Transformation of SQL-based combinatorial optimization problems into Constraint problems

Genki Sakanashi, Masahiko Sakai

The 112th meeting of Special Interest Group on Fundamental Problems in Artificial Intelligence (SIG-FPAI), SIG-FPAI-B903-03, pp. 12–17, Mar 8–9, 2020.

Transformation of SQL-based combinatorial optimization problems into Constraint problems

Genki Sakanashi[1]*, Masahiko Sakai[1]

[1] Graduate School of Informatics, Nagoya University

Abstract: The authors recently proposed an SQL-based language CombSQL+ for specifying combinatorial optimization problems, and showed a solving method that uses an SMT solver. This method has an advantage that (i) problem constraints are written in ordinary SQL queries, and hence it enables an easy way to describe specification for SQL programmers, and (ii) problem instances can be supplied as ordinary database file. The generation method to produce SMT constraints are not only complex but also does not cover full-SQL query facilities.
To overcome the problem, this paper proposes a generation method of an ordinary SQL query, whose execution by an existing SQL-engine causes constraints for SMT solvers, from a specification. Here, a key technique is the use of user-definable functions, which may cause side-effects, allowed in the database system SQLite. We also show the correctness of the transformation.

1 Introduction

The *combinatorial optimization (CO)* is an important topic, which gives one of the best solutions for various problems. It is an intelligent work to describe a specification in a constraint modeling languages. In order to ease this, the authors have proposed an SQL-based language CombSQL+, its abstract semantics, and constructive semantics [4], whose benefits are listed as follows:

- Constraints and goal functions, which take the most important role, are written as ordinary SQL queries, hence everyone who knows SQL language can describe CO-problems in CombSQL+. Moreover, such SQL constraints can be debugged by an existing database engine if candidate solution tables are prepared as test data.
- It provides an easy separation of a problem description and its instances. Here the former is a specification, and the latter is stored in a database file.
- It is not necessary to introduce variables to specify a search space. Such variables are automatically introduce from the CombSQL+ description when producing a CP/SMT constraints.

In the language, the search space is given as a set of relational tables. The abstract semantics of SQL-operations is defined as the set extension. In [4], a constructive semantics is also given for implementation as an interpretation to quantifier free linear integer arithmetic (QFLIA) constraints, but it is not only complex but also does not cover full-SQL syntax.

*Present affiliation: Nomura Research Institute, Ltd.

This paper propose an SQL transformation so that execution of the resulted SQLs generate QFLIA constraints. A key idea is the use of user-definable functions that may cause side-effects allowed in the database system SQLite [2] (See Section 4). To this end, we implemented CombSQL+ system based on this SQL transformation technique, where we use pysmt [1] as an interface for SMT-solvers.

2 Preliminary

The set of Boolean values are $\mathbb{B} = \{\text{true}, \text{false}\}$, which may be interpreted as 1 and 0, respectively, when used in integer expressions. A *multiset* is a collection of elements, which allows multiplicity. This paper uses $\uplus$ and $\subseteq$ to represent the union and subset relation of multisets, respectively. For example, $\{\!\{1,1,2,3\}\!\} \uplus \{\!\{3,4\}\!\} = \{\!\{1,1,2,3,3,4\}\!\}$ and $\{\!\{1,1,2,3\}\!\} \subseteq \{\!\{1,1,2,3,3,4\}\!\}$.

A *valuation* α gives a value to each variable in its *domain* $\mathrm{Dom}(\alpha)$. We may write a valuation α by a set-like notation $\{x \mapsto \alpha(x) \mid x \in \mathrm{Dom}(\alpha)\}$. We use $\mathrm{Var}(\phi)$ for the set of all variables occurring in an object ϕ. If valuations α and β satisfy that $\alpha(x) = \beta(x)$ for common defined variables $x \in \mathrm{Dom}(\alpha) \cap \mathrm{Dom}(\beta)$, we say that they are *compatible*. The *union* $\alpha \cup \beta$ of compatible valuations α and β is well defined as $(\alpha \cup \beta)(x) = \alpha(x)$ if $x \in \mathrm{Dom}(\alpha)$; $(\alpha \cup \beta)(x) = \beta(x)$ otherwise. If compatible valuations α and β satisfies that $\mathrm{Dom}(\alpha) \supseteq \mathrm{Dom}(\beta)$, we say that α is an *extension* of β, denoted by $\alpha \succeq \beta$.

A relational database is a collection of *tables*, each

of which is a multiset of *records*. A record r is a set of values associated with a *column name*, which is represented like $\langle\langle c_1 = d_1, \ldots, c_n = d_n \rangle\rangle$ where each c_i is a column name and d_i is the value associated to c_i. The value d_i is also refereed as $r.c_i$. We use $[Q]_{\texttt{sel}}$ for the result of an execution of a query Q.

3 CombSQL+

3.1 Language

This subsection explains the language CombSQL+ [4] by a simple problem:

Table 1: R

col
1
2
3

Example 1 *Select lines, as much as possible, from the table, like R shown in Table 1, such that the total sum is three.*

A description in CombSQL+ consists of three steps.

