An Application of Constraint Propagation to Data-Flow Analysis

Roberto Bagnara, Roberto Giacobazzi, Giorgio Levi
Dipartimento di Informatica
Università di Pisa
Corso Italia 40, 56125 Pisa
{bagnara,giaco,levi}@di.unipi.it

Abstract

The optimized compilation of Constraint Logic Programming (CLP) languages can give rise to impressive performance improvements, even more impressive than the ones obtainable for the compilation of Prolog. On the other hand, the global analysis techniques needed to derive the necessary information can be significantly more complicated than in the case of Prolog. The original contribution of the present work is the integration of approximate inference techniques, well known in the field of artificial intelligence (AI), with an appropriate framework for the definition of non-standard semantics of CLP. This integration turns out to be particularly appropriate for the considered case of the abstract interpretation of CLP programs over numeric domains. One notable advantage of this approach is that it allows to close the often existing gap between the formalization of data-flow analysis in terms of abstract interpretation and the possibility of efficient implementations. Towards this aim we identified a class of approximate deduction techniques from AI and a semantic framework general enough to accommodate the corresponding approximate constraint systems.

AI topic: automated reasoning, arithmetic reasoning.
Domain area: data-flow analysis, constraint programming, optimized compilation of CLP programs.
Languages/Tools: CLP($\mathcal{R}$).
Status: implementation in progress.
Impact: the data-flow analysis we propose is: clean (semantics based), efficient (techniques from AI), and useful (a speed-up factor of 20 for the execution of CLP($\mathcal{R}$) programs has been obtained in preliminary tests [14]).

To appear in the proceedings of Ninth IEEE Conf. on AI Applications (CAIA'93).

1 Introduction

Constraint satisfaction problems are a very general and powerful formalism for knowledge representation, since many real situations can be conveniently described by a set of objects, together with some relationships among them. The recognition of this fact has lead to much work in the fields of AI and of Logic Programming.

In the last twenty years a number of AI researchers have explored the use of constraints to solve difficult problems [8, 20, 21, 22]. Most of the proposed systems were based on the technique of *constraint propagation* over a declarative structure called *constraint network* [7]. A constraint network consists of a number of nodes connected by constraints. A node represents an individual parameter of the problem at hand, while a constraint represents a relation among the values of the nodes it connects. The constraint propagation technique amounts to deducing information from a local group of constraints and nodes, recording this new information as a change in the network, possibly enabling further information to be inferred. These systems employing constraints have been generally developed in an *ad hoc* fashion in order to explore specific problems. Furthermore, as many problems of interest in AI are intractable in general, many systems needed to give up some degree of completeness, so to obtain useful, though approximate, answers by means of fast algorithms.

Over the last five years a new paradigm has emerged in the field of Logic Programming, and is generating much interest. Constraint Logic Programming (CLP) is a generalization of the pure logic programming paradigm, having similar model-theoretic, fixpoint and operational semantics [10]. One of the major aims of this scheme (class of languages) is to provide declarative programming with constraints, soundly based within a unified framework of formal semantics. For this reason a correct implementation of a CLP language needs a *complete* solver, that is a full decision procedure for the satisfiability of constraints in the language's domain(s). The indiscriminate use of such complete solvers in their full generality can lead to severe inefficiencies. In a paper devoted to compile-time optimizations for CLP($\mathcal{R}$), Jørgensen *et al.* show that, by reducing the number of times the system resorts to the full power of constraint solvers, dramatic performance improvements can be obtained [14]. These compile-time optimizations must be driven by information about the run-time behaviour of programs. An important technique to derive this information by data-flow analysis, is known under the name of *abstract interpretation* [4, 5, 19]. The essential idea is to "mimic" the program run-time behaviour by "executing" it, in a finite way, on an approximated (abstract) domain.

The original contribution of the present work is the integration of approximate inference techniques, well known in the field of artificial intelligence (AI), with an appropriate framework for the definition of non-standard semantics of CLP (see [9]). This integration turns out to be particularly appropriate for the considered case of the abstract interpretation of CLP programs over numeric domains. One notable advantage of this approach is that it allows to close the often existing gap between the formalization of data-flow analysis in terms of abstract interpretation and the possibility of efficient implementations.

