Constraint Processing in Relational Database Systems: From Theory to Implementation

James J. Lu, Sebastien Siva, Ojas Parekh
Emory University
{jlu,ssiva,ojas}@emory.edu

George H. L. Fletcher
Eindhoven University of Technology
g.h.l.fletcher@tue.nl

Hantao Zhang
University of Iowa
hzhang@cs.uiowa.edu

ABSTRACT

Constraint satisfaction problems (CSP) are frequently solved over data residing in relational database systems. In such scenarios, the database is typically just used as a data storage back end. However, there exist important advantages, such as the wide availability of database practices and tools for modeling, to having database systems that are capable of natively modeling and solving CSPs. This paper introduces general concepts and techniques to extend a database system with constraint processing capabilities. Input CSPs are modeled via SQL, augmented with a non-deterministic guess operator as introduced by Cadoli and Mancini (TPLP 2007). Problems are represented with a combination of internal relations and parse trees, and are translated to a flexible intermediate problem representation that is subsequently translated into several common representations for SAT. Benchmarks with a prototype system show the feasibility of the approach and demonstrate the promise of a strong integration of CSP solvers and database systems.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: [Scheduling]

General Terms

Algorithms, Design, Languages

Keywords

CSP Tools, KR Languages, relational databases

1. INTRODUCTION

The need to model and solve constraints with data residing in relational databases is quite common, particularly for various forms of scheduling. A typical example is the flight scheduling problem where aircraft, crew members, and airports are all entities in a database. Moreover, the solution — the computed schedule — is usually also a part of the schema design of the database. Existing approaches to solving such *relational CSPs* involve programming outside the database. Many elegant systems, for example CLP(R), `AMPL`, `OPL`, `CModels` and `DLV`, have been developed to facilitate the specification, solving, and data access process.

Recently, Cadoli and Mancini have shown that extending the relational algebra with a simple non-deterministic guessing operator is sufficient to capture the class of NP [3]. The practical consequence of this is that SQL provides a viable language for modeling constraint satisfaction problems (CSP), thereby opening the possibility for a strong integration of constraint processing and relational database systems. There are important advantages to enable a user to specify, solve, and present CSP solutions *within the database environment*: 1) The user works within a single, presumably SQL based programming language. There is no need to switch programming environment and to confront the impedance mismatch associated with embedded database programming. 2) There is a strong integration between constraint modeling and data definition. A database design also serves as its constraint specification. 3) There are well-established practices and tools for relational databases that can further support the process of constraint modeling. Examples are Entity-Relationship Diagrams for specifying constraints, and Query-By-Examples for developing SQL queries. Recent work on conditional functional dependency [10] for data cleansing and visual modeling [13] further enrich the data constraint capabilities of modeling tools.[1]

These advantages offer significant potential for easing the development of data-centric AI applications and for making the power of constraint processing accessible to a large community of SQL users and practitioners. Indeed, industry has shown clear interest in this direction, as D-Wave systems have chosen SQL as the interface to its quantum-computing based discrete optimization solver.[2]

In this paper, we present general concepts and techniques for extending a relational database system to enable constraint specification and processing in SQL. The abstract language supported by our *SQL Constraint Data Engine* (SCDE) is expressive; it captures the class NP. While some interesting theoretical language issues exist, the current paper focuses on concrete engineering concerns. In particular,

[1]Note that the notion of constraints in CSP is much richer than traditional database integrity constraints such as functional and inclusion dependencies. Part of the advantage for integrating CSP solvers into the database engine is that techniques for specifying integrity constraints can also be applied naturally to model constraints in relational CSPs.

[2]http://www.dwavesys.com/; http://sql.dwavesys.com, viewed September 8, 2009.

we introduce abstractions, data structures and algorithms to facilitate the integration of open-source database systems and constraint solvers into an easy to use, interactive, robust, and efficient SQL constraint problem solver. Our specific contributions are as follows: 1) We introduce two languages, the *SCDE command language* (SCL) and the *extended Conjunctive Normal Form* (eCNF), that bookend the SCDE language sub-system. The SCL, adapted from SQL, provides the user/application interface, while eCNF, generalized from CNF, is the "intermediate code" for interfacing different constraint solvers; 2) We present algorithms for compiling input SCL constraints into eCNF; 3) We describe a prototype, its design, and report preliminary benchmarking results which demonstrate the feasibility of the design and approach.

