Transformation of Combinatorial Optimization Problems Written in Extended SQL into Constraint Problems

Genki Sakanashi
Graduate School of Informatics, Nagoya University
sakanashi@trs.css.i.nagoya-u.ac.jp

Masahiko Sakai
Graduate School of Informatics, Nagoya University
sakai@i.nagoya-u.ac.jp

ABSTRACT

The combinatorial optimization is an important area, which gives one of the best solutions for various problems. This paper focuses on an SQL style of declarative languages to ease describing combinatorial optimization problems, and provides their solution method powered by state-of-the-art CP/SMT solvers. From the semantic point of view, the search space of a combinatorial problem is given as a finite set of relations. Relations in the search space are filtered by constraints of the problem in similar to the `filter`-function on lists in functional languages, and the resulted relations are solutions of the problem. According to this notion, we extended Structured Query Language (SQL) by introducing some operations on sets of relations: generating a set of relations, filtering a set of relations according to constraints, and selecting one of the optimum relations with respect to a goal function. Toward an effective implementation, a set of relations is represented as a pair of a relation containing variables with finite domains and constraints on variables. This enables us to solve the target problem by CP/SMT solvers. We also give an experimental result on the graph vertex coloring optimization problem.

KEYWORDS

combinatorial optimization, SQL, satisfiability solver, language for specifying problems

ACM Reference Format:
Genki Sakanashi and Masahiko Sakai. 2018. Transformation of Combinatorial Optimization Problems Written in Extended SQL into Constraint Problems . In *The 20th International Symposium on Principles and Practice of Declarative Programming (PPDP '18), September 3–5, 2018, Frankfurt am Main, Germany.* ACM, New York, NY, USA, Article 4, 13 pages. https://doi.org/10.1145/3236950.3236963

1 INTRODUCTION

The *combinatorial optimization* is an important area, which gives one of the best solutions for various problems. For example, curriculum based course timetabling [10] is one of such problems. Given general rules such as a room constraint that inhibits to assign more than one class to a room simultaneously, and concrete data such as lists of rooms and classes, it is a problem to implement a curriculum.

PPDP '18, September 3–5, 2018, Frankfurt am Main, Germany

ACM ISBN 978-1-4503-6441-6/18/09…$15.00
https://doi.org/10.1145/3236950.3236963

Constraint programming (CP) is a system for combinatorial optimization problems: OPL [8], EaCL [11, 15], NCL [17], ESRA [6], ESSENCE [7], MiniZinc [12], Copris [13], and so on. In applying a CP system to a given problem, we must break down and describe it in the CP modelling language. In this process, we need to design variables for search space in addition to variables standing for an input instance, and then specify constraints among these variables. It is also important for users that the separation of a problem description and its instances of the problem. It is also necessary to associate data in instances of a real problem/benchmark to be solved with the input variables in a CP description, which is not essential but tiresome work.

Marco Cadoli et al. have approached to these problems by extending *structured query language* (SQL) for relational database. They generalized relational algebra, called NP-Alg, which contains `Guess` operation that declares a guessed relation by introducing variables, and designed a description language *ConSQL* [2]. ConSQL has a benefit that constraints are specified as CHECK queries written in ordinary SQL. A possible extension of `Guessed` relations is a solution if it causes an empty relation for CHECK queries. They designed ConSQL+ [4] by introducing `Choose` operator to declare a guessed relation, in which variable introduction is not necessary. They have proposed local searching methods [3, 4] that find locally optimal solutions, where the global optimality is not guaranteed at all.

The authors' group have proposed CombSQL as a simple extension of SQL by introducing `FIND`-structure [16] with unaware of the series of ConSQL results. CombSQL provides 'subsets' to define a search space, while ConSQL provides not only subsets but also more way: functions, partitions, and permutations. In this sense, the language is essentially included with the work by Cadoli et al. On the other hand, CombSQL uses SAT solvers to obtain globally optimum solutions.

This paper designs an extended language *CombSQL+* with simple and clear semantics by introducing sets of relations as search space, and propose a method to obtain globally optimum solution powered by state-of-the-art CP (Constraint Programming) solvers or SMT (Satisfiable Modulo Theories) solvers. The 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 problems in CombSQL+. Moreover, such parts can be debugged by an existing database engine if test data prepared.
- It provides an easy separation of a problem description and its instances.
- 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.

We will explain these ideas by a tiny example.

Example 1.1. Given three relational tables shown in Table 1, we want to find a products-assignment into baskets that maximizes the total size. Here, the sum of the size of products in a basket may not exceed its capacity. Inhibited combinations of a product and a basket are given in Ban rule (in Table (c)).

The assignments (partial functions) of products into baskets, shown in Table 2, are solutions. The table (a) is optimal one maximizing the goal function, which returns the total size of products in baskets.

