Logic Programming and Constraint Logic Programming

JACQUES COHEN

Brandeis University ⟨*jc@cs.brandeis.edu*⟩

Logic programming is a language paradigm based on logic, more specifically on resolution theorem proving in the predicate calculus as proposed in Robinson [1965]. Robinson had the foresight to distinguish the importance of two components in automatic theorem proving: a single inference rule called *resolution* and the testing for equality of trees called *unification.* Resolution is an inference step used to prove the validity of predicate calculus formulas expressed as clauses. In its simplest version: $P \vee Q$ and $-P \vee R$ imply $Q \vee R$, which is called the resolvant. Unification is the matching of terms used in a resolution step. It consists of testing the satisfiability of the equality of terms (i.e., labeled trees) whose leaves may contain variables. For example, the unification of the terms $p(X, q(Z, a))$ and $p(b, q(a, Y))$ succeeds, yielding the bindings $X = b$, $Z = a$, and $Y = a$.

Prolog, the main representative of LP, consists of a sequence of Horn clauses. A Horn clause is one containing (at most) one positive literal. The term *definite clause* is used to denote a clause with exactly one positive literal. Prolog programs can be viewed as a set of definite clauses in which the positive literal is the head of the rule and the negative literals constitute the body or tail of the rule. From a procedural point of view, a head corresponds to the definition of a Boolean function whose body consists of conjunctions of calls to the Boolean functions representing the tail [Kowalski 1979].

A quintessential example of a Prolog program is that of *append.* It consists of two Horn clauses specifying that list L3 is the concatenation of two lists, L1 and L2:

append (L1, L2, L3) :-L1 = *nil*, L2 = L3.
append (L1, L2, L3) :-L1 = *cons* (H, T),
  L3 = *cons* (H, Z), *append* (T, L2, Z).

The capital letters correspond to variables; *cons* is a term; the equal sign corresponds to a unification, a comma to a Boolean "and," and the ":-" to a (reverse) implication.

In a logical reading, a query Q is either deducible or not from a set of clauses P; if it is, then variables appearing in the query are specified by the resulting unifications. In a procedural interpretation, a query consists of an actual "call." For example, the query append (X1, X2, [*a*, *b*]) provides all lists X1 and X2 that, when appended, result in the list [*a, b*]. This is accomplished by a nondeterministic exploration of the search space.

A good reference on the theoretical foundations of LP is Lloyd [1987]. Sterling and Shapiro [1994] is a recommended introductory text.

Constraint logic programming (CLP) languages are LP languages in which unification is replaced by constraint solving in various domains. Constraints are special predicates whose satisfiability can be established for various domains using efficient algorithms (e.g., inequalities and disequalities). Unification can be viewed as a particular type of constraint that tests equality in the domain of trees. A recent survey on CLP is presented in Jaffar and Maher [1994].

Some of the current CLP languages and their domains are:

Prolog IV—trees, reals, intervals, rationals, finite domains (including Booleans), strings

CLP(R)—trees, floating-point arithmetic

CHIP—trees, floating-point arithmetic, finite domains

CLP(BNR)—trees, intervals

The languages considering intervals (defined by their lower and upper bounds) deal with numeric nonlinear constraints; symbolic linear constraints are handled by the first three languages (see Jaffar and Maher [1994]).

Another class of LP and CLP languages is made up of those that replace *don't-know nondeterminism* by *don't-care nondeterminism.* The first type of nondeterminism consists of specifying a number of options, each of which is considered either in parallel or sequentially (the latter using some type of search mechanism). In don't-care nondeterminism, only one option is considered and the others are disregarded. The LP or CLP languages using don't-care nondeterminism are called concurrent languages and were designed for the development of operating and real-time systems. Their major representatives appear to converge in a common concurrent language [Shapiro 1989].

APPLICATIONS

LP and CLP have proved successful in many applications. This success is due in great part to the conciseness that can be achieved by programs, the nondeterministic searches, and the LP relationship with logic; the latter instills a discipline for writing and debugging programs. Among the successful application areas are the following:

Symbolic Manipulation. Although Lisp and Prolog are currently the main languages in this area, it is quite probable that a CLP language may replace Prolog in the next few years. There is a close relationship between the aims of CLP and those of symbolic languages like Maple, Mathematica, and Macsyma.

