Combining Relational Algebra, SQL, and Constraint Programming

Marco Cadoli and Toni Mancini

Dipartimento di Informatica e Sistemistica
Università di Roma “La Sapienza”
Via Salaria 113, 00198 Roma, ITALY
cadoli|tmancini@dis.uniroma1.it

Abstract. The goal of this paper is to provide a strong interaction between constraint programming and relational DBMSs. To this end we propose extensions of standard query languages such as relational algebra (RA) and SQL, by adding constraint solving capabilities to them. In particular, we propose non-deterministic extensions of both languages, which are specially suited for combinatorial problems. Non-determinism is introduced by means of a guessing operator, which declares a set of relations to have an arbitrary extension. This new operator results in languages with higher expressive power, able to express all problems in the complexity class NP. Some syntactical restrictions which make data complexity polynomial are shown. The effectiveness of both languages is demonstrated by means of several examples.

1 Introduction

The efficient solution of NP-hard combinatorial problems, such as resource allocation, scheduling, planning, etc. is crucial for many industrial applications, and it is often achieved by means of ad-hoc hand-written programs. Specialized programming languages [7,15] or libraries [10] for expressing constraints are commercially available. Data encoding the instance are either in text files in an ad-hoc format, or in standard relational DBs accessed through libraries callable from programming languages such as C++ (cf., e.g., [11]). In other words, there is not a strong integration between the data definition and the constraint programming languages.

The goal of this paper is to integrate constraint programming into relational database management systems (R-DBMSs); to this end we propose extensions of standard query languages such as relational algebra (RA) and SQL, by adding constraint solving capabilities to them.

In principle RA can be used as a language for testing constraints. As an example, given relations $A$ and $B$, testing whether all tuples in $A$ are contained in $B$ can be done by computing the relation $A - B$, and then checking its emptiness. Anyway, it must be noted that RA is unfeasible as a language for expressing NP-hard problems, since it is capable of expressing just a strict subset of the
polynomial-time queries (cf., e.g., [I]). As a consequence, an extension is indeed needed.

The proposed generalization of RA is named NP-Alg, and it is proven to be capable of expressing all problems in the complexity class NP. We focus on NP because this class contains the decisional version of most combinatorial problems of industrial relevance [S]. NP-Alg is RA plus a simple guessing operator, which declares a set of relations to have an arbitrary extension. Algebraic expressions are used to express constraints. Several interesting properties of NP-Alg are provided: its data complexity is shown to be NP-complete, and for each problem \( \xi \) in NP we prove that there is a fixed query that, when evaluated on a database representing the instance of \( \xi \), solves it. Combined complexity is also addressed.

Since NP-Alg expresses all problems in NP, an interesting question is whether a query corresponds to an NP-complete or to a polynomial problem. We give a partial answer to it, by exhibiting some syntactical restrictions of NP-Alg with polynomial-time data complexity.

In the same way, np-sql is the proposed non-deterministic extension of sql, the well-known language for querying relational databases [H], having the same expressive power of NP-Alg. We believe that writing an np-sql query for the solution of a combinatorial problem is only moderately more difficult than writing sql queries for a standard database application. The advantage of using np-sql is twofold: it is not necessary to learn a completely new language or methodology, and integration of the problem solver with the information system of the enterprise can be done very smoothly. The effectiveness of both NP-Alg and np-sql as constraint modeling languages is demonstrated by showing several queries which specify combinatorial problems.

2 NP-Alg: Syntax and Semantics

We refer to a standard definition of RA with the five operators \( \{\sigma, \pi, \times, -, \cup\} \) [I]. Other operators such as “\( \ast \)” and “\( / \)” can be defined as usual. Temporary relations such as \( T = algexpr(...) \) will be used to make expressions easier to read. As usual queries are defined as mappings which are partial recursive and generic, i.e., constants are uninterpreted.

Let \( D \) denote a finite relational database, \( edb(D) \) the set of its relations, and \( DOM \) the unary relation representing the set of all constants occurring in \( D \).

**Definition 1 (Syntax of NP-Alg).** An NP-Alg expression has two parts:

1. A set \( Q = \{Q_1^{(a_1)}, \ldots, Q_n^{(a_n)}\} \) of new relations of arbitrary arity, denoted as Guess \( Q_1^{(a_1)}, \ldots, Q_n^{(a_n)} \). Sets \( edb(D) \) and \( Q \) must be disjoint.
2. An ordinary expression \( exp \) of RA on the new database schema \( [edb(D), Q] \).

For simplicity, in this paper we focus on boolean queries. For this reason we restrict \( exp \) to be a relation which we call FAIL.

**Definition 2 (Semantics of NP-Alg).** The semantics of an NP-Alg expression is as follows: