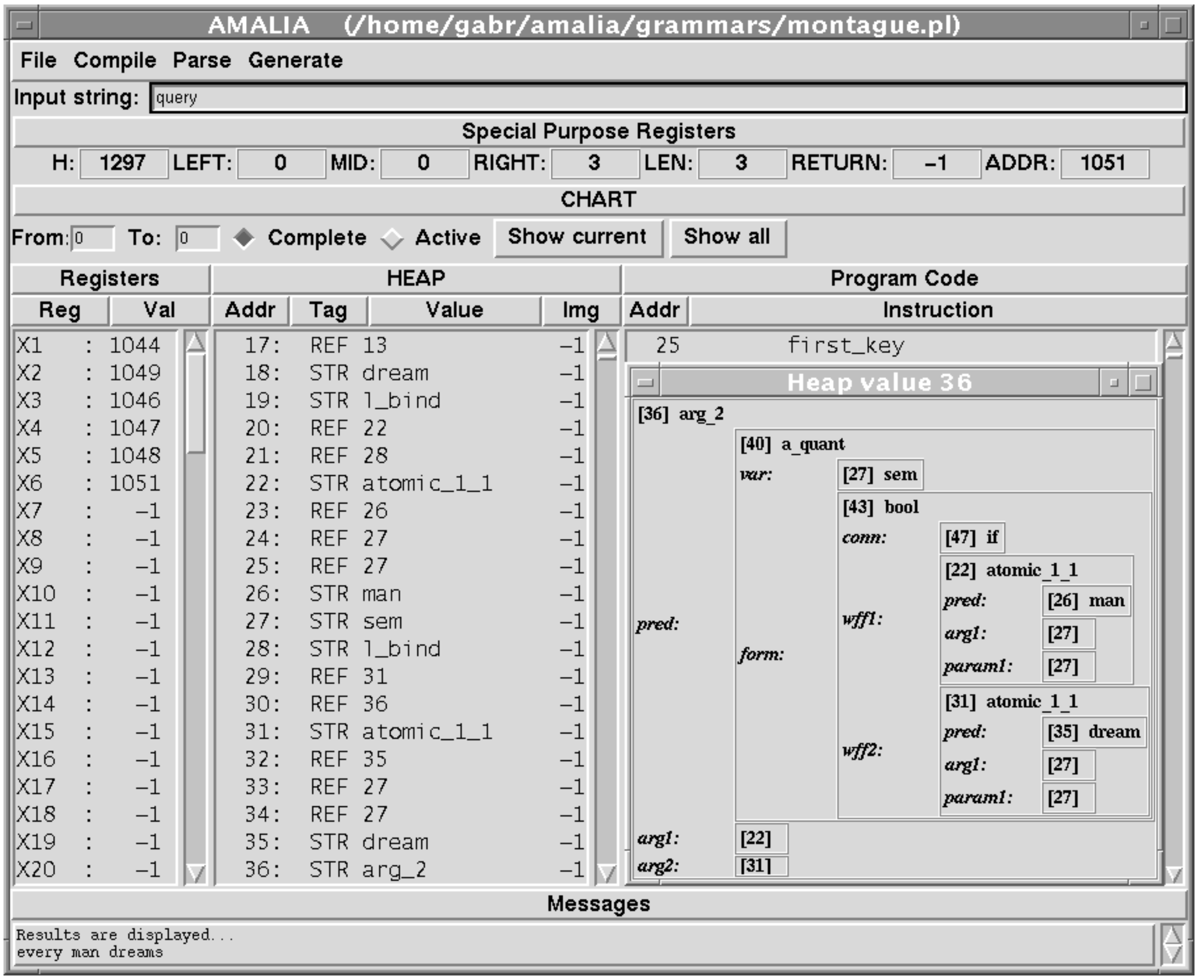

Figure 2: A sample screen shot of $\mathcal{A}$MALIA.

Gabrilovich, Evgeniy. 1998. Natural language generation by abstract machine for typed feature structures. Master's thesis, Technion, Israel Institute of Technology, Haifa, Israel. In preparation.

Samuelsson, Christer. 1995. An efficient algorithm for surface generation. In *Proc. of the 14th Int'l Joint Conference on Artificial Intelligence, Montreal, Canada*, pp. 1414–1419. Morgan Kaufmann, August.

Strzalkowski, Tomek, editor. 1994. *Reversible Grammar in Natural Language Processing.* The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, The Netherlands.

Wintner, Shuly. 1997. *An Abstract Machine for Unification Grammars.* Ph.D. thesis, Technion, Israel Institute of Technology, Haifa, Israel, January.

Wintner, Shuly and Nissim Francez. 1995. An abstract machine for typed feature structures. In *Proc. of the 5th Workshop on Natural Language Understanding and Logic Programming*, pp. 205–220, Lisbon, May.

Wintner, Shuly, Evgeniy Gabrilovich, and Nissim Francez. 1997a. AMALIA – a unified platform for parsing and generation. In R. Mitkov, N. Nicolov, and N. Nicolov, editors, *Proc. of "Recent Advances in Natural Language Processing" (RANLP'97)*, pp. 135–142, Tzigov Chark, Bulgaria, September.

Wintner, Shuly, Evgeniy Gabrilovich, and Nissim Francez, 1997b. *AMALIA – Abstract MAchine for LInguistic Applications – User's Guide.* Laboratory for Computational Linguistics, Computer Science Deparmtent, Technion, Israel Institute of Technology, Haifa, Israel, June.