Generation defines a search space consisting of a tuple of sets of tables in the following form:

```
CREATE SET name_set
   HAS (Q_1(t_1, ..., t_m), ..., Q_n(t_1, ..., t_m))
   FOR t_1 IN chs_qry, ..., t_m IN chs_qry
```

where Q_i is an ordinary SQL, and *chs_qry* is called a *choosing query*, which may contain `CHOOSE` and `SUBTABLE OF` operations. When $m = n = 1$, we have an abbreviation (See [4] for detail):

```
CREATE SET name_set AS chs_qry.
```

Filtering describes a query $Q(t_1, \ldots, t_n)$ that returns *true* when given tables satisfy the problem constraint in the following form:

```
CREATE SET name_set HAS (t_1, ..., t_n) IN name_set
   SUCH THAT Q(t_1, ..., t_n)
```

Selection orders to solve an optimal solution with respect to a given goal function in the following form:

```
CREATE TABLE (name_1, ..., name_n) AS (t_1, ..., t_n)
    IN name_set MAXIMIZING f(t_1, ..., t_n)
```

Example 2 *The problem in Example 1 is specified as follows (See also Tables 1 and 2):* [1]

```
CREATE SET SearchSp AS
   SUBTABLE OF (SELECT * FROM R)
CREATE SET Sol HAS t IN SearchSp
   SUCH THAT (SELECT SUM(col) FROM t) = 3
CREATE TABLE sol AS t IN Sol
   MAXIMIZING (SELECT COUNT(*) FROM t)
```

SQL queries are naturally extended to those on set of tables. We use $Q(R_1, \ldots, R_n)$ for an SQL Q by explicitly presenting tables R_i. In the following definition, we give a semantics of an extended query $Q(\mathcal{R}_1, \ldots, \mathcal{R}_n)$ for sets $\mathcal{R}_i$'s of tables. This notion gives an important role in implementation.

[1] The `SUBTABLE OF` structure is not implemented due to a technical reason in the current system. Nevertheless, we use this specification for simplicity.

Table 2: Tables *SearchSp*, *Sol*, and *sol*

(a) *SearchSp*

col

,

col
1

,

col
2

, ...,

col
1
2
3

(b) *Sol*

col
1
2

,

col
3

(c) *sol*

col
1
2

Table 3: A constraint table $\langle R^+, \psi_1 \rangle$

R^+	
col	`ext`
1	x_1
2	x_2
3	x_3

$\psi_1 =$
$(\bigwedge_{1 \le i \le 3}(x_i = 0) \vee (x_i = 1))$
$\wedge ((1x_1 + 2x_2 + 3x_3) = 3)$

Definition 3 *An execution of an extended query $Q(\mathcal{R}_1, \ldots, \mathcal{R}_n)$ returns a set of tables $\mathcal{S}$ satisfying that $\mathcal{S} = \{[Q(R_1, \ldots, R_n)]_{\texttt{sel}} \mid R_i \in \mathcal{R}_i\}$.*

3.2 Constraint tables

We introduced a constrained table[2] in [4] for representing a set of tables.

Definition 4 • *A* constrained table *is a pair $\langle R^+, \psi \rangle$ of a relational table R^+ and a constraint ψ over those variables. Here R^+ may contain variables as data, and must have a special Boolean column* `ext`.

- *The table $\alpha(R^+)$ obtained from a constrained table R^+ and a valuation α is defined as follows:*
  $\alpha(R^+)$
  $= \{\!\{\alpha(r) \mid (r \cup \langle\langle \texttt{ext} = v \rangle\rangle) \in R^+, \alpha(v) = \text{true}\}\!\}$.
- *R^+ and a constraint ψ represent the following set $\|R^+\|_\psi$ of tables:*
  $\|R^+\|_\psi = \{\alpha(R^+) \mid \alpha \models \psi\}$
  Here, $\alpha \models \psi$ denotes that α satisfies ψ. An abbreviation of $\|R^+\|_{\text{true}}$ is $\|R^+\|$.
- *$\langle R^+, \psi \rangle$ is a represention of a set $\|R^+\|_\psi$ of tables.*

Note that an ordinary relational table can be seen as a constrained one by regarding that the column `ext` with true data is abbreviated.

Example 5 *The constrained table in Table 3 represents the set Sol in Table 2 (b).*

[2] The notion of constrained tables is different from constraint database [3]. The latter represents a relational table containing possibly infinite records.

3.3 Solving combinatorial problems

We can solve problems in a following way, if we obtain an implementation of the extended SQL on constrained tables.

