TOWARDS COMPUTER-AIDED LINGUISTIC ENGINEERING

Vers une méthodologie et des outils pour le génie linguistique

RÉMI ZAJAC
GRIL
Université Blaise Pascal, 34 avenue Carnot
F–63037 Clermont-Ferrand cedex
remi@ucfsl.uucp

Dans cet article, nous proposons une méthodologie de génie logiciel pour le traitement automatique des langues fondée sur la génération (semi-) automatique de programmes de TALN à partir de spécifications formelles.

Cette méthodologie est concue pour favoriser la réutilisation de spécifications linguistiques dans la génération de différentes applications de TALN, ainsi que le développement incrémental de ces spécifications linguistiques.

Le langage de spécification formelle est fondé sur les structures de traits typés. La réutilisation de spécifications linguistiques est favorisée par l'organisation de ces spécifications dans un style par objets, en plusieurs niveaux de spécificité croissante. Ce langage est suffisement puissant pour pouvoir décrire tout type d'objet linguistique, et a l'avantage d'utiliser une notation largement répandue en TALN.

L'acquisition de connaissances linguistiques formalisées au moyen de ce langage peut être automatisée en utilisant des outils d'exploration de corpus et des éditeurs spécialisés fondés sur ce langage et directement connectés à la base de connaissances linguistiques.

La génération de programmes spécifiques d'analyse ou de génération peut être automatisée dans la mesure ou les langages de programmation cibles sont des langages de programmation logique à contraintes dont les structures de données sont des structures de traits typés.

Les différents éléments constituant cette approche sont actuellement dans un état d'avancement varié. Toutefois, cette approche est déjà partiellement utilisée par différents groupes dans plusieurs projets nationaux et européens, en particulier dans le domaine des dictionnaires électroniques.

TOWARDS COMPUTER-AIDED LINGUISTIC ENGINEERING

RÉMI ZAJAC
GRIL
Université Blaise Pascal, 34 avenue Carnot
F–63037 Clermont-Ferrand cedex
remi@ucfsl.uucp

We outline a framework for computer-aided linguistic engineering based on the automatic generation of NLP programs from specifications, and an automated construction of reusable linguistic specifications. The specification language is based on Typed Feature Structures, and the target programming language is a constraint logic programming language which data structures are typed feature structures. Reusability of linguistic specification is enhanced by the organization of the specifications in an object-oriented style in several layers of increasing specificity, supporting for example the incremental development of grammar specification for sublanguages.

1 A framework for NLP Software Engineering

The development of reliable high-quality linguistic software is a time-consuming, error-prone, and costly process. A parser used in an industrial NLP system is typically developed by one person over several years. The development of a linguistic engineering methodology is one of the major in the development of a language industry. The process of developing an NLP application is an application and an adaptation of the classicalsoftware engineering development methodology and follows three major steps: the initial requirements and specifications expressed in natural language, the formal specification of the system, and finally the implementation of the system [Biggerstaff/Perlis 89].

The requirements specific to a *linguistic* engineering methodology are:

1. The initial requirements are complemented by a *corpus* giving typical examples of the *texts* and linguistic phenomena contained in these texts to be treated by the system;
2. The set of formal specifications constitutes a standardized repository of formalized linguistic knowledge

   reusable across different NLP applications – a crucial property given the sheer size of grammars and dictionaries;

   executable – to be able to test the specifications against corpora.
3. NLP programs are generated (semi-) automatically from formal specifications.

These particularities have the following implications:

1. The availability of a corpus allows to develop a methodology based on sublanguages and corpus analysis, *automating the knowledge acquisition process.*
2. The linguistic specification does not include any information specific to some application (especially, it does not contain any control information), thus the same specification can be reused for different applications (genericity).

   A specification language for describing linguistic knowledge could be based on a feature logic and has an object-oriented inheritance style that makes it possible to distinguish formally between generic knowledge and specific (e.g., sublanguage) knowledge, thus enabling the reuse of specifications in the development of the specifications themselves.

   The expressive power of the specification language (a non-decidable subset of first order logic) allows to remove the conventional distinction between dictionaries and grammars, providing a single homogeneous framework for an integrated development of linguistic knowledge bases.

   The use of a feature-based language also favors standardization, as feature structures become a "lingua franca" for computational linguists.
3. Several modern specialized linguistic programming languages can be the targets of the automated generation process. Since the specification language is based on typed feature structures, natural candidates are «unification-based grammar formalisms».