Table 1: Tables of an instance

(a) Products (PL)

product	size
A	4
B	3
C	6
D	1
E	3

(b) Baskets (BL)

basket	capacity
a	10
b	6

(c) Ban rule (Ban)

product	basket
A	b
B	a
D	a

Table 2: Solutions

(a) An optimal solution

product	basket
A	a
B	b
C	a
E	b

(b) A non-optimal solution

product	basket
A	a
C	b
E	a

Table 3: A non-solution

product	basket
B	a
C	b
D	b
E	a

The problem is described in a *generate-filter-select* style in similar to ordinary list programming in functional language. (The descriptions are found in Section 3)

Generate: We define the set SearchSpace of assignments so that it covers all solutions. Since a solution is a partial function in this example, SearchSpace consists of all the partial functions from products to baskets. Of course, SearchSpace contains solutions in Table 2 as well as a non-solution in Table 3.

Filter: We write a constraint to eliminate the tables according to Ban rule, and also the tables with over capacities, which results in the set ReducedSpace of solutions.

Select: We specify a declaration to select a table having a maximum goal value from ReducedSpace nondeterministically.

For an implementation, we represent the set SearchSpace as a single table that contains variables, called a constrained table (Table 4). Here the Boolean value in the column ext shows that the record exists or not. The generating step produces such a table, where ranges of variables are shown below the table. The filtering step produces constraints over variables such as $(x_A \neq b \vee \neg y_A) \wedge (x_B \neq a \vee \neg y_B) \wedge (x_D \neq a \vee \neg y_D)$ for Ban rules, and some constraints on basket capacities. Finally the selecting step finds an optimal solution by solving the produced constraints using an appropriate CP/SMT solver.

Table 4: A constrained table

product	basket	ext
A	x_A	y_A
B	x_B	y_B
C	x_C	y_C
D	x_D	y_D
E	x_E	y_E

$x_A, x_B, \ldots, x_E \in \{a, b\}$
$\wedge\ y_A, y_B, \ldots, y_E \in \{\text{true, false}\}$

We designed its syntax so as to reflect our generate-filter-select policy. Here, the essential idea of the syntax is the same as the work of Mancini et al. [4]. Our set semantics is very clear and simple. Implementation idea is quite different: they proposed a local search algorithm while we propose a constraint-based entire search. We propose a transformation them into constraints, which enable us to enjoy benefits of the stat-of-art CP/SMT-solvers.

As a common benefit, both constraints and optimization functions are mostly represented in ordinary SQL. In real, as shown later, we can give an SQL description (see lines 5–9 in Figure 2) that returns the table consisting of baskets spilled over in capacity for each solution candidate table. Hence, the capacity constraint is represented as the emptiness of the returned table of the sub-query.

2 PRELIMINARY

The set of Boolean values are $\mathbb{B} = \{\text{true}, \text{false}\}$. 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 $\{\{1, 1, 2, 3\}\} \subseteq \{\{1, 1, 2, 3, 3, 4\}\}$.

An *assignment* α gives a value to each variable in its *domain* $\mathrm{Dom}(\alpha)$. We write $\emptyset$ for the *empty assignment* having an empty domain. We use $[x_1 \leftarrow v_1, \ldots, x_n \leftarrow v_n]$ to represent an assignment α with $\alpha(x_i) = v_i$ for $x_i \in \mathrm{Dom}(\alpha)$. If assignments 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 assignments 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 assignments α and β satisfies that $\mathrm{Dom}(\alpha) \supseteq \mathrm{Dom}(\beta)$, we say that α is an *instance* of β, written as $\beta \preceq \alpha$. We use $\alpha|_W$ for β such that $\beta \preceq \alpha$ and $\mathrm{Dom}(\beta) = W$.

We use a substitution for replacing an object in the sentences with some symbol or value. We write a substitution θ as $[x_1 \leftarrow$

$y_1, \ldots, x_n \leftarrow y_n]$. $\theta(T)$ represents the one obtained from T by replacing each occurrence of x_i with y_i.

A relational database is regarded as a collection of *tables*. A table is a collection of *records*, each of which is a collection of data (or values). Each data is associated with a *column name*. For example, consider a table PL in Table 1 (a). This table has two column names `product` and `size`, and five records. The data 4 in the first record $\{A, 4\}$ is associated with a column name `size`. A *framework* of tables is a collection of pairs of a column name and its data type. The framework of PL is {(`product`, STRING), (`size`, INT)}. All values in a table must follow its framework, that is, each value in a column must be with the type determined by the framework.

We write a record r as $\{c_1 = d_1, \ldots, c_n = d_n\}$ for the column names $c_1, \ldots, c_n$ of a table T and their corresponding data values $d_1, \ldots, d_n$. The set $\{c_1, \ldots, c_n\}$ of column names of a record r is denoted by $\mathrm{Col}(r)$ or $\mathrm{Col}(T)$. The value d_i is represented as $c_i(r)$ by regarding the column name c_i as a function from records to data. We use $c_i(T)$ or $\mathrm{Img}(c_i)$ for the image $\{c_i(r) \mid r \in T\}$ of c_i in T. For example, the first record r_1 of PL in Table 1 (a) is displayed as {`product` = A, `size` = 4}. We have $\mathrm{Col}(r_1)$ = {`product`, `size`}, `size`$(r_1) = 4$, and `size`(PL) = {1, 3, 4, 6}.