Algorithm 6 ([4]) *Input: a CombSQL+ problem description (as shown below), and tables that store a problem instance.*

`CREATE SET` *SearchSp* `AS` Q_0
`CREATE SET` *Sol* `HAS` *t* `IN` *SearchSp* `AS` $Q(t)$
`CREATE TABLE` *sol* `AS` *t* `IN` *Sol* `MAXIMIZING` $f(t)$

Output: a solution table $\alpha(R^+)$

Procedure:

1. *Generation: SearchSp is a constraint table* $\langle R^+, \psi \rangle$ *(detailed explanation is omitted).*
2. *Filtering: supposing* $Q(\langle R^+, \psi \rangle) = \langle x, \eta \rangle$*, Sol is a constraint table* $\langle R^+, x \wedge \eta \rangle$*.*
3. *Selection: supposing* $f(\langle R^+, x \wedge \eta \rangle) = \langle n, \eta' \rangle$*, a valuation is calculated by a SMT solver from constraint* η' *with optimization goal* n*.*
   The procedure output a table $\alpha(R+)$ *for a valuation* α *obtained by the solver.*

Theorem 7 ([4]) *A table* $\alpha(R^+)$ *obtained by Algorithm 6 is an optimal solution.*

4 Basic idea

We have shown a constructive semantics of the extended queries in [4], which is not only complex but also has a restriction that e in `SELECT` e `FROM ...` must not contain recursive query. This section illustrates the outline of a simpler method for the extended queries based on an SQL transformation, where an execution of the resulted (ordinary) query produces SMT constraints, which can be send to SMT solvers.

A key idea is the use of functions like those shown in Figure 1, where these functions are regarded as user defined ones for ordinary SQL engines.

Given an extended query $Q(\langle R^+, \psi \rangle)$, we generate an ordinary query $Q'(R^+)$ satisfying the following property:

Property 8 *A query* Q' implements *a query* Q *for constrained table* $\langle R^+, \psi \rangle$ *if*

$[Q'(R^+)]_{sel} = S^+$ *with* η
implies $\|S^+\|_{\psi \wedge \eta} = \{[Q(R)]_{sel} \mid R \in \|R^+\|_\psi\}$.

```
1  | def SUM'((x1,b1),...,(xn,bn))
2  |   y = new_fresh_variable()
3  |   send("y = x1*b1 + ... + xn*bn")
4  |   return y
5  | end
6  | def EQ'((x1, x2)
7  |   y = new_fresh_variable()
8  |   send("y <=> (x1 = x2)")
9  |   return y
10 | end
```

Figure 1: Pseudo-codes of functions used in the obtained queries

Here, we assume that variables in R^+ are represented as strings, and η is stored by side effects of the functions like in Figure 1.

Example 9 *An extended query*

$Q =$ `(SELECT SUM(col) FROM` $\langle R^+, \text{true} \rangle$`) = 3`

is transformed to

$Q' =$ `EQ'((SELECT SUM'(col,ext) FROM` R^+`), 3)`,

where `SUM'` *and* `EQ'` *are functions shown in Figure 1 as pseudo code. The evaluation of the query* Q' *for* R^+ *in Table 3 produces* z *while sending* $y = 1x_1 + 2x_2 + 3x_3 \wedge (z \Leftrightarrow (y = 3))$ *to an SMT-solver. Remark that Boolean values here is interpreted as 0 or 1.*

This approach is simple, and some of SQL operations cannot be processed without a drastic change on queries. For example, consider a constraint table R_2^+ in Figure 4. Suppose processing the following extended query:

Table 4: R_2^+

col	`ext`
y_1	true
y_2	true
y_3	true

`SELECT COUNT(col) FROM` $\langle R^+, \text{true} \rangle$ `GROUP BY col`

Transformation seems very difficult, because the target column of `GROUP BY` operation is given as variables whose values are not fixed. Therefore we eliminate such difficult operations, as presented in the next section, before transformation.

5 Eliminating some structures

This section defines a *simple* subclass of SQL from the reason explained in the bottom of Section 4. Each `SELECT` sentence in simple queries must satisfy the following conditions:

- No `GROUP BY`, `HAVING`, nor `DISTINCT`.
- No set operations except for `UNION ALL`.
- No `JOIN` except for `CROSS JOIN`.
- At most one `CROSS JOIN`.
- If containing `CROSS JOIN` then no operations in result columns and also no `WHERE`.
- No operation in `WHERE`.
- Every column/table is named by `AS`.
- No `AVG`.
- No operation in arguments of aggregation functions.

We use Q for selection queries that return tables, c for column names, e for value expressions or selection queries that return values, f for non-aggregate functions, and a for aggregate functions. Now, we show the syntax and semantics of simple queries as follows Note that the semantics are given allowing side effects in functions.