A Computer-Aided Linguistic Engineering methodology should also address the following points:

- strict separation between pure linguistic knowledge and knowledge about strategies for its use in a particular application, a condition sine qua non for reusability;
- concepts of modularity for linguistic description, e.g., formal separation of knowledge pertaining to different levels of linguistic description, organization of linguistic knowledge in hierarchies (from generic to specific);
- team organization of linguistic development projects.

1 Reusable linguistic descriptions

In software engineering, the use of the term «reusability» covers two main trends: the composition-based approach and the generation-based approach. In the first approach, software components can be plugged together with no or small modifications in order to build software systems: programming languages such as ADA or object-oriented languages are designed to support this type of reuse. This approach is successful when the components are small and perform very precise functions, as for numerical analysis [Biggerstaff/Perlis 89]. In NLP, this approach is exemplified by the reuse of various «engines» such as parsers.

In the second approach, software components are generated (semi-automatically) from a set of formal specifications, instantiating these specifications in a programming language by choosing appropriate data representations and control structures: the knowledge expressed in the specification is reused in various contexts to generate different applications. This approach is successful when a fair amount of domain knowledge is built into the specification and the generation environment, e.g., business knowledge in 4GL (Fourth Generation Languages) environments. This is the approach we envisage for producing NLP programs.

To support reusability and incremental development of specifications, we organize and describe linguistic knowledge using partial specifications and controlled degrees of abstraction in the overall design. This approach should of course be supported by a specification language which will be based on the concept of partial information and provides the means of structuring a specification in a hierarchy of subspecifications of increasing specificity.

We envisage three basic levels of abstraction. The initial design of the linguistic domain is rather abstract and largely free of details. It establishes the basic buildings blocks, the basic structures and the foundations of the linguistic domain. At that level, we could aim at providing a consensual formal definition of these basic building blocks as a first step towards the definition of standards for representing linguistic knowledge. For example, the initial level of abstraction could start from basic descriptive classifications, e.g. at the categorial level nouns, verbs, etc., and from the basic syntactic dependencies between these categories, and give them a formal definition.

A second level of specialization makes choices as for the distribution of linguistic properties into more fine grained categories. At that level, we observe the emergence of linguistic theories, where choices are triggered by theoretical assumptions. Given the relative freedom of structuration, the choice between competing representations should be guided by the concern for modularity and reusability (internal constraints) and by the external constraints on the coverage and the adequacy of the linguistic representation to the needs of NLP of applications. Linguistic specifications should be developed as a set of independently defined modules with well-defined interconnections: modularity is essential in supporting reusability and team work in the development of specifications.

At the third level of specialization, the linguistic organization principles are instantiated in the fully detailed description of specific linguistic phenomena. This level is sufficiently detailed to test the specification against actual sentences (strings of word forms). Previous levels can also be tested but only against abstract descriptions representing sets of sentences. This is also the level at which we have several different instances corresponding to different sublanguages, each sublanguage description reusing the same first and second levels of specification, freeing the linguistic of redoing the same design decisions for each instance. There could also be a structuration among sublanguages which could introduce finer levels of abstraction, thus achieving a higher degree of reusability.

This overall framework in which each level sets partial constraints on the most specific instances is able to support the incremental development of linguistic knowledge by successive refinements and thus, farther reusability.

2 A linguistic description language

The crucial issue in the generation-based approach to reusability is the nature and the definition of the specification language. A specification language has to be defined and implemented as pure logic to fully support reusability. It should be suitable to describe the knowledge of a particular domain and should build on well-accepted notions and notations for that domain: here, natural language processing. In NLP, the emergence of unification-based grammar formalisms promoted the use of feature structures as a «lingua franca» for representing linguistic information. Although some work on unification-based grammar formalisms is motivated by reusability of linguistic specifications (e.g., «reversible grammars»), such work does usually not address the problem of specifications in engineering terms. Furthermore, these formalisms make strong assumptions about the nature of linguistic representation[1] thereby limiting severely the expressive power of these languages. The linguistic specification language is based on a typed version of a logic for feature structures which allows to define specifications at different levels of abstraction. Using this language, it will be possible to eliminate the conventional division between lexical and grammatical knowledge, and also the division between generic and specific (e.g., sublanguage) knowledge.