The general technique we borrow from AI is *constraint propagation*, whose main advantages, for our purposes, are: simplicity, incrementality and good performance degradation when time limitations are imposed. Our study on the applications to data-flow analysis started with the definition of an abstract interpretation for CLP($\mathcal{FD}$)[1] [2, 3], following the framework proposed in [9]. The objective was to derive *spatial approximations* of the success set of program predicates. The concrete interpretation of a constraint, that is, a *shape* in k-dimensional space, was abstracted by an enclosing *bounding box*. Bounding boxes are rectangular regions with sides parallel to the axes. Thus, a bounding box is univocally identified by its projections (i.e. intervals) over the axes associated to the variables. In order to derive bounding boxes from sets of constraints we used the Waltz algorithm [15, 16, 22] applying *label refinement*, where the bounding box corresponding to a set of constraints is gradually restricted [7]. The approximations obtained by means of this analysis are important for achieving compile-time *domain reduction* (thus reducing the cost of backtracking search inside the constraint solver of a CLP($\mathcal{FD}$) system), and for the generation of (optimized) code where unnecessary choicepoint creations have been removed.

[1] CLP($\mathcal{FD}$) is an instance of the CLP scheme over *Finite numeric Domains*.

In this paper we present another application of constraint propagation (in its variant called *constraint inference*) to data-flow analysis. The objective is to detect a particular kind of constraints in CLP($\Re$) programs, that are of great interest for efficient code generation. In the next section we describe briefly the CLP($\Re$) language, that was presented as an example of the CLP scheme, and operates in the domain of real numbers [10, 13]. Section 3 explains what particular property of constraints we are interested in and why is it so important. In Section 4 we present our data-flow analysis for the optimized compilation of CLP($\mathcal{R}$)[2] programs. Section 5 contains some final remarks and conclusions.

2 The CLP($\Re$) Language

A CLP program is a finite set of *clauses* of the form

$$H :- c \,\Box\, B.$$

where H is an atom, c is a (possibly empty) conjunction of constraints, and B is a (possibly empty) sequence of atoms. Syntactically speaking, a *query* is a clause with no head, and is normally indicated with the notation

$$?- c \,\Box\, B.$$

CLP($\Re$) extends Prolog by adding constraint solving over the real numbers. In CLP($\Re$) constraints are equalities and inequalities over arithmetic expressions and equalities over symbolic expressions (terms). An example will serve to illustrate the flavour of the language.

```
mortgage(P,T,I,R,B) :-
    T = 1,
    B = P*(1+I/1200)-R □.
mortgage(P,T,I,R,B) :-
    T > 1,
    T1 = T-1,
    P >= 0,
    P1 = P*(1+I/1200)-R □
      mortgage(P1,T1,I,R,B).
```

This is a declarative specification of the relationship in

[2] We denote by CLP($\mathcal{R}$) the CLP($\Re$) (approximated) implementation described in [13].

a mortgage agreement between the principal (P), the duration of the loan in months (T), the annual interest rate (I), the monthly payment (R) and the outstanding balance (B). The program can be queried in several ways. For example the query

$$?-\ mortgage(P, 360, 12, 1025, 12625.9)$$

results in the *answer constraint* $P = 100000$, while the query

$$?-\ R > 0 \ \Box \ mortgage(P, 360, 12, R, B)$$

gives the "symbolic output"

$$R > 0 \wedge P = 0.0278 * B + 97.22 * R.$$

The operational semantics of CLP languages are given by query evaluation. We can describe it by means of a transition system whose configurations are just the queries described above. Terminal configurations are queries with an empty sequence of atoms. Let P be a CLP program and :: denote sequence concatenation. A query $?-\ C \ \Box \ A :: G$ is rewritten, in the context of P, as follows. A (renamed apart) clause $H :-\ c \ \Box \ B$ is selected from P to *resolve* against A. In case more than one clause is a candidate for selection, a *choicepoint* is created for later consideration. The constraint $C' \equiv C \wedge A = H \wedge c$ is then checked for satisfiability. If C' is satisfiable then the above query is rewritten to $?-\ C' \ \Box \ B :: G$. If C' is unsatisfiable execution backtracks to the most recently created choicepoint, and an alternative clause is chosen to resolve against A. A query is repeatedly rewritten until a terminal configuration is reached. The resulting constraint, simplified and *projected* onto the variables of the original query, is the *answer constraint* for the original query in the context of the program.

As far as the implementation is concerned, the CLP($\mathcal{R}$) interpreter [11, 13] consists of a *logical engine*, an *interface*, and a *constraint solver*. The engine performs traditional Prolog-like operation such as clause search and backtracking. The interface classifies and solves trivial constraints. The constraint solver handles arithmetic constraints. It uses Gaussian elimination to process equalities and an incremental version of the first phase of the Simplex algorithm to process inequalities.