The *structured query language* (SQL) [5] is a standard language for managing data held in a relational database system. A query that retrieves some kind of information as a table, is called a *table query*. *Boolean query*s and *integer query*s are used similarly. For example, the following table query returns the table consisting of the records in PL in Table 1 (a) whose size are 3:

SELECT $*$ FROM PL WHERE `size` = 3

Here, $*$ is an abbreviation of the sequence of column names in PL.

We briefly illustrate fragments of SQL used in this paper. Here, we classify them by the types of their return values: Boolean operators, integer operators, and table operators. In the sequel, we use b, k, and R to represent a Boolean value, an integer value, and a table, respectively.

Boolean operators are defined as follows:

(1) The following operators have the expected meaning.
  - NOT b.
  - b_1 *bop* b_2, where *bop* is AND or OR.
  - k_1 *cop* k_2, where *cop* is either =, <>, <, etc.

(2) EXISTS R, which returns true if and only if the table R has at least one record.

Integer operators are defined as follows:

(1) k_1 *iop* k_2, where *iop* is either +, −, etc.
(2) SELECT $a(c)$ FROM R, where c is a column name of a table R, and a is an aggregate operator. This structure returns $\bar{a}\{c(r) \mid r \in R\}$, where $\bar{a}$ is a function on sets of values corresponds to a. For example, for the aggregate operator SUM, $\overline{\text{SUM}}$ is a function Σ that returns the total sum of the integers in a given set.

Here is a list of typical aggregate operators.

- SUM: the total sum of integers.
- AVG: the average of integers.
- MAX: the maximum number of integers.
- COUNT: the number of records.

For example, the following query returns the total size of products in the table PL.

```
SELECT SUM(size) FROM PL
```

Table operators are defined as follows:

(1) R_1 UNION ALL R_2, which returns $R_1 \uplus R_2$, where tables R_1 and R_2 must have the same number of columns without type mismatching.
(2) R_1 CROSS JOIN R_2, which returns $\{\!\{r_1 \cup r_2 \mid r_1 \in R_1, r_2 \in R_2\}\!\}$.
(3) SELECT $*$ FROM R AS *tn* WHERE *const*, which returns
$$\{\!\{r \mid r \in R, \overline{const}^{\,r} = \text{true}\}\!\},$$
where $\overline{const}^{\,r}$ is the value of an expression $\theta(const)$, which is obtained from *const* by applying the substitution θ:
$$\theta = [tn.c \leftarrow c(r) \mid c \in \mathrm{Col}(r)].$$
(4) SELECT $c_1, \ldots, c_n$ FROM R, which returns
$$\{\!\{\{c_1 = c_1(r), \ldots, c_n = c_n(r)\} \mid r \in R\}\!\}$$
(5) SELECT $c, a(d)$ AS *dn* FROM R GROUP BY c, which returns
$$\{\!\{\{c = v, dn = w\} \mid v \in c(R), w = \bar{a}\{d(r) \mid r \in R, c(r) = v\}\}\!\}$$
Here c,d is column names and a is an aggregate operator, c and $a(d)$ are possibly lists but we presented this simple case to ease presentation.

Note that WHERE constraints sometimes contain a query, called *subquery*. A single columned table, which is a result of a subquery, is regarded as a value. If it has more than one records, the first record is used for the value.

SQL has the other operators, but most of them can be decomposed to the preceding ones. We provide some of such decomposition.

(1) SELECT *columns* FROM R WHERE *const* is decomposed to

SELECT *columns* FROM (SELECT $*$ FROM R WHERE *const*).

(2) R_1 INNER JOIN R_2 ON $R_1.c_1 = R_2.c_2$ is decomposed to

SELECT $*$ FROM (R_1 CROSS JOIN R_2) WHERE $R_1.c_1 = R_2.c_2$.

(3) SELECT $c, a(d)$ FROM R WHERE $const_1$
GROUP BY c HAVING $const_2$
is decomposed to

SELECT $c, a(d)$
FROM (SELECT $c, a(d)$
FROM (SELECT $*$ FROM R WHERE $const_1$)
GROUP BY c)
WHERE $const_2$

(4) SELECT DISTINCT c FROM R is decomposed to

SELECT c FROM R GROUP BY c

(5) SELECT a(DISTINCT $c2$) FROM R GROUP BY $c1$
is decomposed to

SELECT $a(c2)$ FROM (
SELECT $c1, c2$ FROM R GROUP BY $c1, c2$
) GROUP BY $c1$

Note that the relational algebra is designed by regarding each table as a set of records. On the other hand, the SQL follows multiset semantics by default, and uses the word DISTINCT explicitly to remove duplicated elements from a table. This paper discusses based on multiset setting.

