LPG: A generic, logic and functional programming language

D. Bert, P. Drabik, R. Echahed

Av. Félix Viallet, 46
F-38000 Grenoble (FRANCE)

O. Declerfayt, B. Demeuse, P-Y. Schobbens, F. Wautier

Unité d'Informatique, Université Catholique de Louvain
Place Sainte-Barbe, 2
B-1348 Louvain-La-Neuve (BELGIQUE)

LPG is a programming language designed to implement and experiment new concepts in the field of specification languages. In LPG, programs are theories in Horn clause logic with equality [3] [4]. Syntactically, a program is a theory presentation $TP=(S,\Sigma,\Pi,E,C)$ in which sorts (S), operators (Σ) and predicates (Π) can be declared; operators are specified by conditional equations (E), and predicates are specified by Horn clauses (C). Literals of a clause body may be equalities on terms. There are three kinds of presentations:

- properties (i.e. class of structures like group theory, ring theory, etc.);
- data types where the semantics is given by initial models;
- enrichments (i.e. hierarchical definitions of new operators and predicates on previously defined data types).

Moreover, data type and enrichment presentations may be parameterized by properties [1], thus providing generic data types and generic enrichments. It is also possible to relate theories between themselves by declaring theory morphisms. Those declarations are used by the instantiation mechanism, which is needed because of genericity.

Up to now, the following tools are available:

- An interpreter: this tool is designed to reduce ground terms to their normal form, with respect to the rewriting system deduced from the equations. This interpreter uses an abstract machine which deals with compiled programs. It also uses built-in procedures for operators over predefined data types like natural numbers, strings, etc. Taking advantage of the instantiation mechanism of generic operators, the interpreter is a good tool for testing specifications, prototyping, and functional programming.
- A logical evaluator (called "solver"): given a list of literals and/or equalities on terms, called a goal, the solver attempts to find the values of the variables which satisfy the goal. The underlying algorithm combines the resolution principle and conditional narrowing [2]. It has been shown sound and complete for a canonical rewrite rule system, but termination is not guaranteed for every goal.
- An interactive theorem prover: an interface with the OASIS system [5] has been developed in order to prove theorems. Implemented strategies are equational rewriting, case analysis, and proofs by induction. This system is also used to verify semantic correctness of specifications, like the validity of theory morphisms. (Multics version only.)
- A completer: this tool uses the Knuth-Bendix completion algorithm [7] to derive confluent systems of equational rewrite rules.
- A syntax-directed editor: this tool, based on [6], guides the user in the syntax of the LPG language (Sun version only), and performs context-dependent checks.

Two versions of the system are available, one written in Pl/1 and running under the Multics operating system, the other written in Ada and running on Sun workstations under Unix BSD 4.3. It is planned to port the Ada version to other machines, and to extend the language and its environment, in collaboration between LIFIA, UCL, Cert/DERI (Toulouse), CISI Ingénierie (Toulouse).

The presentation of LPG can be sketched as follows:

- Generic and functional programming: presentation of the instantiation mechanism, examples of interpretations.
- Generic and logic programming: examples of evaluations of goals.
- Demonstration of the auxiliary tools (syntax-directed editor, etc.)

References

[1] D. Bert, R. Echahed, Design and Implementation of a Generic, Logic and Functional Programming Language, Proc. of the European Symposium on Programming (ESOP'86), LNCS 213, Springer, 1986, pp. 119-132

[2] R. Echahed, Prédicats et sous-types en LPG. Réalisation de la E-unification, RR-IMAG-550-LIFIA-29, July 1985.

[3] J.A. Goguen, J. Meseguer, Introducing institutions, Proc. Logic for Programs, LNCS 164, Springer, 1984, pp. 115-125

[4] J.A. Goguen, J. Meseguer, Equality, Types, Modules and (Why not?) Generics for Logic Programming, Proc. Int. Conf. on Logic Programming, Uppsala, 1984, pp. 115-125 (also in J. Logic Programming, Vol.1 n.2, 1984, pp.179-210)

[5] E. Paul, Manuel OASIS, Note technique NT/PAA/CLC/LSC/959, CNET, 1985

[6] T. Teitelbaum, Th. Reps, Synthesizer Generator (2nd edition), Departement of Computer Science, Cornell University, Ithaca, 1987

[7] D.E. Knuth, P.B. Bendix, Simple word problem in universal algebra, in Computable problems in abstract algebra, J. Leech (ed.), Pergamon Press, 1970, pp. 263-297.