EAGLES

Expert Advisory Group on
Language Engineering Standards

EAGLES

Report of the EAGLES Workshop on the Connection between CLP and Linguistic Formalisms

EAGLES Document EAG–FWKS–3

Version of Feb, 1994

Contents

1 Editors

Martin Emele, Universität Stuttgart
Stephen Spackman, DFKI Saarbrücken.

2 Partecipants

Rolf Backofen (DFKI Saarbrücken, chair) DISCO
Luis Damas (Universidade do Porto) CLG
Martin Emele (Universität Stuttgart) TFS
Nelma Moreira (Universidade do Porto) CLG
Andreas Podelski (DEC Paris) LIFE
Roland Seiffert (IBM Germany) STUF
Gert Smolka (DFKI Saarbrücken) Oz
Stephen Spackman (DFKI Saarbrücken, editor) DISCO

3 Questions and Answers provided during the Workshop

3.1 What are the most interesting current developments and trends in CLP?

CLP and Concurrent Constraint Programming Languages extend logic programming languages like Prolog in two ways:

- unification of first-order terms is replaced by constraint solving and constraint solvers for different domains, thus providing more general constraints than those corresponding to equations between first-order sorts. This yields absolutely greater expressivity at the base level since constraints cannot generally be programmed directly in, say, Prolog.
- the sequential nature of control is replaced by a richer set of control primitives supporting concurrent programming.

These languages might be richer than predicate logic, with a number of higher-order facilities. Higher-orderness has rich applications in NLP; for instance, lambda-calculus is already widely used in natural language semantics.

Contrary to earlier CLP languages, new formalisms exist which provide efficient implementations of feature constraint normalization, directly within the language (e.g. Oz and LIFE).

Slogans for the richer control structures:

- Replace search as much as possible by constraint propagation.
- Disjunctions need not cause combinatorial explosion.
- Deep guards simplify programming.
- Generic form of negation (cf. paper on relative simplification by Smolka, Podelski, Ait-Kaci). Complete for feature terms, rational trees.

- programmable search (depth-first, breadth-first, iterative deepening) for the enumeration of solutions requires higher-orderness, if the solution is not to be interpretive (Oz).

Scalable systems can be constructed by providing high-performance compilation and program transformation techniques like partial evaluation and abstract interpretation. This was early demonstrated by Peter Van Roy in the Aquarius system, and he is now bringing that work to the LIFE system.

Integration of Object Orientation (OO). This provides both state encapsulation, an important software engineering tool, and inheritance, a data structuring tool that is already important in NLP.

Lazy sort unfolding is another kind of concurrent programming based on a different evalution model that makes use of memoization techniques to avoid replicated computations (LIFE, TDL, TFS).

3.2 Which of these developments are directly relevant to the open problems in NLP?

Higher-orderness and the richer control facilities are especially important for practical NLP systems. It is particularly desirable to be able to control search without sacrificing high-level and the essentially declarative description of data. Higher-order constructs are also used directly in current linguistic semantic formalisms.

If we distinguish between the inference engine and the linguistic workbench (i.e the grammar development environment) then richer control and compilation techniques are of the utmost importance for an efficient implementation of a particular grammar formalism. The object-oriented features might be useful for the implementation of the linguistic workbench itself.

Better software engineering facilities are essential to the NLP undertaking; even if most linguists do not consider themselves to be ”programmers”, they are still subject to the same complexity-control problems faced by the rest of the software industry. Indeed, the need for this kind of separation of concerns places even higher demands on modularity and abstraction facilities.

In sum: these are the days of real engineering, and if the open problems in NLP are construed to include what is necessary to its practical application, then everything becomes relevant. Indeed, such ”peripheral” issues as user interface facilities can make or break a system - though this does not necessarily imply that they must be provided as primitives by the system.

3.3 Are existing LP and CLP languages (e.g. LIFE and Oz) appropriate platforms for the implementation of advanced linguistic formalisms?