3 COMBSQL+: LANGUAGE FOR COMBINATORIAL PROBLEMS

An instance of *combinatorial optimization problems* is $\langle X, P, f \rangle$, where X is a set of solution candidates, called a *search space*, P is a predicate on X, and $f : X \to \mathbb{Z}$ is an *optimization function* (or, *goal function*). For a given instance, the *optimum solutions* are the ones with the greatest value of f among the *feasible solutions* $\{x \in X \mid P(x)\}$. For Example 1.1, the set X is SearchSpace, the predicate P corresponds to the Ban constraint and the capacity constraint, and the goal function f returns the total size of products in baskets given a table in ReducedSpace.

In the sequel, we design an extended SQL, named CombSQL+, for describing combinatorial optimization problems as combination of generation, filtering, and selection steps.

For the generating step, we introduce a *choose_query*[1] to produce a set of tables:

$$\text{SELECT } c_1, \ldots, c_n \text{ FROM } R$$

where each c_i is either a column name of the table R or a term in the form of CHOOSE(S_i). Here S_i is a single columned table. For a simple presentation, we focus on the form

$$\text{SELECT } c, \text{CHOOSE}(S) \text{ FROM } R$$

where S has a single column name s. The query represents the following set:

$$\{ \{\!\{\{c = c(r), s = f(r)\} \mid r \in R\}\!\} \mid f : R \to \mathrm{Img}(S)\}$$

where f is a total function that returns a value in S given a record r in R.

When the choosing-query has a preceding SUBTABLE OF, like

$$\text{SUBTABLE OF } \textit{choose\_query},$$

it gives the union of the power sets of $R \in \textit{SetTab}$

$$\{R' \mid R' \subseteq R,\ R \in \textit{SetTab}\},$$

where *SetTab* is the set of tables induced by *choose_query*.

We introduce a creating set statement that bind a set of tuples of tables to a name.

$$\begin{array}{l} \text{CREATE SET } \textit{sname} \\ \quad \text{HAS } (Q_1(t_1, \ldots, t_n), \ldots, Q_m(t_1, \ldots, t_n)) \\ \quad \text{FOR } t_1 \text{ IN } \textit{choose\_query}_1,\ \ldots,\ t_n \text{ IN } \textit{choose\_query}_n \end{array}$$

where each Q_i is an ordinary SQL query that returns a table. The meaning is given by

$$\begin{array}{rl} \textit{sname} := \{(s_1, \ldots, s_m) \mid & s_j = Q_j(t_1, \ldots, t_n), t_i \in \textit{SetTab}_i, \\ & i = 1, \ldots, n, j = 1, \ldots, m\}, \end{array}$$

where each $\textit{SetTab}_i$ is the set of tables induced by $\textit{choose\_query}_i$. In case of $n = m = 1$, it is in the form of

$$\text{CREATE SET } \textit{sname} \text{ HAS } Q(t) \text{ FOR } t \text{ IN } \textit{choose\_query}.$$

[1] CHOOSE-structure was proposed by Mancini et al. [4] to denote an arbitrary element in the given single-columned table.

Moreover, if Q is identity, we may write it simply as

$$\text{CREATE SET } \textit{sname} \text{ AS } \textit{choose\_query}.$$

The generating step of Example 1.1 is described as shown in Figure 1. It creates a set of tables containing all possible tables including Table 2 (a) and (b), Table 3, and so on.

```
CREATE SET SeachSpace AS
  SUBTABLE OF (SELECT product,CHOOSE(BL) FROM PL)
```

Figure 1: Generation of the search space for Example 1.1

Filtering steps are given in the following form, called *filtering statements*:

$$\begin{array}{c} \text{CREATE SET } \textit{sname}_1 \text{ HAS } (t_1, \ldots, t_n) \text{ IN } \textit{sname}_2 \\ \text{SUCH THAT } \textit{BoolQuery}(t_1, \ldots, t_n) \end{array}$$

where $\textit{BoolQuery}(t_1, \ldots, t_n)$ is an ordinary SQL returning Boolean value, where bound variables $t_1, \ldots, t_n$ over tables are explicitly displayed. The meaning is given as

$$\textit{sname}_1 := \{(t_1, \ldots, t_n) \in \textit{sname}_2 \mid \textit{BoolQuery}(t_1, \ldots, t_n)\}.$$

The filtering step shown in Figure 2 works for Example 1.1. Here we introduce a macro notation similarly to `#define` in C language. The lines 2–3 declare a look-up function of a given association list, where the function searches a given value v from the column *key* of the table *asc* and returns the value of the column *val*. The lines 8–12 specifies Ban constraint by stating no common record exists. The lines 13–19 specifies the capacity constraint, in which GROUP BY indicates that a record is possibly created for each basket, and HAVING specifies the condition: the summation of the size of products in the basket exceeds of its capacity.