3 Future Redundant Constraints

The Simplex algorithm is computationally expensive and the cost is proportional to the number of constraints and the number of variables it must take care of. The operational semantics given in the previous section indicates that the execution process of a CLP($\mathcal{R}$) program amounts to constraint accumulation and checking. This accumulation can give rise to conjunctions where some (maybe many) of the constraints are redundant, that is implied by other constraints in the conjunction. For instance the constrain $T > 1$ is redundant in the conjunction $T > 1 \wedge T_1 = 1 \wedge T_1 = T - 1$. Since the running time of the Simplex algorithm strongly depends on the number of constraints, redundant constraints cause a net loss in performance. This problem does not arise for equality constraints, as Gaussian elimination, like unification, has the property that redundant equations are automatically removed as a byproduct of keeping equalities in solved form. The Simplex algorithm does not have this property and a run-time mechanism for redundant constraint elimination is too much computationally expensive to be worth it.

For the above reasons in [14] the notion of *future redundant constraint* is introduced. Consider the *mortgage* program introduced in Section 2. The constraint $T > 1$ in the second clause is "future redundant". This means that after checking the constraint for satisfiability, adding or not adding the constraint to the *current constraint* (the one accumulated so far in the computation) will not affect the subsequent operational behaviour. In fact, at the next step in any reduction a stronger constraint will be encountered: either $T_1 = T - 1 \wedge T_1 = 1$ (if the first clause is selected) or $T_1 = T - 1 \wedge T_1 > 1$ (if the second clause is selected), so $T > 1$ will be redundant (implied by other constraints) in any future *current* constraint. Now, if T is a variable, not adding the future redundant constraint reduces the size of the current one, and consequently reduces the cost of the satisfiability check in any subsequent reduction step. The preliminary test results given in [14] suggest that this may give a dramatic speed up (i.e. a factor of 20 for non-trivial queries to the *mortgage* program).

The optimization associated with future redundant constraints [14] is obtained by slightly modifying the operational semantics so that future redundant constraints are just checked for satisfiability, but not added to the current constraint. This is handled by special instructions in the abstract machine for CLP($\mathcal{R}$) presented in [12]. In the next section we propose a data-flow analysis based on abstract interpretation aimed at the detection of future redundant constraints.

4 Detection of Future Redundant Constraints

Our analysis is based on constraint inference (a variant of constraint propagation) [7]. This technique, developed in the field of AI, has been applied to temporal and spatial reasoning [1, 17, 18]. Let us focus our attention to arithmetic domains (e.g. the reals or the floating-point numbers), where the constraints are binary relations over expressions. We abstract them by means of labelled digraphs. Nodes are called *quantities* and are labeled with the corresponding arithmetic formula (and possibly a variation interval). Arcs are labelled with relation symbols. More precisely, let

$$C = \{(e_{11} \bowtie_1 e_{12}), \ldots, (e_{n1} \bowtie_n e_{n2})\},$$

where the e_{ij} are terms of the constraint language (assume linear expressions for simplicity) and $\bowtie_i \in \{=, \neq, \leq, >, \geq, <\}$, denote a conjunction of constraints. Let $T = \{e_{ij} \mid i = 1, \ldots n,\ j = 1, 2\}$ be the set (modulo syntactical identity) of terms appearing in C, and let $k = |T|$, where $|\ .\ |$ denotes set cardinality. The abstraction of C is $G = (N, l_n, E, l_e)$, where:

$$\begin{array}{rcl} N & = & \{n_1, \ldots, n_k\} \\ l_n & \text{is} & \text{any bijection between } N \text{ and } T \\ E & = & \{(n1, n2) \mid \exists m \in \{1, \ldots n\} \\ & & \qquad (l_n(n_1) \bowtie_m l_n(n_2)) \in C\} \\ l_e\big((n_1, n_2)\big) & = & \{\bowtie_m \mid (l_n(n_1) \bowtie_m l_n(n_2)) \in C\} \end{array}$$

Disjunctions of constraints are represented by unions of digraphs, while conjunction is handled by connecting digraphs in the obvious way, merging the nodes having equal labels.