Such a specification language is executable (although it is potentially infinitely inefficient), and it should be executable for two reasons. First, since the formal specification is the first level of formality in the conception of a software system, correctness cannot be proved by formal means. However, an executable specification language allows at least to test the specifications against examples. Second, it should be possible to derive an actual program (e.g., a parser) from a specification. An *executable* specification language ensures the basic feasibility of an automated generation of NLP programs.

The specification language is formally based on a subset of first-order logic. In order to make it manageable and intuitive, it employs syntactic constructs called Typed Feature Structures (TFSs). The «vocabulary» of the language, its signature, consists of unary predicates (sorts) and binary predicates (features). Moreover, there is an ordering on the sorts (yielding a lower semi-lattice). The structures over which the language is interpreted are determined in that they have to satisfy certain axioms: the features give partial functions, and the ordering on the sorts is reflected as subset inclusion (unary predicates give sets). They are not fully specific, however, which reflects the situation in knowledge representation where the domain of discourse is not completely specified. By adding new axioms, this domain is made more and more specific; in the extreme case, one structure is singled out.

The sort signature is extendable through (recursive) definitions of new sorts; these are done by defining explicit constraints which come from the language itself (the TFS constraint language). The sorts are organized into an *inheritance* hierarchy, with a clean (logical, algebraic and type-theoretic) semantics of inheritance in the object-oriented programming style. The subset of first-order logic can be made more complex by adding logical connectives, such as negation and quantification.

Given the signature, which defines the constraints available to the user, the user has the option to extend the language by specifying new predicates. These are interpreted as relations between the elements of the domain of the respective interpretation structure. The language is still a subset of first-order logic; thus, its syntax can be chosen like the one of definite clauses, but with TFS's instead of first-order terms.

The specification language thus obtained allows the user to create partial specifications that can be incrementally extended, and to express controlled degrees of abstraction and precision. Although of considerable expressive power, this specification language is executable, but the control information is be abstracted; that is, formally the execution is nondeterministic, and there will be no explicit programming feature to express control. This has a good reason: control information coded in programs is specific to particular applications. For grammars for example, for the same underlying logical specification the control will be different in parsing or in generation, or even in different parsers (e.g., for indexing or for grammar checking). Thus, abstracting from control is important for gaining genericity: logical specifications apply to more problems than programs. The knowledge specification language is used in a first step in the generation of correct programs.

3 Automating the acquisition of linguistic descriptions

We assume that the acquisition of linguistic information will build upon the definition of broad linguistic

1. Which are sometimes only motivated by processing considerations.

categories formalized as the initial and secondary level of linguistic abstraction described above. In a Computer-Aided Linguistic Engineering framework, the acquisition of linguistic information is targeted towards the needs of specific applications: we also assume that the linguist uses for testing purposes a set of examples of the kind of text he describes (test case). These examples (the «corpus») can be constructed (as a way for example to specify the kind of dialogue envisaged for a natural language man-machine interface) or can come from existing texts, for example, existing technical documentation.

The acquisition of linguistic information consists in describing in full detail the set of linguistic phenomena occurring in the corpus as a specialization of linguistic axioms and principles. The acquisition is performed in two steps. First, the linguist uses corpus analysis tools to characterize the particularities of the sublanguage phenomena occurring in the corpus and to define the coverage (set of linguistic categories) that should be reached. Then, the linguist describes formally (i.e., using the specification language) in all details phenomena occurring in the corpus, using corpus analysis tools to find examples and to refine the categorization [Ananiadou 90, Tsujii et al. 90].

This approach to the acquisition of linguistic knowledge leads to the definition of a precise methodology (basic concepts and working procedures) supported by a specific set of software tools:

- *Concepts.* The basic concepts underlying this methodology are the notions of *sublanguage* and *coverage* [Grishman/Kittredge 86, Kittredge/Lehrberger 82, Grishman/Hirschman/Ngo 86]. Given a corpus, a linguist should be able to give a high level description of it in terms of its linguistic particularities which are not found in other kinds of texts, and in terms of the set of linguistic phenomena which are occurring in it: these concepts should be defined operationally to allow the linguist to apply them to actual texts.
- *Working procedure.* A working procedure defines the steps to be taken in the acquisition of linguistic knowledge, both in larger steps (characterization of the corpus, then acquisition) and in details such as how to document the phenomena described, to link a formal description to examples of the corpus, to check the consistency of the description with other parts of the specification, etc. It also gives examples of, e.g., how to define new lexical semantic classes using a cluster analysis tool (see below).
- *Software tools.* The concepts and working procedures are supported by a set of specialized linguistic software tools integrated in a Computer-Aided Linguistic Engineering workstation.