Finally a selection step chooses a table from a given set of tables that maximizes (or minimizes) a goal function.

$$\begin{array}{l} \text{CREATE TABLE } \textit{tname}_1, \ldots, \textit{tname}_n \text{ AS } (t_1, \ldots, t_n) \text{ IN } \textit{sname} \\ \quad \textit{Goal}(t_1, \ldots, t_n) \end{array}$$

```
1  // lookup v from association table (key,val)
2  MACRO lookup(v,asc,key,val) AS
3    (SELECT val FROM asc WHERE v = key)
4
5  CREATE SET ReducedSpace
6    HAS t IN SearchSpace
7    SUCH THAT
8      NOT EXISTS (
9        t INNER JOIN Ban ON
10         t.product = Ban.product
11         AND t.basket = Ban.basket
12     )
13     AND NOT EXISTS(
14       SELECT * FROM t
15         GROUP BY basket
16         HAVING
17          SUM(lookup(t.product,PL,product,size))
18           > lookup(t.basket,BL,basket,capacity)
19     )
```

Figure 2: Filtering constraint for solutions of Example 1.1

where an integer query $Goal(t)$ is optional, and written as either

$$\text{MAXIMIZING } IntegerQuery(t_1, \ldots, t_n), \text{ or}$$
$$\text{MINIMIZING } IntegerQuery(t_1, \ldots, t_n).$$

The trivial meaning is

$$(tname_1, \ldots, tname_n) := (t_1, \ldots, t_n) \in sname$$

such that

$$IntegerQuery(t_1, \ldots, t_n) = \max\{IntegerQuery(\vec{u}) \mid \vec{u} \in sname\}.$$

The selection step shown in Figure 3 works for Example 1.1.

```
CREATE TABLE solution AS t IN ReducedSpace
  MAXIMIZING
    SELECT SUM(sizeOf(product)) FROM t
```

Figure 3: Selection step with goal function of Example 1.1

In the last of this section, we stress that all we introduced here are set manipulating operations except for CHOOSE and SUBTABLE OF fragments. In other words, we didn't extended table handling operators such as JOIN, SELECT, GROUP BY, and so on, hence SQL programmers can easily enjoy CombSQL+ for solving constraint optimization problem.

4 DESCRIPTION EXAMPLES

4.1 Graph vertex coloring problem

The *graph coloring* is an assignment of a color for each vertex such that no two adjacent vertices have the same color. A graph is k-*colorable* if there exists a graph coloring using at most k colors. Suppose we want to know minimum k such that a given graph is k-colorable.

Instances are presented as tables, whose framework is shown in Figure 4, where CREATE TABLE is an ordinary SQL command to create an empty table with the specified framework.

```
CREATE TABLE V ( v      INT KEY  );  // vertices
CREATE TABLE E ( v1,v2 INT       );  // edges
```

Figure 4: Framework of tables for k-coloring instances

Figure 5 is a description of the problem, where we assume a table CL consists of records $1, 2, 3, \ldots, m$ in a single column c for m large enough. Lines 8–9 list up edges connecting the same colored vetices, where each table t in SearchSp can be regarded as an association list whose key is a vertex and the return value is a color.

4.2 Parallel session assignment problem

We show a CombSQL+ description for a parallel session assignment problem, which is a bit simplified version of our initially motivated problem. First we explain this optimization problem. Each student belongs to a laboratory. Each laboratory belongs to a group. The problem is a construction of a presentation program satisfying given constraints. Here, a students-presentation program consists of several sessions scheduled in serial. Each session has a couple of sub-sessions held simultaneously, which are distinguished by rooms.

```
1  CREATE SET SearchSp AS
2    SELECT v,CHOOSE(CL) FROM V
3  // lookup v from association (key,val) in asc
4  MACRO lookup(v,asc,key,val) AS
5    SELECT val FROM asc WHERE v = key
6  CREATE SET Solutions HAS t IN SearchSp
7    SUCH THAT NOT EXISTS(
8      SELECT * FROM E
9        WHERE lookup(E.v1,t,v,c)=lookup(E.v2,t,v,c)
10   )
11 CREATE TABLE solution AS t IN Solutions
12   MINIMIZING (SELECT max(c) FROM t)
```

Figure 5: Query for vertex coloring problem

Instances of the parallel session problem are supplied in a database as a couple of tables, whose framework is shown in Figure 6.

- Lines 1–5: the table Students consists of records each of which specifies a student information including his/her affiliated laboratory name.
- Lines 6–11: the table SubSessions consists of records each of which specifies a sub-session information with the allowed number of presentations.
- Lines 12–15: the table Bans consists of records specifying combinations of a laboratory and a session to be avoided.
- Lines 16–19: the table Labs specifies a laboratory with the affiliated group.

```
1  CREATE TABLE Students (
2    student_id  TEXT KEY,
3    name        TEXT,
4    lab_name    TEXT
5  );
6  CREATE TABLE SubSessions (
7    subsess_id  INT KEY,
8    room        TEXT,
9    session     INT,
10   presen_max  INT
11 );
12 CREATE TABLE Bans (
13   lab_name    TEXT,
14   session     INT
15 );
16 CREATE TABLE Labs (
17   lab_name    TEXT KEY,
18   lab_gr      TEXT,
19 );
```

Figure 6: Framework of tables for problem instances

Constraints are classified into hard ones, which must be satisfied, and soft ones, which are likely to be satisfied as much as possible. The hard constraints are listed as follows:

H1 Each sub-session has at most a specified number of presentations.