Related Work. To our knowledge, two other efforts have been made in the immediate space. 1) D-Wave SQL, previously mentioned, is a proprietary system. We are in conversation with their engineers to explore possible collaborations, and to share implementation ideas and results.[3] 2) The simulator described in [3] was, at the time of publication, under development, and no details of the internal problem representation and translation were given. The simulator employs a local (i.e., incomplete) constraint solver. In contrast, the design of SCDE provides, via the eCNF, the flexibility to adapt to different solving paradigms and tools.

The broader body of related work include answer set programming [16, 20], constraint databases [15], model expansion [22], modeling languages [11], among many others. In these efforts, a tight integration with SQL databases is not the primary concern.

The current work significantly extends an earlier prototype system [23] which limits the number of constraint relations for a CSP to one, does not have the flexibility to work with different constraint solving paradigms, and handles optimization problems by brute force.

2. THE SCL AND THE ECNF

In a nutshell, SCDE first takes a CSP specification in SCL, translates it into eCNF, which is subsequently translated into the language of, and finally solved by, a particular constraint solver. In this section, we introduce SCL and eCNF. We first briefly discuss the abstract constraint language supported in SCDE.

2.1 Relational constraint satisfaction problems

Assume a fixed database schema $\beta = \{B_1, \ldots, B_n\}, n > 0$, i.e., a non-empty set of relation names, which we call our *base* relations. For a database instance $\mathcal{B}$ of β, let $DOM(\mathcal{B})$ denote the finite set of atoms occurring in $\mathcal{B}$.

A *constraint* over β is an expression of the form "$e = \emptyset$," where e is a first-order logic formula over β.[4] A constraint $e = \emptyset$ is said to *hold on instance* $\mathcal{B}$ of β if it is the case that $e(\mathcal{B}) = \emptyset$. A *relational constraint satisfaction problem* (RCSP) on β is a pair $\Pi = (\gamma,\varphi)$ such that 1) $\gamma = \{G_1, \ldots, G_m\}, m > 0$, is a finite set of relation names, of arity $g_1, \ldots, g_m$, respectively, such that $\beta \cap \gamma = \emptyset$, and 2) φ is a finite collection of constraints over $\beta \cup \gamma$.

A *solution* to Π, on a given instance $\mathcal{B}$ of β, is an assignment of an instance $\mathcal{G}_i \subseteq \underbrace{DOM(\mathcal{B}) \times \cdots \times DOM(\mathcal{B})}_{g_i \text{ times}}$ to each relation symbol G_i in γ, respectively, such that each constraint in φ holds on the database instance $\{\mathcal{G}_1, \ldots, \mathcal{G}_m\} \cup \mathcal{B}$. Note that γ are the *guess* relations of [3], which we call *constraint* relations (c-relations), to emphasize that the instances are determined by solving the constraints of φ.

For RCSP Π, let D_Π denote the following decision problem: *For a given database instance $\mathcal{B}$, does* Π *have a solution on $\mathcal{B}$?* We say that a family F of RCSPs is in complexity class C if, for each problem$\Pi \in F$, it is the case that D_Π is in C. We say that F is *complete* for C if F is in C and there is at least one problem$\Pi \in F$ such that D_Π is complete for C. This is the traditional notion of data complexity [17].

RCSPs are essentially the "relational algebra equations" of Biskup et al. [2]. It follows immediately that RCSP is NP-complete. Moreover, from [2], we have that RCSP is equivalent in expressive power to existential second-order logic. From Fagin's classic result on the expressive power of this logic [9, 17], it follows that RCSP captures NP (i.e., every problem of NP can be expressed in RCSP).

