Available online at www.sciencedirect.com
ScienceDirect
Artificial Intelligence 171 (2007) 985–1010
Artificial Intelligence
www.elsevier.com/locate/artint

Exploiting functional dependencies in declarative problem specifications ☆

Toni Mancini *, Marco Cadoli

Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Salaria 113, I-00198 Roma, Italy

Received 4 June 2006; received in revised form 15 March 2007; accepted 30 April 2007

Available online 22 May 2007

Abstract

In this paper we tackle the issue of the automatic recognition of functional dependencies among guessed predicates in constraint problem specifications. Functional dependencies arise frequently in pure declarative specifications, because of the intermediate results that need to be computed in order to express some of the constraints, or due to precise modeling choices, e.g., to provide multiple viewpoints of the search space in order to increase constraint propagation. In either way, the recognition of dependencies greatly helps solvers, allowing them to avoid spending search on unfruitful branches, while maintaining the highest degree of declarativeness. By modeling constraint problem specifications as second-order formulae, we provide a characterization of functional dependencies in terms of semantic properties of first-order ones, and prove undecidability of the problem of their recognition. Despite such negative result, we advocate the (in many cases effective) possibility of using automated tools to mechanize this task. Additionally, we show how suitable search procedures can be automatically synthesized in order to exploit recognized dependencies. We present OPL examples of various problems, taken from bio-informatics, planning and resource allocation, and show how in many cases OPL greatly benefits from the addition of such search procedures. Moreover, we also give evidence that writing sophisticated ad-hoc search procedures that handle dependencies exploiting the peculiarities of the particular problem is a very difficult and error-prone task which in many cases does not seem to pay-off.

Keywords: Modeling; Reformulation; Second-order logic; Constraint satisfaction problems

1. Introduction

Declarative programming, and more specifically constraint programming, is becoming very attractive to solve different classes of problems, one of the main advantages of the approach being the fast prototyping and the high declarativeness exhibited by the problem models (also called "specifications"). Current systems for constraint solving (e.g., AMPL [19], OPL [43], DLV [31], SMODELS [36], and NP-SPEC [8]) allow the programmer to model her problem

☆ This paper is an extended and revised version of [M. Cadoli, T. Mancini, Exploiting functional dependencies in declarative problem specifications, in: Proceedings of the Ninth European Conference on Logics in Artificial Intelligence (JELIA 2004), Lecture Notes in Artificial Intelligence, vol. 3229, Lisbon, Portugal, Springer, 2004, pp. 628–640].

* Corresponding author.
E-mail addresses: tmancini@dis.uniroma1.it (T. Mancini), cadoli@dis.uniroma1.it (M. Cadoli).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2007.04.017

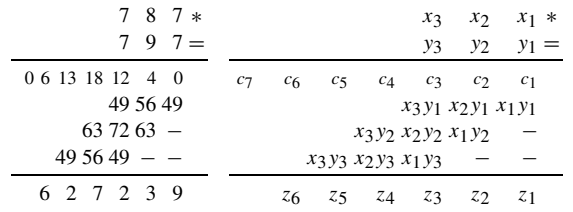

Fig. 1. Factoring instance 627239, $n = 6$, $b = 10$.

in a highly declarative way, supporting a neat separation of the specification from its instances. Such possibility allows the programmer to focus on structural and combinatorial aspects of the problem at hand before committing to actual input data, and hence permits problem modeling at a much higher level than that provided by the CSP framework.

However, it is well-known that the problem model obtained in this way is often not efficient, and much reasoning is required in order to reformulate it to speed-up the solving process. To this end, different approaches have been proposed in the literature, like symmetry detection and breaking (cf., e.g., [5,12]), the addition of implied constraints (cf., e.g., [41]), the deletion or abstraction of some of the constraints [3,16,20,23], and the use of redundant models, i.e., multiple viewpoints of the search space synchronized by channeling constraints, in order to increase constraint propagation [11,18,25,44]. However, many of these approaches either are designed for a specific constraint problem, or act at the instance level, and very little work has been done at the level of problem specification. Indeed, many of the properties of constraint problems amenable to optimizations strongly depend on the problem structure. Hence, their recognition naturally fits at the symbolic level of the specification, both from a methodological and an efficiency point of view.

Our research explicitly focuses on specification-level reasoning, with the goal of reformulating the declarative problem model submitted by the programmer into an equivalent one, more efficiently evaluable by solvers. In particular, in [6] we show how some of the constraints of a specification can be ignored in a first step, and then efficiently reinforced (i.e., without performing additional search, the so-called "safe delay" constraints), and provide a sufficient semantic criterion on the specification that can be used in order to recognize such constraints. Moreover, in [34] we tackle the issue of detecting structural (i.e., problem-dependent) symmetries, and breaking them by adding symmetry-breaking constraints to the problem specification.

In this paper we focus on another interesting property of constraint problems that is expected to benefit from reformulation, i.e., the functional dependencies that can hold among variables in declarative problem specifications. Informally, given a specification, a variable is said to be functional dependent on the others if, for every solution of every instance, its value is determined by those assigned to the others.

Functional dependencies are very common in problem specifications for different reasons: as an example, to allow the modeler to have multiple views of the search space, in order to be able to express the various constraints under the most convenient viewpoint, or to maintain aggregate or intermediate results needed by some of the constraints. The following two examples show the use of dependent variables under the two afore-mentioned circumstances.

Example 1 *(Factoring [30,40]).* This problem is a simplified version of a well-known problem in public-key cryptography. Given a (large) positive integer Z, which is known to be the product of two different *prime* numbers (different from 1), it aims at finding its factors X and Y.

An intuitive formulation of factoring as a constraint problem, in order to deal with arbitrarily large numbers, amounts to encode the combinatorial circuit of integer multiplication. In particular, assuming the input integer Z having n digits (in base b) $z_1, \ldots, z_n$, we consider $2n$ variables $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ one for each digit (in base b) of the two factors, X and Y (with z_1, x_1, and y_1 being the least significant digits for Z, X, and Y, respectively). The domain for all these variables is $[0, b-1]$. In order to maintain information about the carries, $n+1$ additional variables $c_1, \ldots, c_{n+1}$ must be considered, with domain $[0, (b-1)^2 n/b]$.

As for the constraints (cf. Fig. 1 for the intuition, where $x_4, x_5, x_6, y_4, y_5, y_6$ are equal to 0, and are omitted for readability), they are the following:[1]

[1] Since integer Z is *assumed* to be the product of two prime numbers, constraints ensuring that X and Y are prime are not needed.