Definition 10 *We use* $[Q]_{\texttt{sel}} = R$ with ψ *to present that an execution of* Q *returns table* R *with constraint* ψ *as side effect. Similarly,* $[e]^r_{\texttt{exp}} = x$ with ψ *presents that an evaluation of* e *for a record* r *results in* x *with constraint* ψ *as side effect. These are defined as follows:*

- $[table]_{sel} = table$ *with* true.
- $[\textit{SELECT}\ e_1\ \textit{AS}\ c_1, \ldots, e_n\ \textit{AS}\ c_n\ \textit{FROM}\ Q]_{sel}$ $= \{\!\{\langle\!\langle c_1 = v^r_1, \ldots, c_n = v^r_n \rangle\!\rangle \mid r \in R\}\!\}$ *with* $(\psi \wedge \psi')$, *where* ψ' *presents* $\bigwedge_{r \in R, 1 \le i \le n} \psi^r_i$ *for* $[Q]_{sel} = R$ *with* ψ *and* $[e_i]^r_{exp} = v^r_i$ *with* ψ^r_i.
- $[Q_1\ \textit{UNION ALL}\ Q_2]_{sel} = S_1 \uplus S_2$ *with* $(\psi_1 \wedge \psi_2)$, *where* $[Q_i]_{sel} = S_i$ *with* ψ_i.
- $[\textit{SELECT}\ e_1\ \textit{AS}\ c_1, \ldots, e_n\ \textit{AS}\ c_n$ $\textit{FROM}\ Q_1\ \textit{CROSS JOIN}\ Q_2]_{sel}$ $= \{\!\{\langle\!\langle c_1 = v^r_1, \ldots, c_n = v^r_n \rangle\!\rangle \mid r \in (R_1 \times R_2)\}\!\}$ *with* $(\xi_1 \wedge \xi_2 \wedge \bigwedge_{r \in (R_1 \times R_2), 1 \le i \le n} \psi^r_i)$, *where* $[Q_j]_{sel} = R_j$ *with* ξ_j *and* $[e_i]^r_{exp} = x^r_i$ *with* ψ^r_i.
- $[literal]^r_{exp} = literal$ *with* true.
- $[c]^r_{exp} = r.c$ *with* true.
- $[f(e_1, \ldots, e_n)]^r_{exp} = v$ *with* $\psi \wedge \bigwedge_{1 \le i \le n} \psi_i$, *where* $[e_i]^r_{exp} = v_i$ *with* ψ_i *and an evaluation of* $f(v_1, \ldots, v_n)$ *results in* v *with* ψ *as side effect.*
- $[(\textit{SELECT}\ a(c_1)\ \textit{AS}\ c_2\ \textit{FROM}\ Q)]^r_{exp} = v$ *with* $\psi \wedge \psi'$, *where* $[Q]_{sel} = R$ *with* ψ *and an evaluation of* $\overline{a}(\{\!\{r.c_1 \mid r \in R\}\!\})$ *results in* v *with* ψ' *as side effect.*

It is possible to transform a given (full) query into an equivalent simple query. In the sequel, we explain elimination of `HAVING`, `GROUP BY`, and `DISTINCT`.

- Elimination of `HAVING`: Supposing no `WHERE`, we transform a query

```
1 | SELECT e1 AS c1, ..., en AS cn FROM t
2 |   GROUP BY f1, ..., fn HAVING h
```

  into the following query without `HAVING`:

```
1 | SELECT c1, ..., cn FROM
2 |   (SELECT e1 AS c1, ..., en AS cn, h AS ch
3 |    FROM t GROUP BY f1, ..., fn)
4 | WHERE ch
```

  Here, in the inner selection the resulted values of `h` is stored as a fresh column `ch`, which is used in the outer selection to remove unnecessary records in the outer selection.

- Elimination of `GROUP BY`: Supposing no `HAVING` nor `WHERE`, we transform a query

```
1 | SELECT e1, ..., en FROM t GROUP BY f1, ..., fn
```

  into the following query without `GROUP BY`:

```
1 | WITH tt AS (SELECT *, f1 AS g1, ..., fn AS gn FROM t)
2 | SELECT
3 |   (SELECT e1 FROM tt
4 |     WHERE tt.g1 = t2.g1 AND ... AND tt.gn = t2.gn),
5 |   ...
6 |   (SELECT en FROM tt
7 |     WHERE tt.g1 = t2.g1 AND ... AND tt.gn = t2.gn)
8 | FROM (SELECT DISTINCT g1, ..., gn FROM tt) AS t2
```

  Here, `GROUP BY` divides records into groups having same values in columns `f1,...,fn` so that aggregation function can work for each group. Thus, in order to eliminate `GROUP BY`, it enough to apply the aggregation functions `e1, ..., en` for the table `t2` with distinct value in column `f1,...,fn`.

- Elimination of `DISTINCT`: Supposing no `HAVING`, `WHERE`, nor `GROUP BY`, we transform a query

```
1 | SELECT DISTINCT c1, ..., cm FROM t
```

  into the following query without `DISTINCT`:

```
1 | WITH tt AS (SELECT c1, ..., cm, ROW_NUMBER()
2 |   OVER(ORDER BY c1, ..., cn) AS ID FROM t)
3 | SELECT c1, ... cm FROM tt AS t1
4 |   WHERE NOT EXISTS(
5 |     SELECT c1 FROM tt AS t2 WHERE t2.ID < t1.ID AND
6 |       t2.c1 = t1.c1 AND ... AND t2.cn = t1.cn
7 |   )
```

  Here, `DISTINCT` specifies that the resulted table is a set of records by eliminating redundant ones. Preparing a table `tt` attached a column `ID` with distinct value for records, the resulted query extracts a record with smallest `ID` from records having the same values in columns `c1,...,cm`. Note that we use `ROW_NUMBER`, a kind of window functions, for attaching the column `ID`.