Yes, but no programming language will solve the problems for you. First, you have to understand the problem in order to come up with appropriate representations (data structures) and an operationalization of the problem. They are essentially platforms, not formalisms in the linguistic sense. First you make the linguistic formalism precise (semantically as well as syntactically), then you map it into the language. The result of this mapping is what must be judged.

Once again, higher order facilities and object orientation may be very good in implementation terms (facilitating maintainability, abstraction and modularity), whatever their relationship to linguistics per se. Good support for rapid prototyping may also be useful in the case where the linguistic formalism still retains some fluidity.

There is an additional question relating to scale of implementation: bulk data handling is only feasible with formalisms that can be compiled effectively. In principle CLP languages provide no barrier to this kind of processing, and indeed the LIFE compiler is planned to provide industrial-scale performance.

3.4 What are they lacking?

- stability?
- efficiency?
- availability?
- special data or constraint types such as sets, (linguistic) trees, etc.?
- appropriate control facilities for NLP?

Stability varies for the different prototype systems available. As criteria for choosing a particular language, one should check whether they provide a reasonable user-interface, a garbage collector, and an efficient implementation using high-performance compilation. The formalisms discussed have the property that they are amenable to compilation, but currently only attain research-system performance levels; this does not mean that performance is unacceptably slow, since efficiency is one of the research goals, but it would have to be addressed when planning systems for very large scale application.

Pre-release versions of LIFE and Oz are available by ftp, and these should be adequate to support prototyping. Full public release of both systems is anticipated in the near future.

Stability, efficiency and availability are particularly essential to many large-scale industrial applications. It may still be more cost-effective to develop for older formalisms (building new dusty decks, as it were!) than to delay until new formalisms stabilise completely. This may argue that separation of implementation from specification may be even more important than we thought - the ”operationalisation” of a new formalism must be made very transparent indeed, if development of sources and processing technology must proceed in parallel.

As for special data types, the case of sets stands out as a difficult (but perhaps not atypical) example. Sets as they appear in linguistic formalisms are often not used in a consistent way. Therefore, it has to be determined what precise properties are needed and how they can be encoded in an underlying system, rather than asking whether they are abstractly present or not. In general, however, a programming language is more likely to be able to support a given particular data type than a tight implementation of a given formalism. It may actually be a disadvantage if the programming system provides too many specialised data types at the expense of a flexible data representation toolkit.

The right control constructs for NLP are not yet known. But a linguistic formalism per se will abstract away from this issue rather than solving it; thus, as with data types, there is a slight advantage to be had from using a more language-based approach.

Work does still remain to be done on making programmable search facilities more semantically or statistically guided, and indeed on what exactly might be the right basic tools for doing this. The present facilities of Oz, at least, only allow the search topology (and not the search heuristics) to be programmed. In general, statistically guided search might be quite hard to add to extant systems.

3.5 Which of these things can be expected in the near future?

The compiler for LIFE will be available by June. A full release of Oz might be due by February and at the end of this year, by then both systems should be rather stable and efficient.

3.6 What do LP and CLP languages have that is still missing in NLP

- professional programming language design and implementation
- efficiency
- economic power behind their further development and maintenance
- additional data types
- better control methods
- better integration of non discrete methods (preferences, statistical choices)
- concurrency and parallelism (currently only concurrency)

The lack of comparable properties in NLP formalisms is due to the fact, that so far there have been no projects that were solely concerned with the development of an industrial strength representation formalism.

Strong research is still required in computational models, doing a deep analysis of what the problems are and how they could be solved by available implementation techniques.

Research on grammar formalisms should put more emphasis on computational and software engineering aspects.

In regard to economic power, it should be remembered that programming language development is an economic sink, rather than a source: while multipurpose languages can use funding derived from multiple areas of research, each such area sees it as a minor subsidiary field. Direct funding for these areas is currently quite rare. This means that the situation for languages may not be so very different from that in NLP, unless a project is very lucky indeed.

3.7 What are the main approaches for integrating CLP methodology into NLP (CLG, CUF/STUFF-III, TDL, TFS, ...)?

There seem to be at least three possible approaches:

- Integrating the CLP techniques into the NLP formalisms
- Map high-level linguistic descriptions into an existing CLP formalism
- Express grammars directly in a CLP formalism by operationalizing the given grammar specification by means of existing language facilities.