These software tools supporting the acquisition of linguistic knowledge should have the following functionalities:

- *Tagging.* A first set of functionalities is to tag a corpus using linguistic markers such as the category of word forms, their inflection, etc. Several levels of sophistication will be distinguished depending on the availability of the appropriate set of parameters: sets of closed categories, sets of word forms, sets of morphemes, definition of phrase boundaries, etc.
- *Text DBMS.* A tagged corpus is be loaded into a text DBMS for further exploitation, and accessed through a specialized linguistic interface (using a specialized query language).
- *Statistics and cluster analysis.* Two kinds of information can be extracted from a tagged corpus: statistical information and concordance and clustering information. Statistical and clustering analysis algorithms will be implemented and incorporated as functionalities of the linguistic interface of the text database.
- *Semantic editor.* The essential operation in linguistic acquisition is the creation of specializations of existing categories. A semantic editor takes into account the definition of existing classes and interactively guides the user in the creation of instances.

4 Automating the generation of NLP programs

In the development process sketched above (Section 1) the last step is the implementation of the system. Automatic generation of NLP software has been focused to the (crucial) domain of lexical resources (how to build generic resources and compilers that can extract electronic dictionaries from a lexical knowledge base for NLP systems) and to the domain of «reversible grammars»[1].

The process of transforming a specification into an efficient program is very similar to compilation. If the structure of a set of specification is stable, a compiler can be built to generate a program. This is the approach envisaged for lexical information[2]. Lexical

1. See for example the proceedings of the ACL Workshop on Reversible Grammars, Berkeley, June 1991.

information is here considered as «static» information: once the structure of the lexicon is defined, adding or removing an entry will not modify the compilation process. This is less true for grammatical information which defines how the basic linguistic buildings blocks, i.e., lexical entries, are combined into larger structures. Here, the needs may vary depending on the processing requirements of different NLP applications. For example, a grammar checker and an indexing system will most probably not use the same parsing scheme: they will treat differently errors and ambiguities. Thus, a general approach is needed.

Since the knowledge specification language is executable, this means that, to generate a program, there are two basic choices to be made: the selection of data structures and the selection of control structures. The nature and the complexity of these choices depend on the distance between the specification language and the targeted programming language.

As a programming language into which the specifications are derived, we envisage to use the Constraint Logic Programming (CLP) language LIFE developed at DEC-PRL [Aït-Kaci/Meyer 90, Aït-Kaci/Podelski 91]. The reason is that its formal foundation has parts in common with the Knowledge Specification Language; in particular, its basic data structures are also Typed Feature Structures, thus ensuring a basic level of compatibility between the two. Another reason is its descriptive power, its efficiency and its flexibility in execution («data-driven»): LIFE subsumes the two main programming paradigms (logic programming, as in PROLOG, and functional programming, as in LISP or ML). That is, a «logic» (or «functional») programmer may stick to his favorite programming style and still write code in LIFE.

Since the data model is the same, to generate an efficient program form a specification, the user will only have to select appropriate control structures. For example, to generate dictionaries for a parsing program, the only refinement the user will have to develop is to define an efficient indexing mechanism that allows a parser direct access to a lexical entry. In generating NLP parsers or NLP generators, the user will have to choose between a functional control structure (as in ML) or a relational control structure, as in PROLOG. For the latter, additional choices have to be made, such as the ordering of clauses, the introduction of cuts, etc. [Deville 90]. Research in computational linguistics has identified a few central computational concepts appropriate for NLP, among them regular grammars and regular transducers, augmented context-free grammars and tree transducers. In particular, augmented context-free grammars are the framework of the research in so-called «reversible grammars». This research can be used in the development of NLP processing schemes defined as annotations to the specification [Deville 90, Uszkoreit 91].

Assuming that a set of specifications is stable, it is possible to write a specialized compiler to generate a LIFE program for, e.g., parsing or generation. This compiler will embed the control choices that a designer of a parser makes when developing a parsing algorithm. This kind of generation has been shown practically feasible for lexical information, and research on «reversible grammars» has demonstrated the feasibility for grammatical information as well (see for example [Dymetman/Isabelle 88] who present a prototype of a machine translation system capable of translating in both directions using the same grammars and dictionaries).