6 SMT constraint generation via SQL transformation

This section gives a transformation tr that converts a simple query Q into a query Q' that satisfies Property 8.

For an aggregation function agg, a query `SELECT` $agg(c)$ `FROM` t returns a value determined from all data of column c in table t. We use $\overline{agg}$ to present the function used in the calculation for agg.

We prepare a function f' having side effects for each SQL function f as in Figure 1. These functions appear in the resulted SQL by the transformation tr.

Definition 11 *1. For a SQL function* f, *the function* $f'(v_1, \ldots, v_n)$ *returns a fresh variable* y, *where it produces the constraint* $y = f(v_1, \ldots, v_n)$ *as side effect.*

2. For an aggregation function $a(c)$ *we prepare* $a'(c, b)$, *which is defined as*

$$\overline{a'}(\{(v_1, b_1), \ldots, (v_n, b_n)\}) = y,$$

where y *is a fresh variable and the function produces a constraint* $y = \overline{a}(\{v_i \mid b_i = \text{true}\})$ *as side effect.*

Example 12 *1. For function* $\textit{AND}(b_1, b_2)$, *function* $\textit{AND'}(b_1, b_2)$ *returns a fresh variable* y *and produces a constraint* $y \Leftrightarrow b_1 \wedge b_2$ *as side effect.*

2. For aggregate function $\texttt{SUM}(c)$*, aggregate function* $\texttt{SUM'}(c, \texttt{ext})$ *is defined by*

$$\overline{\texttt{SUM'}}(\{(k_1, b_1), \ldots, (k_n, b_n)\}) = y$$

Here y *is a fresh variable, and it produces a constraint* $y = \mathrm{if}(b_1, k_1, 0) + \cdots + \mathrm{if}(b_n, k_n, 0)$.

We now define the SQL transformation $tr(Q)$ for simple queries Q, in which each table contains `ext` column. For simplicity, we assume that input queries have a single value expression for each `SELECT` sentence.

Definition 13 *tr is defined as follows:*

- $tr_{sel}(R^+) = R^+$.
- $tr_{sel}(\texttt{SELECT}\ e\ \texttt{AS}\ c\ \texttt{FROM}\ Q)$ $= \texttt{SELECT}\ tr_{exp}(e)\ \texttt{AS}\ c,\ \texttt{ext}\ \texttt{FROM}\ tr_{sel}(Q)$.
- $tr_{sel}(\texttt{SELECT}\ c\ \texttt{AS}\ c_1\ \texttt{FROM}\ Q\ \texttt{WHERE}\ c_2)$ $= \texttt{SELECT}\ c\ \texttt{AS}\ c_1,\ \texttt{AND'}(c_2, \texttt{ext})\ \texttt{AS ext FROM}\ tr_{sel}(Q)$.
- $tr_{sel}(\texttt{SELECT}\ c\ \texttt{AS}\ c_1\ \texttt{FROM}\ Q_1\ \texttt{CROSS JOIN}\ Q_2)$ $= \texttt{SELECT}\ c\ \texttt{AS}\ c_1,\ \texttt{AND'}(t_1.\texttt{ext}, t_2.\texttt{ext})\ \texttt{AS ext}$ $\texttt{FROM}\ tr_{sel}(Q_1)\ \texttt{AS}\ t_1\ \texttt{CROSS JOIN}\ tr_{sel}(Q_2)\ \texttt{AS}\ t_2$.
- $tr_{sel}(Q_1\ \texttt{UNION ALL}\ Q_2)$ $= tr_{sel}(Q_1)\ \texttt{UNION ALL}\ tr_{sel}(Q_2)$.
- $tr_{exp}(literal) = literal$.
- $tr_{exp}(c) = c$.
- $tr_{exp}(f(e_1, \ldots, e_n)) = f'(tr_{exp}(e_1), \ldots, tr_{exp}(e_n))$.
- $tr_{exp}(\texttt{SELECT}\ a(c)\ \texttt{AS}\ c\ \texttt{FROM}\ Q)$ $= \texttt{SELECT}\ a'(c, \texttt{ext})\ \texttt{AS}\ c\ \texttt{FROM}\ tr_{sel}(Q)$.
- $tr_{exp}(\texttt{EXISTS}\ Q)$ $= \texttt{SELECT}\ tr_{exp}(\texttt{OR}(\texttt{ext}))\ \texttt{FROM}\ tr_{sel}(Q)$, *where* `OR` *is an aggregate function defined as* $\overline{\texttt{OR}}(\{b_1, \ldots, b_n\}) = \bigvee_{1 \le i \le n} b_i$.

Example 14 *A transformation*

$tr_{sel}($`(SELECT SUM(col) FROM` R^+`)=3`$)$

produces Q' *shown in Example 9, where we regard equivalence operation* $=$ *on integers as* `EQ`.

The correctness of the transformation is induced from the following theorem, whose proof is found in the appendix.