H2 Any presentation given by a student in a session is not allowed if the combination of the laboratory of the student and the session appears in the ban list.

H3 For each session, the presentations given by students in the same laboratory must be summarized to a single room, i.e., no two presentations by students of the same laboratory in different rooms are inhibited in a session.

The soft constraints are listed as follows:

S1 It is preferred to increase the number of laboratories for each sub-sessions.

S2 It is preferred to decrease the number of groups for each sub-sessions.

S3 It is preferred to decrease the number of rooms for each laboratory.

Figure 7 shows a CombSQL+ description for the problem. The

```
20 CREATE VIEW SubSessionIds AS
21   SELECT subsess_id FROM SubSessions;
22 CREATE VIEW StudentsWithGroup AS
23   SELECT * FROM Students CROSS JOIN Labs
24     WHERE Labs.lab_name = Students.lab_name;
25 CREATE SET SearchSpace AS
26   SELECT *, CHOOSE(SubSessionIds)
27     FROM StudentsWithGroup;
28 CREATE SET ExtendedSearchSpace HAS
29   SELECT * FROM t CROSS JOIN SubSessions
30     WHERE subsess_id = SubSessions.id
31   FOR t IN SearchSpace;
32 /* for H1 */
33 CREATE SET ReducedH2Space HAS t
34   IN ExtendedSearchSpace
35   SUCH THAT NOT EXISTS(
36     SELECT * FROM t GROUP BY subsess_id
37       HAVING COUNT(student_id) > presen_max
38   );
39 /* for H2 */
40 CREATE SET ReducedH3Space HAS t IN ReducedH2Space
41   SUCH THAT NOT EXISTS(
42     SELECT * FROM t WHERE EXISTS (
43       SELECT * FROM Bans
44         WHERE Bans.lab_name = t.lab_name
45           AND Bans.session = t.session
46     )
47   );
48 /* for H3 */
49 CREATE SET ReducedH4Space HAS t IN ReducedH3Space
50   SUCH THAT NOT EXISTS(
51     SELECT * FROM t AS t1 CROSS JOIN t AS t2
52       WHERE t1.lab_name = t2.lab_name
53         AND t1.room <> t2.room
54         AND t1.session = t2.session
55   );
56 CREATE TABLE solution AS t IN ReducedH4Space
57   MAXIMIZING
58   /* for S1 */
59   (SELECT SUM(kind) AS s
60      FROM (SELECT COUNT(DISTINCT lab_name) AS kind
61        FROM t GROUP BY room, session))
62   /* for S2 */
63   - (SELECT SUM(kind) AS s
64      FROM (SELECT COUNT(DISTINCT lab_gr) AS kind
65        FROM t GROUP BY room, session))
66   /* for S3 */
67   - (SELECT SUM(kind) AS s
68      FROM (SELECT COUNT(DISTINCT room) AS kind
69        FROM t GROUP BY lab_name));
```

Figure 7: A CombSQL+ description for the parallel session problem

lines 25–31 is the generating step. This part generates the set SearchSpace in lines 25–27 and then attaches session information in lines 28–31. The lines 32–55 is the filtering step. This part describes the hard constraints. The lines 56–69 is the selecting step. This part describes the goal function representing the soft constraints.

We give the correspondence between each constraints and the lines of Figure 7, and brief explanations.

- The hard constraint H1 is specified in lines 33–38 as the emptiness of the sub-sessions having exceeded number of presentations.
- The hard constraint H2 is specified in lines 40–47 as the emptiness of the assignment that matches Bans.
- The hard constraint H3 is specified in lines 49–55 as the emptiness of the combination of two presentations with the same session and the same laboratory but different rooms.
- The soft constraint S1 is specified in lines 59–61 as the total sum of numbers each of which is the distinct count of laboratories for each sub-session.
- The soft constraint S2 is specified in lines 63–65 as the total sum of numbers each of which is the distinct count of groups for each sub-session.
- The soft constraint S3 is specified in lines 67–69 as the total sum of numbers each of which is the distinct count of assigned rooms for each laboratory.

5 PROCESSING COMBSQL+ DESCRIPTION

This section discusses on processing CombSQL+ descriptions introduced in Section 3. We now explain the overview of the process for a problem without optimization for simplicity. First, we compile a problem description written in CombSQL+ and an instance of the problem in a database, and produce a CP/SMT instance. Second, we obtain a solution for the CP/SMT instance by a solver. Finally, we construct a solution of the original problem instance from the solution of the CP/SMT instance.

The key ideas are a constrained table for representing a set of tables, and an extension of ordinary SQL operations to those on constrained tables. Here a constrained table is a pair $\langle R, \varphi\rangle$ of a single table containing variables over finite domains and a constraint on them. Suppose that a constrained table $\langle R, \varphi\rangle$ represents the search space, and that $BoolQuery(t)$ is located after SUCH THAT in a filtering-step description. By the extended SQL operations, $BoolQuery(\langle R, \varphi\rangle)$ results in a constrained table $\langle y, \varphi'\rangle$, where the first item y is a Boolean variable because the resulted constrained table represents a set of Boolean values. Once a satisfiable solution σ for $\varphi \wedge y \wedge \varphi'$ is found by a CP/SMT solver, the final solution is obtained from the first item R of the solution space by applying σ.

5.1 Constrained tables

We formalize a constrained table to represent a set of tables. In the introduction, we have shown a constrained table in Table 4 that represents the search space of the example.

Definition 5.1. An *extended table* is a table that possibly contains variables as data, and must have the column name ext with Boolean type. An *constrained table* $\mathcal{R}$ is a pair $\langle R^+, \varphi\rangle$ of an extended table R^+ and a constraint φ, where constraints are on some fixed model.

Remark that extended tables may contain variables, but may not contain expressions.

For an assignment α that assigns a variable to a data value with an appropriate type, we write $\alpha \models \varphi$ when an α satisfies a constraint φ.

The set $\|\mathcal{R}\|_\emptyset$ represented by a constrained table $\mathcal{R}$ is defined as follows, where $\emptyset$ is an empty assignment having empty domain.

Definition 5.2. Let $\langle R^+, \varphi\rangle$ be a constrained table.

(1) $\mathrm{Var}(R^+)$ denotes the set of variables in R^+.
(2) $\alpha(R^+)$ is a table consisting of all the records $\langle c_1 = \alpha(d_1), \ldots, c_n = \alpha(d_n)\rangle$ for $\{c_1 = d_1, \ldots, c_n = d_n, \mathtt{ext} = x\} \in R^+$ such that $\alpha(x) = $ true.
(3) The set of tables represented by a constrained table $\langle R^+, \varphi\rangle$ with respect to an assignment β is given by

$$\|\langle R^+, \varphi\rangle\|_\beta = \{\alpha(R^+) \mid \alpha \models \varphi, \beta \le \alpha, \mathrm{Var}(R^+) \subseteq \mathrm{Dom}(\alpha)\}.$$

The set represented by a constrained table in this paper is always finite. Thus the finite domain property for each variable is (explicitly or implicitly) described in φ, or somewhere outside as a context. For the latter case, β is used to guarantee the finiteness of such variables by fixing their values. We abbreviate the assignment in the suffix of $\|\ \|$ if it is an empty assignment.

In similar to constrained table, we use a constrained integer (or Boolean) value for representing a set of values of the respective type. A *constrained integer (resp. Boolean) value* is a pair $\langle x, \varphi\rangle$ of a variable and a constraint, which represents the following set of values.

$$\|\langle x, \varphi\rangle\|_\beta = \{\alpha(x) \mid \alpha \models \varphi, \beta \le \alpha, x \in \mathrm{Dom}(\alpha)\}$$

We use the term *constrained values* to refer constrained table, Boolean values, or integer values.

5.2 Operations on constrained values

We now extend SQL operations on constrained values, which are used to define the transformation in Section 5.3. These extended operations assume a fixed set W of variables, whose values are already fixed in a context outside when using these operations.

We extend operations Q to Q^W on constrained values $\mathcal{R}_i$ to work correctly for any fixed assignment β with $\mathrm{Dom}(\beta) = W$:

$$\begin{aligned}&\|Q^W(\mathcal{R}_1, \ldots, \mathcal{R}_n)\|_\beta\\ &= \{Q(t_1, \ldots, t_n) \mid t_i \in \|\mathcal{R}_i\|_\beta, i = 1, \ldots, n\},\end{aligned}$$

Note that this complex treatment of β is necessary in order to synchronize assignments in $\mathcal{R}_1, \ldots, \mathcal{R}_n$, because of a search space is given in the form of $\langle\langle R_1^+, \ldots, R_n^+\rangle, \varphi\rangle$ for $\mathcal{R}_i = \langle R_i^+, \varphi_i\rangle$. (See the following subsection.)

Now we give an extended operations. Through the definition, introduced variables are fresh and also not in W. For each operator having more or equal to two constrained tables as its arguments, we can assume that common variables among such constrained tables are all in W, which is possible by renaming variables (not in W) in the constrained tables.

Boolean operators. The extension for Boolean queries is designed so that it returns a constrained Boolean value.

(1) $E^W = \langle x, x \Leftrightarrow E\rangle$, for a Boolean constant E.
(2) $\mathsf{NOT}^W \langle y, \varphi\rangle = \langle x, \varphi \wedge (x \Leftrightarrow \neg y)\rangle$.
(3) $\langle y_1, \varphi_1\rangle\ \mathit{bop}^W\ \langle y_2, \varphi_2\rangle = \langle x, \varphi_1 \wedge \varphi_2 \wedge \psi\rangle$, where ψ is $x \Leftrightarrow y_1\ \mathit{bop}\ y_2$, for a Boolean operator *bop*.
(4) $\langle k_1, \varphi_1\rangle\ \mathit{cop}^W\ \langle k_2, \varphi_2\rangle = \langle x, \varphi_1 \wedge \varphi_2 \wedge \psi\rangle$, where ψ is $x \Leftrightarrow k_1\ \mathit{cop}\ k_2$, for a comparison operator *cop*.
(5) $\mathsf{EXISTS}^W \langle R^+, \varphi\rangle = \langle x, \varphi \wedge \psi\rangle$, where ψ is $x \Leftrightarrow \bigvee_{r \in R^+} \mathtt{ext}(r)$.