Let us consider a digraph representing a conjunction of constraints. We can *enrich* it by either adding new arcs to it or by adding (stronger) relations to the labels of existing arcs (these two operations collapse into just one if you consider that a missing arc is equivalent with one labelled with the always-true relation). Of course, since we are interested in approximate but *sound* deduction, we are interested only in correct enrichments. The addition of an arc (n_1, n_2) with label $\bowtie$ to the digraph abstracting C is correct iff $C \models_{\mathcal{R}} l_n(n_1) \bowtie l_n(n_2)$[3].

Sound enrichment of our graphs can be easily obtained by use of constraint inference techniques. In [17, 18] an arithmetic reasoning system, called the

[3]This means $\mathcal{R} \models \big(C \Rightarrow l_n(n_1) \bowtie l_n(n_2)\big)$, where $\mathcal{R}$ is the structure that *interprets* the constraints.

	$<$	$\leq$	$>$	$\geq$	$=$	$\neq$
$<$	$<$	$<$	??	??	$<$	??
$\leq$	$<$	$\leq$	??	??	$\leq$	??
$>$	??	??	$>$	$>$	$>$	??
$\geq$	??	??	$>$	$\geq$	$\geq$	??
$=$	$<$	$\leq$	$>$	$\geq$	$=$	$\neq$
$\neq$	??	??	??	??	$\neq$	??

Figure 1: Transitivity table for the graph search technique.

Quantity Lattice, is described. The Quantity Lattices supports, in a computationally efficient way, various qualitative and quantitative reasoning techniques. All these techniques are integrated, that is, inference made with one technique can trigger further inferences by the other ones. As a result the range of arithmetic inferences the system is able to perform is quite wide and suitable for our application. Given one of the above mentioned digraphs the Quantity Lattice can apply five different inference techniques:

1. Determining new relationships using graph search.
2. Determining new relationships using numeric constraint propagation.
3. Constraining the value of arithmetic expressions using interval arithmetic.
4. Constraining the value of arithmetic expressions using relational arithmetic.
5. Constraining the value of arithmetic expressions using constant elimination arithmetic.

Previous inference results are *cached* by adding arcs or refining their labels (and possibly by restricting the intervals associated with quantities). In our work we currently use only the inference techniques 1 (but computing the full transitive closure, as in [1]) and 4. The resulting system is still powerful enough to perform useful deductions for our purposes. Furthermore these techniques, as reported in [1, 18], can be implemented by efficient algorithms having linear complexity in the number of arcs. For example, technique 1 makes use of a slight modification of the standard breadth-first search. Relationships are found by using the simple transitivity table of Figure 1. The relational arithmetic technique is encoded by a certain number of axioms. These axioms capture some simple forms of reasoning, for example they allow the system to infer that $T > T - 1$. All the inferences made are recorded along with their justifications (i.e. sets of arcs). This is important, as we will see shortly.

Coming back to our application, namely data-flow analysis of CLP programs, consider the following example:

$$\begin{array}{lll} mg(T) & :- & T = 1. \\ mg(T) & :- & T > 1, T1 = T - 1 \Box mg(T1). \end{array}$$

The *behaviour* of the above program with respect to T is the same as the *mortgage* well known example.

In the model obtained by our abstract interpretation for the $mg/2$ predicate there are two four–nodes digraphs associated with the second clause (see Figure 2).

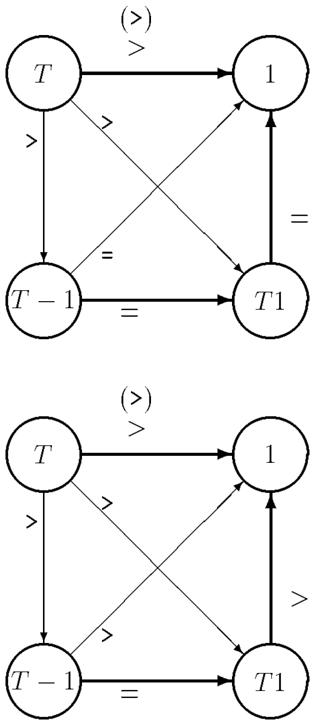

Figure 2: Portions of the two digraphs for the second clause of $mg/2$.Textual arcs have thick lines and are labelled with $=$ and $>$. Induced arcs have thin lines and are labelled with = and >.