Theorem 15 *Suppose* $[tr_{sel}(Q(R^+))]_{sel} = S^+$ *with* ψ. *Then,* $\|S^+\|_{\eta \wedge \psi} = \{[Q(R)]_{sel} \mid R \in \|R^+\|_\eta\}$.

7 Concluding remarks

All source files of CombSQL+ implemented according to this paper are open to public [5].

Acknowledgments

The work is supported by JSPS KAKENHI Grant Number JP17H01721.

References

[1] M. Gario and A. Michieli. Pysmt: A Solver-Agnostic Library for Fast Prototyping of SMT-based Algorithms. *SMT workshop 2015*, `http://smt2015.csl.sri.com/accepted-papers/`.

[2] D. Richard Hipp, Dan Kennedy, and Joe Mistachkin. SQLite. `http://www.sqlite.org`, 2000.

[3] Peter Revesz. Introduction to Constraint database. Sprinter, 2002.

[4] Genki Sakanashi and Masahiko Sakai. Transformation of Combinatorial Optimization Problems Written in Extended SQL into Constraint Problems. *PPDP 2018* pp. 19:1–19:13.

[5] CombSQL+. `https://www.trs.css.i.nagoya-u.ac.jp/projects/CombSQLplus/`

A Correctness of the transformation

This section presents a proof of Theorem 15.

From the freshness of variable names SQL functions with side effects (intorduced in Definition 11), the following properties hold.

Lemma 16 *Let* $[tr_{sel}(Q(R^+))]_{sel} = S^+$ *with* ψ. *Then*

1. $\mathrm{Var}(R^+) \cup \mathrm{Var}(S^+) \subseteq \mathrm{Var}(\psi)$, *and*

2. Variables in $\mathrm{Var}(\psi) \setminus \mathrm{Var}(R^+)$ *are all fresh.*

Lemma 17 *1. Suppose* $[tr_{sel}(Q(R^+))]_{sel} = S^+$ *with* ψ. *If a valuation* α *with* $\mathrm{Dom}(\alpha) \supseteq \mathrm{Var}(\psi)$ *satisfies* $\alpha \models \psi$, *then* $\alpha(S^+) = [Q(\alpha(R^+))]_{sel}$.

2. Suppose $[tr_{exp}(e(R^+))]^r_{exp} = v$ *with* ψ. *If a valuation* α *with* $\mathrm{Dom}(\alpha) \supseteq \mathrm{Var}(\psi)$ *satisfies* $\alpha \models \psi$, *then* $\alpha(v) = [e(\alpha(R^+)))]^{\alpha(r)}_{exp}$.

Proof We show 1. and 2. simultaneously by induction on the definition tr.

1. We present only the case

$Q(R^+) = \texttt{SELECT}\ e(R^+)\ \texttt{AS}\ c\ \texttt{FROM}\ Q_1(R^+)$.

Let $[tr_{\texttt{sel}}(Q_1(R^+))]_{\texttt{sel}} = S_1^+$ with ψ_1, and let $[tr_{\texttt{exp}}(e(R^+))]^r_{\texttt{exp}} = v^r$ with ψ^r for every $r \in S_1^+$. Then,

$[tr_{\texttt{sel}}(Q(R^+))]_{\texttt{sel}}$
$= [\texttt{SELECT}\ tr_{\texttt{exp}}(e(R^+))\ \texttt{AS}\ c, \texttt{ext}\ \texttt{FROM}\ tr_{\texttt{sel}}(Q_1(R^+))]_{\texttt{sel}}$
$= S^+$ with ψ

Here, S^+ and ψ can be represent as follows:

$S^+ = \{\!\{\langle\!\langle c = v^r, \texttt{ext} = [\texttt{ext}]^r_{\texttt{exp}} \rangle\!\rangle \mid r \in S_1^+ \}\!\}$,
$\psi = \psi_1 \wedge \bigwedge_{r \in S_1^+} \psi^r$.

Let α be a valuation that satisfies $\alpha \models \psi$. Since $\mathrm{Dom}(\alpha) \supseteq \mathrm{Var}(\psi)$ by Lemma 16, we can assume by

induction hypothesis that $\alpha(S_1^+) = [Q_1(\alpha(R^+))]_{\texttt{sel}}$, and $\alpha(v^r) = [(e(\alpha(R^+)))]_{\texttt{exp}}^{\alpha(r)}$ for each $r \in S_1^+$. From these facts, we have

$\alpha(S^+)$
$= \{\!\{\langle\!\langle c = \alpha(v^r)\rangle\!\rangle \mid r \in S_1^+, [\texttt{ext}]_{\texttt{exp}}^{\alpha(r)} = \text{true}\}\!\}$
$= \{\!\{\langle\!\langle c = [e(\alpha(R^+))]_{\texttt{exp}}^{\alpha(r)}\rangle\!\rangle \mid r \in S_1^+, [\texttt{ext}]_{\texttt{exp}}^{\alpha(r)} = \text{true}\}\!\}$
$= \{\!\{\langle\!\langle c = [e(\alpha(R^+))]_{\texttt{exp}}^{r'}\rangle\!\rangle \mid r' \in \alpha(S_1^+)\}\!\}$
$= \{\!\{\langle\!\langle c = [e(\alpha(R^+))]_{\texttt{exp}}^{r'}\rangle\!\rangle \mid r' \in [Q_1(\alpha(R^+))]_{\texttt{sel}}\}\!\}$
$= [Q(\alpha(R^+))]_{\texttt{sel}}$.