Numerical Analysis and Operations Research. The proposed CLP languages allow their users to generate and refine sets that involve a large number of equations and inequations. The possibility of expressing inequations in a computer language has attracted the interest of specialists in operations research. Difficult problems in scheduling have been solved using CLP in finite domains.

Combinatorics. Nondeterministic languages like Prolog have been successful in the solution of combinatorial problems. The availability of constraints extends the scope of problems that can be expressed by CLP programs.

Artificial Intelligence Applications. Boolean constraints have been utilized in the design of expert systems. Constraints have also been used in natural language processing. The increased potential for inversibility makes CLP languages unique in programming certain applications.

Deductive Databases. These applications have attracted a considerable number of researchers [Minker 1987] and developers who are now extending the DB domains to include constraints.

Engineering Applications. The ease with which CLP can be used for generating large numbers of equations and inequations makes it useful in the solution of engineering problems. Ohm's and Kirchhoff's laws can readily be used to generate equations describing the behavior of electrical circuits.

IMPLEMENTATION

The implementation of Prolog requires the management of a stack, a heap, and a trail. The first two are well-known data structures used in processing most languages. In Prolog, the trail is used to implement nondeterminism and backtracking. The reader interested in a quick implementation of a nucleus of a

Prolog interpreter written in Pascal or C is referred to Cohen [1985]. Most of the existing Prolog on compilers generates a special code called Warren Abstract Machine (WAM), named after its developer. The WAM allows the compilation of efficient Prolog code [Ait-Kacl 1991]. There exists a substantial body of research in parallel implementations of LP [Kergommeaux and Codognet 1994].

As to CLP languages, variations of the simplex algorithm are used to test the satisfiability of linear equations and inequations. In the case of interval constraints, the operation of narrowing is applied repeatedly until convergence or failure is reached. The algorithms for testing satisfiability have to be incremental (i.e., a new constraint added to a satisfiable set of constraints involves only a small amount of recomputation). Simplification and elimination of redundant constraints are also desirable.

FINAL REMARKS

During the past twenty years, LP has followed a creative and productive course. It is not unusual for a fundamental scientific endeavor to branch out into many interesting subfields. An interesting aspect of these developments is that LP's original body of knowledge actually branched into subareas that joined previously existing research areas. For example, CLP is being merged with the area of constraint satisfaction problems; LP researchers are interested in modal, temporal, intuitionistic, and linear logic; relational database research now includes constraints; and operations research and CLP have found previously unexplored similarities.

The several subfields of LP now include research on CLP in various domains, typing, nonmonotonic reasoning, inductive LP, semantics, concurrency, nonstandard logic, abstract interpretation, partial evaluation, and blending with functional and object-oriented languages. It will not be surprising if each of these subfields becomes fairly independent of their LP roots and the various specialized groups organize autonomous journals and conferences. The available literature on LP is abundant and is likely to be followed by a plentiful number of publications on its autonomous subfields.

REFERENCES

AIT-KACL, H. 1991. *The WAM: A (Real) Tutorial.* MIT Press, Cambridge, MA.

COHEN, J. 1985. Describing Prolog by its interpretation and compilation. *Commun. ACM 28,* 12 (Dec.), 1311–1324.

JAFFAR, J. AND MAHER, M. 1994. Constraint logic programming, a survey. *J. Logic Program.* 503–581.

KERGOMMEAUX, J. C. AND CODOGNET, P. 1994. Parallel LP systems. *ACM Comput. Surv. 26,* 3.

KOWALSKI, R. A. 1979. Algorithm = logic + control. *Commun. ACM 22,* 7 (July), 424–436.

LLOYD, J. W. 1987. *Foundations of Logic Programming.* Springer-Verlag, New York.

MINKER, J. 1987. *Foundations of Deductive Databases and LP.* Morgan Kaufmann, San Mateo, CA.

ROBINSON, J. A. 1965. A machine-oriented logic based on the resolution principle. *J. ACM 12,* 1 (Jan), 23–41.

SHAPIRO, E. 1989. The family of concurrent LP languages. *ACM Compu. Surv. 21,* 3.

STERLING, L. AND SHAPIRO, E. 1994. *The Art of Prolog.* MIT Press, Cambridge, MA.