Of particular interest to our work in this paper is the RCSP fragment limited to constraints of the forms: $e = \emptyset$ (NEC) and $e \neq \emptyset$ (EC), in which the relation symbols in γ do not appear at quantifier depth greater than 1 in e. In SQL syntax, these are expressed as statements of the form

```
     NEC:  NOT EXISTS (SELECT ...)
     EC:   EXISTS (SELECT ...)
```

where c-relations may appear in the `FROM` clause of the top-level `SELECT` statement, but not in any nested queries. From our experience, NEC and EC, together with two more general forms of cardinality constraints can be used to model a large number of CSPs easily and naturally. Most typically, one designs constraints starting with EC, to state conditions that must hold for particular data instances. NEC can be used (with double negation) to state when some condition must hold *for all* data instances of a given relation.[5] As noted, conditional functional dependencies [10] such as [`A = 44`,`B`] $\rightarrow$ `C` (`A` and `B` functionally determines `C` whenever `A = 44`) may be stated easily with NEC as follows (we assume the schema `R(A,B,C)`):

```
NOT EXISTS (SELECT * FROM R r1, R r2
            WHERE  r1.A = r2.A AND r1.B = r2.B
            AND  r1.A = '44'  AND r1.C <> r2.C)
```

More example constraints will be given in the next section.

Aside from their wide applicability, it can be shown that 3SAT reduces to RCSP, restricted to NEC and EC. Hence NEC and EC form a reasonable starting point for the development of a general SQL-based constraint language.

2.2 The SCDE Command Language

Mostly overlooked in other CSP tools but central to the design of SCDE is enabling intuitive and frequent user interactions with metadata objects. Frequent user interaction is a basic premise of dynamic CSPs. Furthermore, there is growing recognition that the ability to incrementally fine-tune constraints is important to fully exploit the power of

[3]Private communications on July 2009.

[4]Note that constraints of the form $e_1 = e_2$ and $e_1 \neq \emptyset$ can be equivalently rewritten as constraints of the form $e = \emptyset$ [2]. Hence, for simplicity, we focus on "emptiness" constraints.

[5]We note that D-Wave SQL extends SQL with a new construct: `FORALL`. While such syntactic extensions are possible for SCDE, our concerns have been, as much as possible, with internal problem representations and the translation of the minimal subset of standard SQL.

1	`CREATE CONSTRAINT TABLE <Q> ( {<col> <type> FOREIGN KEY REFERENCES <ftable>(<fcol>),} {CONSTRAINT <c> <constraint>,} [OBJECTIVE MAX|MIN <obj_func>])`
2	`ALTER CONSTRAINT TABLE <Q> ( ADD CONSTRAINT <c> <constraint>);`
3	`ALTER CONSTRAINT TABLE <Q> ( DROP CONSTRAINT <c>);`
4	`TRACE CONSTRAINT TABLE <Q> CONSTRAINT <c>;`
5	`SOLVE TABLE <Q>;`
6	`COMMIT SOLUTION;`
7	`DROP CONSTRAINT TABLE <Q>;`

Table 1: Key Syntax Supported

NEC	`CHECK NOT EXISTS (SELECT ... )`
EC	`CHECK EXISTS (SELECT ... )`
NECC	`CHECK NOT EXISTS (SELECT ..... WHERE...[<|<>|>](SELECT COUNT(*) ...))` where c-tables do not occur in the outer select
ECC	`CHECK EXISTS (k [<|=|>] (SELECT COUNT(*) ...))` where `k` a non-negative integer constant

Table 2: Basic Constraint Syntax

intelligent tools in solving real-world problems [21]. Table 1 shows the basic SQL extensions to support the specification of RCSPs in SCDE. This syntax is aimed at flexible user interaction and align well with the standard SQL model for administering metadata objects.

The `CREATE CONSTRAINT TABLE` command declares the associated table (sometimes abbreviated c-table) to have an arbitrary extension. A set of constraints, written in usual SQL syntax over both ordinary and c-relations, may be included as part of the constraint relation specification. They may also be added and removed in subsequent statements, to this and other constraint relations. The syntax of NEC and EC (explained above), ECC and NECC are shown in Table 2. The latter two are forms of cardinality constraints that arise frequently in CSPs – expressions of the form$\Sigma_{i} l_i \geq k$ in pseudo-boolean constraint notation [6]. Other than the primary key, each column of the constraint table must be a foreign key to another table, which may be a base table or itself a constraint table. `SOLVE` triggers the compiling, solving, and populating of the c-table. `COMMIT SOLUTION` writes the current results (of all active c-tables) to disk. An optional objective function may be provided for constraint optimization problems.