2. We show only the case
$e(R^+) = f(e_1(R^+), \ldots, e_n(R^+))$.
Let $[tr_{\texttt{exp}}(e_i(R^+))]_{\texttt{exp}}^r = y_i$ with ψ_i for each i $(1 \leq i \leq n)$. Then,
$[tr_{\texttt{exp}}(e(R^+))]_{\texttt{exp}}^r$
$= f'([tr_{\texttt{exp}}(e_1(R^+))]_{\texttt{exp}}^r, \ldots, [tr_{\texttt{exp}}(e_n(R^+))]_{\texttt{exp}}^r)$
$= y$ with $\psi \wedge \psi_1 \wedge \cdots \wedge \psi_n$.
Here, ψ is a constraint representing $y = f(y_1, \ldots, y_n)$.
Let α be a valuation that satisfies $\alpha \models \psi \wedge \psi_1 \wedge \cdots \wedge \psi_n$. Since $\mathrm{Dom}(\alpha) \supseteq \mathrm{Var}(\psi)$ by Lemma 16, we can assume by induction hypothesis that $\alpha(y_i) = [e_i(\alpha(R^+))]_{\texttt{exp}}^{\alpha(r)}$. From these facts, we have
$\alpha(y) = [f(\alpha(y_1), \ldots, \alpha(y_n)))]_{\texttt{exp}}^{\alpha(r)}$
$= f([e_1(\alpha(R^+))]_{\texttt{exp}}^{\alpha(r)}, \ldots, [e_n(\alpha(R^+))]_{\texttt{exp}}^{\alpha(r)})$
$= [e(\alpha(R^+))]_{\texttt{exp}}^{\alpha(r)}$. □

Lemma 18 *1. Suppose* $[tr_{\texttt{sel}}(Q(R^+))]_{\texttt{sel}} = S^+$ *with* ψ. *For a valuation* β *such that* $\mathrm{Dom}(\beta) = \mathrm{Var}(R^+)$, *there exists a valuation* $\alpha(\succeq \beta)$ *such that* $\alpha \models \psi$ *and* $\alpha(S^+) = [Q(\alpha(R^+))]_{\texttt{sel}}$.

2. Suppose $[tr_{\texttt{exp}}(e(R^+))]_{\texttt{exp}}^r = v$ *with* ψ. *For a valuation* β *such that* $\mathrm{Dom}(\beta) = \mathrm{Var}(R^+)$, *there exists a valuation* α *such that* $\alpha \models \psi$ *and* $\alpha(v) = [e(\alpha(R^+)))]_{\texttt{exp}}^{\alpha(r)}$.

Proof We show 1. and 2. simultaneously by induction on the definition tr.

1. We present only the case
$Q(R^+) = \texttt{SELECT } e(R^+) \texttt{ AS } c \texttt{ FROM } Q_1(R^+)$.
Suppose $[\mathrm{tr}(Q_1(R^+))]_{\texttt{sel}} = S_1^+$ with ψ_1, and $[tr_{\texttt{exp}}(e(R^+))]_{\texttt{exp}}^r = v^r$ with ψ^r for every $r \in S_1^+$. Then,
$[tr_{\texttt{sel}}(Q(R^+))]_{\texttt{sel}}$
$= [\texttt{SELECT } tr_{\texttt{exp}}(e(R^+)) \texttt{ AS } c, \texttt{ext FROM } tr_{\texttt{sel}}(Q_1(R^+))]_{\texttt{sel}}$
$= S^+$ with ψ.
Here, S^+ and ψ can be represent as follows:
$S^+ = \{\!\{\langle\!\langle c = v^r, \texttt{ext} = [\texttt{ext}]_{\texttt{exp}}^r\rangle\!\rangle \mid r \in S_1^+\}\!\}$,
$\psi = \psi_1 \wedge \bigwedge_{r \in S_1^+} \psi^r$.
For a given valuation β such that $\mathrm{Dom}(\beta) = \mathrm{Var}(R^+)$, by induction hypothesis there exists a valuation α_1, α^r $(\succeq \beta)$ such that $\alpha_1 \models \psi_1$, $\alpha^r \models \psi^r$, $\alpha_1(S_1^+) = [Q_1(\alpha_1(R^+))]_{\texttt{sel}}$, and $\alpha^r(v^r) = [e(\alpha^r(R^+))]_{\texttt{exp}}^r$. By 2. of Lemma 16, these valuations are compatible, and hence union α of all these valuations are well defined. Here, $\alpha_1(R^+) = \alpha^r(R^+) = \alpha(R^+) = \beta(R^+)$ and $\alpha \models \psi$. From these facts, we have
$\alpha(S^+)$
$= \{\!\{\langle\!\langle c = \alpha(v^r)\rangle\!\rangle \mid r \in S_1^+, [\texttt{ext}]_{\texttt{exp}}^{\alpha(r)} = \text{true}\}\!\}$
$= \{\!\{\langle\!\langle c = [e(\alpha(R^+))]_{\texttt{exp}}^{\alpha(r)}\rangle\!\rangle \mid r \in S_1^+, [\texttt{ext}]_{\texttt{exp}}^{\alpha(r)} = \text{true}\}\!\}$
$= \{\!\{\langle\!\langle c = [e(\alpha(R^+))]_{\texttt{exp}}^{r'}\rangle\!\rangle \mid r' \in \alpha(S_1^+)\}\!\}$
$= \{\!\{\langle\!\langle c = [e(\alpha(R^+))]_{\texttt{exp}}^{r'}\rangle\!\rangle \mid r' \in [Q_1(\alpha(R^+))]_{\texttt{sel}}\}\!\}$
$= [Q(\alpha(R^+))]_{\texttt{sel}}$.