The nodes (quantities) are labelled with the formulas T, $T - 1$, $T1$ and 1. Both the digraphs contain the *textual* arcs (that is arcs coming from the program's text) $T > 1$ and $T1 = T - 1$. The first digraph contains also the textual arc $T1 = 1$, and the *induced* arcs $T > T - 1$ (inferred by technique 4. above), $T > T1$ (inferred by 1. and justified by $\{T > T - 1, T1 = T - 1\}$), $T - 1 = 1$ (inferred by 1. and justified by $\{T1 = T - 1, T1 = 1\}$). Now by technique 1. we are able to parallel the textual arc $T > 1$ with an induced arc $T > 1$ with justification $\{T > T - 1, T - 1 = 1\}$. Similarly, considering the second digraph, we end up with an induced arc $T > 1$ with justification $\{T > T - 1, T - 1 > 1\}$. Since in both cases the induced arc $T > 1$ does not have the textual arc $T > 1$ into its justification, the textual arc $T > 1$ is future redundant. In general a textual arc (constraint) is future redundant if it can be doubled by an induced arc labelled with an equal or stronger relation and if this induced arc does not have the textual arc into its justification.

Future redundant constraint detection can be formalized in a slightly different way by considering deductions on interval arithmetics.

Let P be a CLP($\mathcal{R}$) program and ρ be an *upper closure operator* [4] on the domain of (linear) constraints on $\mathcal{R}$, ordered by entailment (logic implication). Let

$$C = p(t) :- c \wedge c' \Box B.$$

be a clause defining the predicate p in P. Consider the modified program

$$P' = (P \setminus \{C\}) \cup \{p(t) :- c' \Box B\}$$

and the query $G = ?- c_0 \Box p(x)$. If for any answer constraint c_p for G in P' we have $c_p \wedge c \not\equiv \mathit{False}$ and $\rho(c_p) \Rightarrow c$, then c is future redundant in C. To prove this claim we just note that by ρ-extensivity, for each constraint c we have $c \Rightarrow \rho(c)$, thus c does not contribute to any of the answer constraints because $c_p \Rightarrow c$.

A suitable choice of closure operators on the domain of linear constraints is provided by approximating each space region defined by a conjunction of linear constraints (i.e. a convex polytope) with a corresponding *bounding-box*. Similar techniques have been used [4] in for static array bound checking. This *quantitative* technique (contrasted to the *qualitative* one described above) allows for detection of future redundancy of constraints. To this end we can extend the work done in [2, 3] for CLP($\mathcal{FD}$), that is we can use constraint propagation with label refinement [7] to derive an enclosing bounding box for a given set of constraints. Even though this technique alone is not guaranteed to terminate when variables on the reals are concerned[5] [7], it is possible to provide external termination without prejudice for soundness. To force

[4]An *upper closure operator* on a poset $(A, \preceq)$ is a mapping $\rho: A \to A$ which is idempotent ($\rho(\rho(c)) = \rho(c)$), extensive ($c \preceq \rho(c)$), and monotonic.

[5]Or, better, it can converge very slowly when floating-point variables are concerned.

termination in the abstract answer constraint computation we can use *fixpoint-acceleration* techniques such as widening/narrowing proposed in [4, 6].

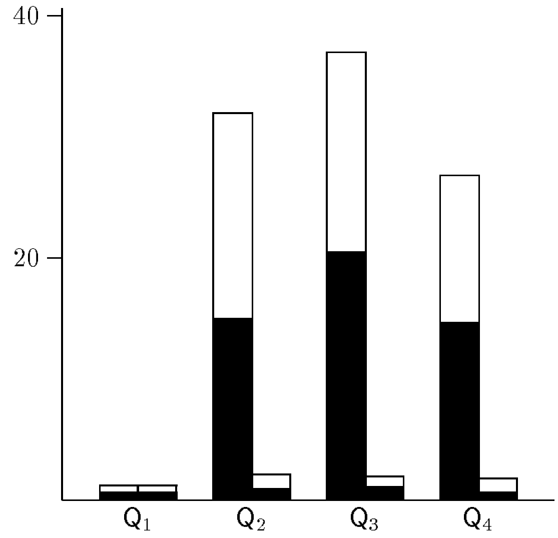

Figure 3: Run time improvement from future redundancy optimization, reproduced from [14]. The queries are:

- Q_1 :?− $mortgage(100000, 360, 12, 1025, B)$
- Q_2 :?− $mortgage(P, 360, 12, 1025, 12625.9)$
- Q_3 :?− $R > 0 \square mortgage(P, 360, 12, R, B)$
- Q_4 :?− $0 \leq B, B \leq 1030 \square$ $mortgage(100000, T, 12, 1030, B)$.