Example 1. Consider the problem of creating a teaching schedule over the database schema described by the ER diagram in Figure 1. The c-relation is the shaded relationship `schedule`. The diagram is annotated with several constraints. First, the total participation (represented by

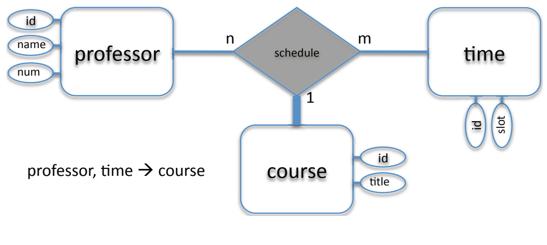

Figure 1: Course Scheduling ER Diagram

the thick line between `course` and `schedule`) along with the cardinality constraint indicates that each course must be taught exactly once. Second, the functional dependency `professor, time→course` specifies that no one may teach two courses at the same time. Other cardinality constraints of note: n represents the number of courses a professor must teach (specified by the `professor.num` field), while m represents the number of rooms available (or the maximum number of courses that can be taught during one time slot). These constraints are all naturally expressed as instances of NECC and NEC. For example, the required number of courses taught by each professor is given by

```
CHECK NOT EXISTS (SELECT p.id, p.num
  FROM professors p WHERE p.num <>
  (SELECT COUNT(*) FROM schedule s WHERE s.pid=p.id)))
```

Problem Decomposition. An alternative design to Figure 1 is to represent the ternary relationship, schedule, as two binary relations, `course-professor` and `course-time`. Such decomposition dramatically reduces the size and the search space in the underlying CSP. Any constraint that bridges the `professor` and `time` relations may be modified to reference both c-relations. Preliminary tests on several larger versions of the course scheduling problem (that include an entity for classroom) produce up to ten-fold performance gains. From these positive results, we are exploring heuristics to automatically discover useful decompositions.

Constraint Optimization. To allow instructors to indicate preferences for teaching times or courses, we may model the problem as a constraint optimization problem by adding the preference relation `pref(pid,cid,tid,score)`:

```
ALTER CONSTRAINT TABLE schedule ADD OBJECTIVE MAX pref
```

The attribute `score` is an integer reflecting the preference of professor for the given time or course. The schema of such a preference table may be any sub-schema of a c-table with an additional numeric score column. More general objective functions specified as SQL expressions are possible [3] and are under development.

2.3 eCNF

To enable SCDE to work with different constraint solvers, we introduce a flexible intermediate representation as the target of compiling SCL constraints. The idea of the representation, Extended Conjunctive Normal Form (eCNF), is to generalize CNF by allowing 1) conjunctions of literals to occur in place of literals and 2), a cardinality constraint. Each clause in eCNF has the form $F_1 F_2 \ldots F_n @k$ where k is a non-negative integer $0 \leq k \leq n$, @ is an integer comparison such as $\geq, \leq, =$, and each disjunct F_i is a conjunction of literals. Such a clause is true if $@k$ formulae among $F_1, ..., F_n$ are true. For example $(A \wedge B \wedge C)(\neg D)(\neg E \wedge F) \geq 2$ is true if at least two of $(A \wedge B \wedge C), (\neg D)$ and $(\neg E \wedge F)$ are true.

The eCNF notation succinctly captures some important concepts including cardinality constraints and grouping. It also captures constraints in disjunctive normal form (DNF), which is the natural interpretation of EC constraints when more than one copy of a constraint relation occurs (see discussion in Section 2.4). Lastly, eCNF facilitates easy translation to common constraint representations: CNF for SAT solvers, pseudo-boolean constraints [5, 8], MPS for integer programming solver.

Remarks. To translate DNF to CNF, the straightforward approach of distributing $\vee$ over $\wedge$ in general requires an

Algorithm NEC

```
1. Input: $C$ is an NEC $e = \emptyset$ and $\gamma$ is the set of c-relations in $e$.
2. $e' \leftarrow$ replace each $G \in \gamma$ with a relation of all possible tuples of $G$. .
3. $res \leftarrow$ compute the query $e'$
4. return $\{\{\neg v_G(t(G))\} \geq 1 \mid t \in res, G \in \gamma\}$
```

Algorithm ECC

```
1. Input: $C$ is an EC $e \geq k$ and $\gamma$ is the set of c-relations appearing in $e$.
2. $e' \leftarrow$ replace each $G \in \gamma$ with a relation of all possible tuples of $G$.
3. $res \leftarrow$ compute the query $e'$
4. return $\{\wedge_{G \in \gamma} v_G(t(G)) \mid t \in res\} \geq k$
```

Figure 2: Compiling NEC and ECC to eCNF

exponential blow-up of the formula. A more compact representation is via variable substitution: for each conjunct F in a conjunctive clause C, replace F in C by a new variable x_F and add the clauses for the formula $x_F \rightarrow F$. The number of new clauses introduced is $O(mn)$ where m is the size of the largest conjunct in C and n the number of conjuncts. This is a substantially less than the $O(m^n)$ clauses produced by distribution.

When cardinality constraints are involved (e.g., $F_1 \ldots F_n \leq 1$), variable substitutions as described above can be applied but with two modifications: a) the cardinality constraint is kept as part of clause C after the variable substitution; b) the relationship between x_F and F is $\rightarrow$, $\leftarrow$, or $\leftrightarrow$ depending whether the cardinality constraint is a $\geq$, $\leq$, or $=$, respectively. These may or may not need to be further translated depending on the type of solver employed.

2.4 Translation

Compiling SCL constraints into eCNF is very different from traditional piecemeal, syntax-directed compiling of programming languages. In a SQL parse tree, the semantic content of a single node type can vary greatly with respect to context. Moreover, the "virtual machines" of SQL and the target, whether SAT or LP, are quite different. Compared to other studies on translating high-level specification languages to SAT (e.g., [14, 4]), our main challenge lies in accommodating the flexibility of SQL in specifying constraints (which of course is also a strength of SQL). We approach the task one statement pattern at a time, based on the pattern and semantics of the four SCL constraints.

Given an NEC $e = \emptyset$, suppose γ is the set of c-relations appearing in e. In order for the constraint to hold, it must be that for any tuple t of the result of e, $t(G)$ must be absent for some $G \in \gamma$. Here, $t(G)$ denotes the projection of the tuple t on the attributes of G. For each $G \in \gamma$, let v_G denote a function that associates each possible tuple of G with a globally unique boolean variable. Then NEC may be expressed equivalently in eCNF: $\{\neg v_G(t(G)) | G \in \gamma\} \geq 1$ for each $t \in e$. This establishes the correctness of the algorithm for compiling an NEC into eCNF, given in Figure 2 (top). Step 2 is necessary to produce a query, e', that can be evaluated (since G is empty).

EC is an interesting dual to NEC, and is the special case of ECC $e \geq k$ where $k = 1$. In order for an ECC $e \geq k$ to hold, it must be that the cardinality of the result of e is greater than or equal to k. This is equivalent to the eCNF formula: $\{\wedge_{G \in \gamma} v_G(t(G)) | t \in e\}$, and we have the algorithm for compiling ECC shown in Figure 2 (bottom).

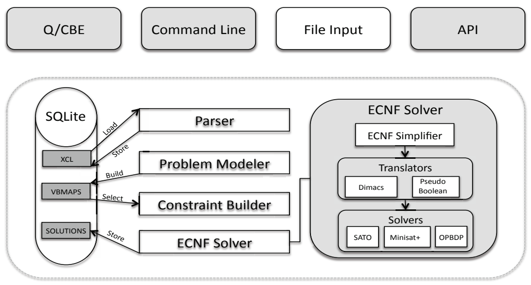

Figure 3: The SCDE Architecture

Cases involving other comparisons (e.g., $\leq$) are similarly translated. Finally, an NECC is first translated to a set S of ECC, one for each possible instantiation of the complement cardinality constraint. The translation is completed by applying the translation of ECC to each element of S.

Remarks. These algorithms may be seen as variations of the grounding algorithms employed by constraint systems such as answer set programming [12] and model expansion [22]. Key differences are the forms of the input constraints and the output to eCNF. Techniques for minimizing the encoding for both the intermediate and the target representation are part of our ongoing research, e.g., heuristics to avoid translation of the entire problem specification at once.

3. SYSTEM DESIGN, BENCHMARKS

The SCDE kernel consists of a collection of algorithms to parse input problems, to build an internal model of the problem, to query the database for building the eCNF representation, and to translate and solve the eCNF instance. We adopt Sqlite as the database engine (`www.sqlite.org`). Three constraint solvers, SATO [24], MiniSAT+ [8] and OPDPB [1] have been integrated. Figure 3 depicts the overall architecture of the system.

For each c-relation G, SCDE maintains a bookkeeping relation, $\texttt{vbmap}_G$, that implements the mapping v_G (Section 2.4) between possible tuples of G and logical variables of the eCNF. The schema of `vbmap` is that of G with an additional integer attribute, `vbid`. A meta-table that keeps track of all c-table names along with the size of each `vbmap` is used to standardize variable names apart when generating outputs in eCNF.

In some cases, constraints that belong to the same constraint pattern may be processed differently. As an example, Example 1 may include the constraint that no professor teaches courses outside of his/her area of expertise. These are commonly occurring constraints that translate to a set of unit negative clauses, but they are more efficiently processed as *filters* by excluding the invalid tuples (and their associated boolean variable) from the `vbmap`. In general, translation incurs a modest but predictable overhead.

Benchmarks. The current approach to encodings of constraints is basic, in both the eCNF and the target representation. Some simple strategies, such as variable substitution and filtering, have been implemented to automatically reduce the output size. Comparisons are made with

Instance	SCDE	L/S	Sol
nQ 20	1.018s	14.172s	Y
nQ 28	2m9.8s	> 15m	Y
BnQ (28.1449849431)	2m8.8s	15.9s	Y
BnQ (28.1449838621)	104.3s	> 10m	N
BnQ (30.1449838621)	41.58s	> 10m	Y
HC (1-.8-.11X.0)	3.768s	> 10m	Y
HC (1-.8-.11X.4)	3.529s	0.948s	Y
HC (2-.18-.11X.18)	19.5s	> 10m	Y
SG (w3g3s3)	0.13s	0.38s	Y
SG (w3g3s4)	1.83s	74.267s	N
SG (w5g5s3)	0.876s	4.281s	Y

Table 3: Comparison with Lparse/Smodels

Lparse/Smodels [20] on a number of problems. Table 3 highlights a few instances using MiniSat+ as the target solver for the eCNF encoding of SCDE. A complete set of experiments will be reported in the full paper.

The instances shown in Table 3 are for nQueens, Hamiltonian Cycle (HC), and the Social Golfer (SG). For nQueens, both the standard problem (nQ) and the Blocked n-Queens problem (BnQ) have been tested. For HC, we adopt the specification by Lifschitz [18]. The annotations inside parentheses are in reference to the instances found in the Asparagus repository.[6] Note that BnQ.30.1449838621 is modified from the 28x28 version by adding two unconstrained rows and columns. Experiments with time indicated by $> n$ means that the process was timed out after n minutes or seconds had elapsed.

The speed of SCDE is derived from its encoding and the quality of its solver, and SCDE owes much of its positive performance to the power of MiniSAT+ for solving psuedo-boolean constraints. In fact, testing of the basketball scheduling problem proposed in [19] revealed significant performance improvement (> 10x) of SCDE over an earlier implementation. More important than the absolute comparisons of time, however, Table 3 shows that despite its rudimentary encoding scheme, SCDE's performance already renders it a useful tool capable of solving a number of non-trivial problems.

4. CONCLUDING REMARKS

We have described an applied research effort to embed constraint processing capabilities inside relational database systems. Preliminary benchmarking demonstrates that a strong integration of CSP solvers and database systems is feasible, and initial comparison on a range of test problems shows that even a straightforward encoding, when used in conjunction with an appropriate constraint solver, yields very good performance.

Ongoing research includes an indepth study of encoding and solving strategies on CSPs involving multiple c-relations. Algorithms for automatic decomposition of complex CSPs into sub-CSPs will be designed and integrated. We are also working to extend the language sub-system to include a greater variety of common constraint patterns, and to provide a translation from visual models, such as Entity-Relationship diagrams, into SCL.

Acknowledgement. S. Siva's research is partially supported by the Emory University Research Committee.

[6]http://asparagus.cs.uni-potsdam.de/

5. REFERENCES

[1] P. Barth. A Davis-Putnam Based Enumeration Algorithm for Linear Pseudo-Boolean Optimization. Technical Report MPI-I-95-2-003.

[2] J. Biskup *et al.* Solving Equations in the Relational Algebra. *SIAM J. Comput.*, 33(5):1052–1066, 2004.

[3] M. Cadoli and T. Mancini. Combining Relational Algebra, SQL, Constraint Modeling, and Local Search. *TPLP*, 7(1-2):37–65, 2007.

[4] M. Cadoli and A. Schaerf. Compiling Problem Specification into SAT. *Artif. Intell.*, 162:89–120, 2005.

[5] D. Chai and A. Kuehlmann. A Fast Pseudo-Boolean Constraint Solver. In *Proc. of DAC*, pp. 830–835, 2003.

[6] H. E. Dixon *et al.* Generalizing Boolean Satisfiability I: Background and Survey of Existing Work. *J. Artificial Intelligence Research*, 21:193–243, 2004.

[7] N. Eén and N. Sörensson. An Extensible SAT-solver. In *SAT*, pp. 502–518, 2003.

[8] N. Eén and N. Sörensson. Translating Pseudo-Boolean Constraints into SAT. *JSAT*, 2(1-4):1–26, 2006.

[9] R. Fagin. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In *Complexity of Computation, SIAM-AMS Proc. 7*, pp. 43–73, 1974.

[10] W. Fan *et al.* Conditional Functional Dependencies for Capturing Data Inconsistencies. *ACM Trans. Database Syst.*, 33(2):1–48, 2008.

[11] A. M. Frisch *et al.* The Rules of Constraint Modeling. *IJCAI*, pp. 109–116, 2005.

[12] M. Gebser *et al.* GrinGo : A New Grounder for Answer Set Programming. In *LPNMR*, 2007.

[13] J. Gil *et al.* Advanced Visual Modeling: Beyond UML. In *ICSE*, pp. 697–698, Orlando 2002.

[14] E. Giunchiglia, Y. Lierler, and M. Maratea. Answer Set Programming Based on Propositional Satisfiability. *J. Autom. Reason.*, 36(4):345–377, 2006.

[15] D. Q. Goldin and P. C. Kanellakis. Constraint Query Algebras. *Constraints*, 1:45-83, 1996.

[16] N. Leone *et al.* The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Logic*, 7(3):499–562, 2006.

[17] L. Libkin. *Elements of Finite Model Theory.* Springer, Berlin, 2004.

[18] V. Lifschitz. What Is Answer Set Programming? In *AAAI*, pages 1594–1597, 2008.

[19] G. L. Nemhauser *et al.* Scheduling a Major College Basketball Conference. *Oper. Res.*, 46:1–8, 1998.

[20] I. Niemelä, P. Simons, and T. Syrjänen. Smodels: A System for Answer Set Programming. *CoRR*, cs.AI/0003033, 2000.

[21] B. O'Sullivan. Introduction to the Special Issue on User-Interaction in Constraint Satisfaction. *Constraints*, 9(4):239–240, 2004.

[22] M. Patterson *et al.* Grounding for Model Expansion in k-Guarded Formulas with Inductive Definitions. In *IJCAI*, pp. 161–166, 2007.

[23] S. Siva, J. J. Lu, and H. Zhang. A Case Study in Engineering SQL Constraint Database Systems. *ICLP*, pp. 774–778, Udine, Italy, 2008.

[24] H. Zhang. SATO: An Efficient Propositional Prover. *CADE*, pp. 272–275, 1997.