2. We show only the case
$e(R^+) = f(e_1(R^+), \ldots, e_n(R^+))$.
Suppose $[tr_{\texttt{exp}}(e_i(R^+))]_{\texttt{exp}}^r = y_i$ with ψ_i for each i $(1 \leq i \leq n)$. Then,
$[tr_{\texttt{exp}}(e(R^+))]_{\texttt{exp}}^r$
$= f'([tr_{\texttt{exp}}(e_1(R^+))]_{\texttt{exp}}^r, \ldots, [tr_{\texttt{exp}}(e_n(R^+))]_{\texttt{exp}}^r)$
$= y$ with $\psi \wedge \psi_1 \wedge \cdots \wedge \psi_n$,
where, ψ is an SMT-constraint that represents $y = f(y_1, \ldots, y_n)$.
For a given valuation β such that $\mathrm{Dom}(\beta) = \mathrm{Var}(R^+)$, by induction hypothesis there exists a valuation $\alpha_i(\succeq \beta)$ for $i = 1, \ldots, n)$ such that $\alpha_i \models \psi_i$ and $\alpha_i(y_i) = [e_i(\alpha_i(R^+))]_{\texttt{exp}}^{\alpha_i(r)}$. By 2. of Lemma 16 and the freshness of y, the valuations α_i and $\{y \mapsto f(\alpha_1(y_1), \ldots, \alpha_n(y_n))\}$ are compatible. Let α be the union of the all valuations, where $\alpha_i(R^+) = \alpha(R^+) = \beta(R^+)$ and $\alpha \models \psi \wedge \bigwedge_i \psi_i$. From these facts, we have
$\alpha(y) = [f(\alpha(y_1), \ldots, \alpha(y_n)))]_{\texttt{exp}}^{\alpha(r)}$
$= f([e_1(\alpha(R^+))]_{\texttt{exp}}^{\alpha(r)}, \ldots, [e_n(\alpha(R^+))]_{\texttt{exp}}^{\alpha(r)})$
$= [e(\alpha(R^+))]_{\texttt{exp}}^{\alpha(r)}$. □

Theorem 15 *Suppose* $[tr_{\texttt{sel}}(Q(R^+))]_{\texttt{sel}} = S^+$ *with* ψ. *Then,* $\|S^+\|_{\eta \wedge \psi} = \{[Q(R)]_{\texttt{sel}} \mid R \in \|R^+\|_\eta\}$.

Proof $(\subseteq)$ Let $S \in \|S^+\|_{\eta \wedge \psi}$. Then $S = \alpha(S^+)$ for some α such that $\alpha \models \eta \wedge \psi$. By 1. of Lemma 17, it follows that $\alpha(S^+) = [Q(\alpha(R^+))]_{\texttt{sel}}$. Since $\alpha(R^+) \in \|R^+\|_\eta$ is induced from $\alpha \models \eta$, we obtain that S is the right-hand side.

$(\supseteq)$ Let S in the right-hand side, then $S = [Q(R)]_{\texttt{sel}}$ for some $R \in \|R^+\|_\eta$, and thus there exists a valuation β such that $\beta \models \eta$ and $R = \beta(R^+)$. From these, we obtain $S = [Q(\beta(R^+))]_{\texttt{sel}}$, where we can assume $\mathrm{Dom}(\beta) = \mathrm{Var}(R^+)$. By Lemma 18, there exists a valuation $\alpha(\succeq \beta)$ such that $\alpha \models \psi$ and $\alpha(S^+) = [Q(\alpha(R^+))]_{\texttt{sel}}$. Since $[Q(\alpha(R^+))]_{\texttt{sel}} = [Q(\beta(R^+))]_{\texttt{sel}}$ and $\beta \models \psi \wedge \eta$ induced from these, we obtain $S \in \|S^+\|_{\eta \wedge \psi}$. □