However, we have also a long term more ambitious goal, which is to develop methods and tools for fully automating the generation of a program. Using these tools, the user will interactively guide the system in the generation of a program, experimenting with various choices and recording the design decisions for control to be used in a fully automatic step once the design is completed [Biggerstaff/Perlis 89].

5 Towards Computer-Aided Linguistic Engineering

We have outlined a framework for Computer-Aided Linguistic Engineering based on the concepts of reusability and automatic programming [Biggerstaff/Perlis 89], and showed that we have already all the basic ingredients (although at various degree of elaboration):

- a TFS based specification language [Emele/Zajac 90a, Emele/Zajac 90b];
- a TFS based constraint logic programming language [Aït-Kaci/Meyer 90, Aït-Kaci/Podelski 91];
- a methodology for the generation of NLP programs [Deville 90, Uszkoreit 91];
- a methodology for linguistic acquisition [Ananiadou 90, Tsujii et al. 90].

2. This is also the approach envisaged in the ESPRIT project Multilex and in the Eurotra-7 study.

To arrive a a fully detailed framework that could be implemented in a Computer-Aided Linguistic Engineering workstation, the major parts that need to be researched and developed are the elaboration of an annotation system to bridge the gap between the specification language and the programming language, and the development of adequate tools for the automated acquisition of linguistic knowledge. Of course, this approach has to be tested on a larger scale than what have been possible using the partial implementations available at present.

Part of the framework described in this paper is presently used in several on-going projects or proposed in several projects proposals. In the current projects, the primary domain of application of this framework is in the area of lexical representations (e.g., the MULTILEX ESPRIT project, the EUROLANG EUREKA project, the DELIS LRE proposal).

Acknowledgments. This paper was written while I was working in the Polygloss project at the IMS (University of Stuttgart). Many of the ideas presented in this paper have been discussed during the preparation of an ESPRIT project proposal on Computer-Aided Linguistic Engineering. I would especially like to thank Hassan Aït-Kaci, Gabriel Bès, Ulrich Heid, Andreas Podelski, and Hans Uszkoreit.

References

[Aït-Kaci 84] Hassan Aït-Kaci. *A Lattice Theoretic Approach to Computation based on a Calculus of Partially Ordered Types Structures*. Ph.D Dissertation, University of Pennsylvania.

[Aït-Kaci 86] Hassan Aït-Kaci. «An Algebraic Semantics Approach to the Effective Resolution of Type Equations». *Theoretical Computer Science 45*, 293-351.

[Aït-Kaci/Meyer 90] Hassan Aït-Kaci and Richard Meyer. «Wild_LIFE, a user manual». DEC-PRL Technical Note PRL-TN-1, Rueil-Malmaison, France, 1990.

[Aït-Kaci/Podelski 91] Hassan Aït-Kaci and Andreas Podelski. «Towards a meaning of LIFE». DEC-PRL Research Report PRL-RR-11, Rueil-Malmaison, France, June 1991.

[Ananiadou 90] S. Ananiadou. «The use of statistical techniques in identifying sublanguage patterns». Eurotra Research Report, 1990.

[Biggerstaff/Perlis 89] Ted J. Biggerstaff and Alan J. Perlis (eds). *Software Reusability*, 2 volumes. ACM Press – Addison-Wesley, 1989.

[Carpenter 90] Bob Carpenter. «Typed feature structures: inheritance, (in)equality and extensionality». Proc. of the *Workshop on Inheritance in Natural Language Processing*, Institute for Language Technology and AI, Tilburg University, Netherlands, August 1990.

[Deville 90] Yves Deville. *Logic programming. Systematic Program Development*. Addison-Wesley, 1990.

[Dymetman/Isabelle 88] Marc Dymetman and Pierre Isabelle. «Reversible logic grammars for machine translation». Proc. of the *2nd International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Language*, June 1988, Pittsburgh.

[Dymetman et al. 90] Marc Dymetman, Pierre Isabelle and François Perrault. «A symmetrical approach to parsing and generation». Proc. of the *13th International Conference on Computational Linguistics – COLING'90*, Helsinki, August 1990.

