Implementing Constraint Solvers: Theory and Practice

Slim Abdennadher, Thom Frühwirth, Holger Meuss

appeared in *Proc. Forum de la Recherche en Informatique'96 (FRI'96), Tunis, Tunisia, July 1996*
http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-1996-14, July 1996

Implementing Constraint Solvers: Theory and Practice

Slim Abdennadher, Thom Frühwirth, Holger Meuss
Computer Science Department, University of Munich
Oettingenstr. 67, 80538 Munich, Germany
{Slim.Abdennadher,Thom.Fruehwirth,Holger.Meuss}@informatik.uni-muenchen.de
Phone: 0049 89 21782216 Fax: 0049 89 21782211

Abstract

Our research is based on Constraint Handling Rules (CHR), a powerful language for writing constraint solvers. We investigate confluence of CHR programs. This property guarantees that a CHR program will always compute the same result for a given set of constraints independent of which rules are applied. We give a decidable, sufficient and necessary syntactic condition for confluence.

Finally we present an application utilizing CHR offering rent advice: The city government of Munich regularly publishes a booklet called the "Mietspiegel" (MS). The MS basically contains a verbal description of an expert system. It allows to calculate the estimated fair rent of a flat. With our computerized version, "The Munich Rent Advisor", we extended the functionality and applicability of the MS so that the user need not answer all questions of the form. The key to computing with partial information was to use constraint technology.

1 Introduction

Logic Programming (LP) originates from the discovery that a fragment of predicate logic can be given a procedural interpretation thus forming the basis for programming languages like Prolog. The major advantage of such languages is that they can be declaratively interpreted in logic. Problems are solved by the built-in logic engine using chronological backtracking search. In *Constraint Solving* (CS), efficient special-purpose algorithms are employed to solve problems involving distinguished relations referred to as constraints. *Constraint Logic Programming* (CLP) [7] is a new class of programming languages combining the declarativity of LP with the efficiency of CS. However most CLP languages are not extensible: Constraint solving is usually hard-wired in a built-in constraint solver written in a low-level language. They do not allow for user-defined constraints.

Constraint handling rules (CHR) [4, 8] are a high-level language extension to write constraint systems. CHR support rapid prototyping of application-oriented constraint systems by providing executable specifications and efficient implementations due to an optimizing compiler. CHR allows for specialization, modification and combination of constraint solvers.

CHR is essentially a committed-choice language consisting of guarded rules that rewrite constraints into simpler ones until they are solved. There are two kinds of CHR: Simplification CHR rewrite constraints to simpler constraints while preserving logical equivalence (e.g. `X>Y,Y>X <=> false`). Propagation CHR add new constraints which are logically redundant but may cause further simplification (e.g. `X>Y,Y>Z ==> X>Z`). Repeatedly applying the rules incrementally solves constraints (e.g. `A>B,B>C,C>A` leads to `false`). With multiple heads and propagation rules, CHR provide two features which are essential for non-trivial constraint handling. Due to space limitations, we cannot give a formal account of syntax and semantics of CHR in this paper.

2 Confluence of CHR programs

In contrast to the family of the general-purpose concurrent constraint languages (CC) [11] and the ALPS [9] framework, CHR allow "multiple heads", i.e. conjunctions of atoms in the head of a rule. Multiple heads are a feature that is essential in solving conjunctions of constraints. With single-headed CHR rules alone, unsatisfiability of constraints could not always be detected (e.g `X<Y,Y<X`) and global constraint satisfaction could not be achieved.

Nondeterminacy in CHR arises when two or more rules can fire. It is obviously desirable that the result of a computation in a solver will always be the same, semantically and syntactically, no matter which CHR rules are applied. This property of constraint solvers will be called confluence.

We introduced in [2] a decidable, sufficient and necessary syntactic condition for confluence. Although confluence is in general undecidable, it turns out to be decidable for terminating programs (i.e. there are no infinite computation sequences): Let P be a terminating CHR program, then P is confluent iff P is locally confluent. The condition for local confluence can be expressed in terms of critical pairs as known from term rewrite systems [5]. We adopted and extended the terminology and techniques of conditional term rewriting systems (CTRS) [5]. A straightforward translation of results in the field of CTRS was not possible, because the CHR formalism gives rise to phenomena not appearing in CTRS. These include the existence of global knowledge (the built–in constraint store) and local variables.

To show a CHR program P is locally confluent, we need to know that for all constraints C, if C can be simplified in two different ways, then these two different results can subsequently be simplified into the same constraint. The possibility for a constraint to be simplified in two different ways is when the two heads of CHR rules overlap, i.e. they share at least one constraint. This is called an overlap, and the pair of constraints resulting form this is called *critical pair*. Critical pairs reflect choice points between different reduction sequences. So, we have the result that P is locally confluent if all its critical pairs can be joined (i.e. simplified into the same constraint).

Confluence turns out to be important with regard to both theoretical and practical aspects: We show in [2] that confluence implies correctness of a program. By correctness we mean that the declarative semantic of a CHR program is a consistent theory. Furthermore we show how to strengthen the declarative reading of a CHR program if it is confluent. A practical application of our definition of confluence lies in program analysis, where we can identify non–confluent parts of CHR programs by examining the critical pairs. Programs with non–confluent parts essentially represent an ill-defined constraint solving algorithm.