"The figure shows the effect of the future redundancy optimizations when applied to the four queries, in before/after pairs. Note that there is no change for Q_1 since the inequalities are ground. [...] All times are in CPU seconds. All measurements were obtained using a Sun 3/60 workstation with a 68881 floating point accelerator. To help factor out the cost of floating point arithmetic, measurements were taken using both the accelerator and software emulated floating point. The times using a floating point accelerator are shown as the [black] part of each bar, since they are uniformly shorter as expected" [14].

5 Conclusions

We have presented an application of constraint propagation to data-flow analysis of constraint logic programs. This application is of great importance for the optimized compilation of CLP programs, as its result are of immediate use, producing dramatic performance improvements such as the ones depicted in Figure 3.

This work, together with a previous one, confirms the starting hypothesis of our research: AI can give much to CLP in terms of techniques and knowledge, enabling (among other things) the generation of efficient code by means of data-flow analysis.

References

[1] J. F. Allen. Maintaining Knowledge About Temporal Intervals. *Commun. of the ACM*, 26(11):832–843, 1983.

[2] R. Bagnara. *Interpretazione Astratta di Linguaggi Logici con Vincoli su Domini Finiti.* M.Sc. thesis, Università di Pisa, July 1992. (In italian).

[3] R. Bagnara, R. Giacobazzi, and G. Levi. Static Analysis of CLP Programs over Numeric Domains. In *Actes Workshop on Static Analysis '92, Bordeaux*, 1992.

[4] P. Cousot and R. Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proc. Fourth ACM Symp. Principles of Programming Languages*, pages 238–252, 1977.

[5] P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Proc. Sixth ACM Symp. Principles of Programming Languages*, pages 269–282, 1979.

[6] P. Cousot and R. Cousot. Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. In M. Bruynooghe and M. Wirsing, editors, *Proc. of PLILP'92*, volume 631 of *Lecture Notes in Computer Science*, pages 269–295. Springer-Verlag, Berlin, 1992.

[7] E. Davis. Constraint Propagation with Interval Labels. *Artificial Intelligence*, 32:281–331, 1987.

[8] M. S. Fox. Constraint-Directed Search: A Case Study of Job-Shop Scheduling. Technical Report CMU-CS-83-161, Carnegie-Mellon University, 1983. Also published by Morgan Kaufmann, 1987.

[9] R. Giacobazzi, S. Debray, and G. Levi. A Generalized Semantics for Constraint Logic Programs. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, pages 581–591, 1992.

[10] J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In *Proc. Fourteenth Annual ACM Symp. on Principles of Programming Languages*, pages 111–119. ACM, 1987.

[11] J. Jaffar and S. Michaylov. Methodology and Implementation of a CLP System. In J.-L. Lassez, editor, *Proc. Fourth Int'l Conf. on Logic Programming*, pages 196–218. The MIT Press, Cambridge, Mass., 1987.

[12] J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. An Abstract Machine for CLP($\mathcal{R}$). To appear at SIGPLAN PLDI'92, November 1991.

[13] J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. The CLP($\mathcal{R}$) Language and System. *ACM Transactions on Programming Languages and Systems*, 1991. To appear.

[14] N. Jørgensen, K. Marriot, and S. Michaylov. Some Global Compile-Time Optimizations for CLP($\mathcal{R}$). In *Proc. 1991 Int'l Symposium on Logic Programming*, pages 420–434, 1991.

[15] A. K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8:99–118, 1977.

[16] A. K. Mackworth and E. C. Freuder. The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems. *Artificial Intelligence*, 25:65–74, 1985.

[17] R. Simmons. Representing and Reasoning About Change in Geologic Interpretation. Technical Report 749, MIT AI Laboratory, 1983.

[18] R. Simmons. Commonsense Arithmetic Reasoning. In *Proc. AAAI-86*, pages 118–124, 1986.

[19] M. Sintzoff. Calculating properties of programs by valuations on specific models. *SIGPLAN Notices*, 7(1):203–207, 1972.

[20] M. Stefik. Planning with Constraints (MOLGEN: Part 1). *Artificial Intelligence*, 16:111–139, 1981.

[21] G. J. Sussman and G. L. Steele. CONSTRAINTS: a Language for Expressing Almost Hierarchical Descriptions. *Artificial Intelligence*, 14(1):1–39, 1980.

[22] D. Waltz. Understanding Line Drawings of Scenes with Shadows. In P. Winston, editor, *The Psychology of Computer Vision*, chapter 2. McGraw-Hill, New York, 1975.