[Emele 1988] Martin Emele. «A typed feature structure unification-based approach to generation». Proc. of the *WGNLC of the IECE*, Oiso University, Japan, 1988.

[Emele 1991] Martin Emele. «Unification with lazy non-redundant copying». *29th Annual Meeting of the ACL*, Berkeley, June 1991.

[Emele/Zajac 90a] Martin Emele and Rémi Zajac. «A fixed-point semantics for feature type systems». Proc. of the *2nd Workshop on Conditional and Typed Rewriting Systems – CTRS'90*, Montréal, June 1990.

[Emele/Zajac 90b] Martin Emele and Rémi Zajac. «Typed Unification Grammars». Proc. of the *13th International Conference on Computational Linguistics – COLING'90*, Helsinki, August 1990.

[Emele et al. 90] Martin Emele, Ulrich Heid, Stefan Momma and Rémi Zajac. «Organizing linguistic knowledge for multilingual generation». Proc. of the *13th International Conference on Computational Linguistics – COLING'90*, Helsinki, August 1990.

[Franz 90] Alex Franz. «A parser for HPSG». CMU report CMU-LCL-90-3, Laboratory for Computational Linguistics, Carnegie Mellon University, July 1990.

[Grishman/Kittredge86] R. Grishman and R. Kittredge. *Analyzing Language in Restricted Domains*. Laurence Erlbaum, 1986.

[Grishman/Hirschman/Ngo 86] Hirschman L. Grishman, R. and T.N. Ngo. «Discovery procedures for

sublanguage selectional patterns: initial experiments». *Computational Linguistics*, 12(3):205–215, 1886.

[Kittredge/Lehrberger 82] R. Kittredge and J. Lehrberger. *Sublanguage: Studies of Language in Restricted Semantic Domains*. De Gruyter, 1982.

[Pollard 90] Carl Pollard. «Sorts in unification-based grammar and what they mean». In M. Pinkal and B. Gregor (eds.), *Unification in Natural Language Analysis*, MIT Press. (in press)

[Pollard/Moshier 90] Carl Pollard and Drew Moshier. «Unifying partial descriptions of sets». In P. Hanson (ed.) *Information, Language and Cognition*, Vancouver Studies in Cognitive Science 1, University of British Columbia Press, Vancouver. (in press)

[Pollard/Sag 87] Carl Pollard and Ivan A. Sag. *Information-Based Syntax and Semantics*. CSLI Lecture Notes 13, Chicago University Press, 1987.

[Pollard/Sag 91] Carl Pollard and Ivan A. Sag. *Agreement, Binding and Control. Information-Based Syntax and Semantics. Volume 2*. To appear.

[Smolka 88] Gert Smolka. «A Feature Logic with Subsorts». LILOG Report 33, IBM Deutschland GmbH, Stuttgart.

[Smolka 89] Gert Smolka. «Feature Constraint Logics for Unification Grammars». IWBS Report 93, IBM Deutschland GmbH, Stuttgart.

[Smolka/Aït-Kaci 88] Gert Smolka and Hassan Aït-Kaci. «Inheritance Hierarchies: Semantics and Unification». *J. Symbolic Computation 7*, 343-370.

[Strzalkowski 90] Tomek Strzalkowski. «How to invert a natural language parser into an efficient generator: an algorithm for logic grammars». Proc. of the *13th International Conference on Computational Linguistics – COLING'90*, August 1990, Helsinki.

[Tsujii et al. 90] Tsujii, J., Ananiadou S., Carroli J., and Phillips J.D. «Methodologies for the development of sublanguage MT systems». CCL Research Report CCL/90-10, UMIST, Manchester, 1990.

[Uszkoreit 91] Hans Uszkoreit. «Strategies for adding control information to declarative grammars». In *Proceedings of the 1991 Annual Meeting of the Association of Computational Linguistics*, Berkeley, 1991.

[Zajac 89] Rémi Zajac. «A transfer model using a typed feature structure rewriting system with inheritance». Proc. of the *27th Annual Meeting of the ACL*, 26–27 June 1989, Vancouver.

[Zajac 90a] Rémi Zajac. «A relational approach to translation». Proc. of the *3rd International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Language*, 11–13 June 1990, Austin.

[Zajac 90b] Rémi Zajac. «Semantics of typed feature structures». Presented at the *International Workshop on Constraint Based Formalisms for Natural Language Generation*, Bad Teinach, Germany, November 1990.