Our work extends previous approaches to the notion of determinacy in the field of CC languages: Maher investigates in [9] a class of flat committed choice logic languages (ALPS). He defines the class of deterministic ALPS programs as those programs whose guards are mutually exclusive. The class of deterministic ALPS programs is less expressive than confluent CHR programs. Saraswat defines for the CC framework a similar notion of determinacy [11], which is also more restrictive than confluence. The notion of deterministic programs is less expressive and too strict for the CHR formalism.

Our approach is orthogonal to the work in program analysis in [10] and [6], where a different, less rigid notion of confluence is defined: A CC program is confluent, if different process schedulings (i.e. different orderings of decisions at nondeterministic choice points) give rise to the same set of possible outcomes.

3 The Munich Rent Advisor

The "Mietspiegel"(MS), which is published by the government of Munich, allows to calculate the estimated fair rent of flats. The results of these calculations are typically used in civil court cases. The calculations are based on size, age and location of the flat and a series of detailed questions about the flat and the house it is in. Some of these questions are hard to answer. However to be able to calculate the rent estimate by hand, all questions must be answered.

Equipped with pencil, paper and calculator, one may need a weekend to figure out the estimated rent. Usually, the calculation is performed by hand in about half on hour by an expert from the City of Munich or from one of the renter's associations. The MS is derived from a statistical model compiled from sample data using statistical methods such as regression analysis. Due to the underlying statistical approach, there is the problem of inherent imprecision which is ignored in the paper version of the MS.

In just two man weeks we developed a computerized prototype called "The Munich Rent Advisor" [1, 3], MRA, that brought the calculation time down to few minutes. Using constraints the MRA can account for the statistical imprecision and also compute the estimated rent even in the presence of partial answers.

Our approach was to first implement the tables, rules and formulas of the "Mietspiegel" with high-level and declarative programming in ECL^{i}PS^e [4], ECRC's advanced constraint logic programming platform, as if the provided data was precise. Because of the declarativity of ECL^{i}PS^e it was easy to express the contents of the MS. Then we added constraints to capture the imprecision due to the statistical method and incompleteness in case the user gives no or partial answers. Finally, we considered the formulas of the rent calculation as constraints that refine the rent estimate by propagation from the constrained input variables. While it would have been difficult to model the required constraints in a given black-box constraints system, it was relatively straightforward using constraint handling rules (CHR). It was enough to modify an existing finite domains solver written in CHR that is part of the CHR ECL^{i}PS^e library.

The Munich Rent Advisor (MRA) is accessible through the internet service, and more specifically through World-Wide-Web (WWW). Using the internet, there is no need for the user to acquire specific software.

In about four man-weeks, we developed the form in the *Hyper Text Markup Language (HTML)*. The WWW-front-end is the graphical user-interface that should be handable without experienced computer-knowledge. For reason of simplicity we chose in a first step not to rely on advanced developments like applets in HotJava. To process the answers from the questionnaire and return its result, we wrote a simple special-purpose web-server directly in ECL^{i}PS^e using its C-sockets for internet communication.

References

[1] Abdennadher, S., Blenninger, P., and Frühwirth, T. Rent estimates with constraints over the internet. In *Vèmes Journées Francophones de Programmation en Logique et programmation par Contraintes* (1996).

[2] Abdennadher, S., Frühwirth, T., and Meuss, H. On confluence of constraint handling rules. In *Second International Conference on Principles and Practice of Constraint Programming (CP96)* (Cambridge, Massachusetts, USA, August 1996), Springer LNCS.

[3] Abdennadher, S., and Frühwirth, T. The Munich Rent Advisor. Submitted to 1st Workshop on Logic Programming Tools for internet Applications, 1996.

[4] Brisset, P., Frühwirth, T., Lim, P., Meier, M., Provost, T. L., Schimpf, J., and Wallace, M. *ECL^{i}PS^e 3.4 Extensions User Manual.* ECRC Munich Germany, July 1994.

[5] Dershowitz, N., Okada, N., and Sivakumar, G. Confluence of conditional rewrite systems. In *1st CTRS* (1988), LNCS 308, pp. 31–44.

[6] Falaschi, M., Gabbrielli, M., Marriott, K., and Palamidessi, C. Confluence in concurrent constraint programming. In *Proceedings of AMAST '95, LNCS 936* (1995), Alagar and Nivat, Eds., Springer.

[7] Frühwirth, T., Herold, A., Kchenhoff, V., Provost, T. L., Lim, P., Monfroy, E., and Wallace, M. *Constraint Logic Programming - An Informal Introduction.* Springer LNCS 636, September 1992, ch. Logic Programming in Action.

[8] Frühwirth, T. Constraint handling rules. In *Constraint Programming: Basics and Trends* (March 1995), A. Podelski, Ed., LNCS 910.

[9] Maher, M. J. Logic Semantics for a Class of Committed-Choice Programs. In *Fourth International Conference on Logic Programming* (Melbourne, Australia, May 1987), pp. 858–876.

[10] Marriott, K., and Odersky, M. A confluent calculus for concurrent constraint programming with guarded choice. In *Principles and Practice of Constraint Programming, Proceedings First International Conference, CP'95, Cassis, France* (Berlin, September 1995), U. M. F. Rossi, Ed., Springer, pp. 310–327.

[11] Saraswat, V. A. *Concurrent Constraint Programming.* MIT Press, Cambridge, 1993.