Each of these approaches has further variants depending on the amount of annotation (or transformation) applied to the original NLP formalism and the extent to which this annotation is considered to become ”part of” the resulting system.

3.8 What are the relevant differences among these approaches?

The first approach, if not undertaken as a serious programming language design and implementation enterprise, might cause duplicated effort and less than optimal solutions.

The second approach may have the advantage that the linguists don't have to learn a new programming language, but can continue to use their high-level description languages. But it is not clear how far such a compilation technique can be extended. It might be possible for a certain class of problems according to a specific grammar theory. But in general, it is a nontrivial problem to come up with appropariate representations and control structures that provide an efficient implementation of the phenomena described by a particular grammar. This might be mitigated somewhat by interactive compilation techniques, but this is a research area in its own right.

The third approach is the most direct one and requires a deep understanding of the operational reading of a given grammatical specification. It also requires profound knowledge of the programming language and their available control facilities. It renders it relatively difficult to maintain the distinction between grammar-writing and programming, which may be socially problematic.

3.9 Will the CL work on integrating CLP methods into NLP yield any results that are of real interest to the CLP community?

It seems to be one of the most interesting applications with nontrivial representations and rather complex constraint systems with many unsolved problems that might lead to new control structures.

3.10 Is NLP a good application domain? Is it a good test case? Is it a typical/diagnostic test case or are CLP languages designed for quite different tasks?

It is a good application domain and a good test case, cf. the answer to question 9.

3.11 What are good examples for existing cross-fertilisation between the communities?

The provision of theoretical foundations for feature terms was at originally undertaken independently in both communities and led later on to fruitful exchanges of common ideas.

Delay mechanism, lazy expansion, and higher order facilities all have experienced a certain measure of cross-fertilisation.

3.12 Who has benefited?

Both fields.

3.13 Which are good examples for existing collaboration between the communities?

This workshop, certainly! There are also many well-respected individuals who stand with one foot in each camp.

3.14 Who has benefited?

Both fields. Computer science is driven by applications, while NLP depends on application.

3.15 Why aren't there more or better examples?

Both disciplines stretch out over wide areas of theoretical and practical computer science and linguistics. Hence, it is not easy to keep track of current developments and new trends in both disciplines at the same time.

Furthermore, exchange on the intellectual level far outstrips that on the concrete, which is for the present well and good; but the time has truly come for increased concrete cooperation.

3.16 Should the exchange or collaboration between the communities be improved?

Yes.

3.17 If the answer to the last question is yes, how could this happen? How could and should this be supported by EAGLES, professional organisations, and funding bodies?

Development of NLP formalisms has mainly been accomplished as a by-product of funding linguistic research projects. It should be considered by the funding bodies to fund a common formalism project that combines expertise from practical CLP language design and implementation with expertise on requirements for linguistic representation formalisms. The time seems ripe for a project staffed by the right people and backed by adequate political will to make great advances.

3.18 Other issues discussed:

Smolka made a presentation of the CLP programming language Oz and techniques for encoding unification grammars in it directly. Contrasting Oz with LIFE clarified the distinction between to different approaches to control localisation: controlled type unfolding effectively localises it within predicates, while Oz localises it in independent control constructs.

Podelski gave a talk entitled ”An Order-Sorted Formal Theory of Unification”, and a detailed technical discussion ensued about the various computational mechanisms and their mathematical models of sort hierachies and sort unfolding.

Feature structures may be modelled as either open or closed. Unification-based formalisms differ on this point, as do CLP systems; Oz, for instance, has closed FFS, while LIFE has open. Closed feature structures may be slightly more efficient (though only by a constant factor, since open FFS can be implemented with extensible hash tables, as in LIFE), but there is some question as to how efficient it is to simulate open FFS with them (in the absence of global analysis) if that is what the formalism being implemented requires. On the other hand, functional adjunction (which is perhaps independently motivated in semantics) is available under the closed interpretation, and it is less clear what that would mean with open feature structures.