Integer operations.

(1) $E^W = \langle x, x = E\rangle$, for an integer constant E.
(2) $\langle y_1, \varphi_1\rangle\ \mathit{iop}^W\ \langle y_2, \varphi_2\rangle = \langle x, \varphi_1 \wedge \varphi_2 \wedge \psi\rangle$, where ψ is $x = y_1\ \mathit{iop}\ y_2$, for an integer operator *iop*.
(3) $\mathsf{SELECT}^W a(c)\ \mathsf{FROM}\ \langle R^+, \varphi\rangle = \langle x, \varphi \wedge \psi\rangle$, where ψ is a constraint $x = \bar{a}\{\mathrm{if}(\mathtt{ext}(r), c(r), \mathrm{id}_a) \mid r \in R^+\}$, and $\mathrm{if}(e_1, e_2, e_3)$ denotes $(e_1 \Rightarrow e_2) \wedge (\neg e_1 \Rightarrow e_3)$, id_a is the identity element of function $\bar{a}$.

Note that the average function has no identity element, but can be rephrased as $\overline{\mathrm{ave}}(A) = \Sigma(A)/|A|$.

Table queries.

(1) $R^W = \langle R^+, \mathrm{true}\rangle$ for an ordinary table R, where R^+ is obtained from R by adding a column name `ext` in which all values are true.
(2) $\langle R_1^+, \varphi_1\rangle$ $\mathsf{UNION\ ALL}^W$ $\langle R_2^+, \varphi_2\rangle = \langle R_1^+ \uplus R_2^+, \varphi_1 \wedge \varphi_2\rangle$.
(3) $\langle R_1^+, \varphi_1\rangle$ $\mathsf{CROSS\ JOIN}^W$ $\langle R_2^+, \varphi_2\rangle = \langle R^+, \varphi_1 \wedge \varphi_2 \wedge \psi\rangle$, where R^+ is an extended table obtained from S^+ by removing columns $\mathtt{ext}_1$ and $\mathtt{ext}_2$,

$$\begin{aligned}\psi &= \textstyle\bigwedge_{r \in S^+} (\mathtt{ext}(r) \Leftrightarrow \mathtt{ext}_1(r) \wedge \mathtt{ext}_2(r)),\\ S^+ &= \{\!\{r_1 \cup r_2 \cup \{\mathtt{ext} = x_{r_1, r_2}\} \mid r_1 \in R_1^+,\ r_2 \in R_2^+\}\!\},\end{aligned}$$

assuming the column name `ext` in R_i is renamed as $\mathtt{ext}_i$ for $i = 1, 2$, and that $x_{r_1 r_2}$ is a variables freshly introduce for each pair of r_1 and r_2.
(4) $\mathsf{SELECT}^W *\ \mathsf{FROM}\ \langle R_1^+, \varphi\rangle\ \mathsf{AS}\ \mathit{tn}\ \mathsf{WHERE}\ \mathit{const} = \langle R^+, \varphi \wedge \psi\rangle$, where R^+ is obtained from S^+ by removing column $\mathtt{ext}_1$,

$$\begin{aligned}\psi &= \textstyle\bigwedge_{r \in S^+} (\mathtt{ext}(r) \Leftrightarrow \mathtt{ext}_1(r) \wedge \overline{\mathit{const}}^{\,r}),\\ S^+ &= \{\!\{r_1 \cup \{\mathtt{ext} = x_{r_1}\} \mid r_1 \in R_1^+\}\!\},\end{aligned}$$

assuming the column name `ext` in R_1 is renamed as $\mathtt{ext}_1$, and that x_{r_1} is a variables freshly introduce for each r_1. $\overline{\mathit{const}}^{\,r}$ is an expression $\theta(\mathit{const})$ obtained from *const* by applying the substitution $\theta = [\mathit{tn}.c \leftarrow c(r) \mid c \in \mathrm{Col}(r)]$.
(5) $\mathsf{SELECT}^W c_1, \ldots, c_n\ \mathsf{FROM}\ \langle R^+, \varphi\rangle = \langle S^+, \varphi\rangle$ where S^+ is

$$\{\!\{\langle c_1 = c_1(r), \ldots, c_n = c_n(r), \mathtt{ext} = \mathtt{ext}(r)\rangle \mid r \in R^+\}\!\}$$

(6) $\mathsf{SELECT}^W c, a(d)\ \mathsf{AS}\ \mathit{dn}\ \mathsf{FROM}\ \langle R^+, \varphi\rangle\ \mathsf{GROUP\ BY}\ c$ $= \langle S^+, \varphi \wedge \psi\rangle$, where S^+ and ψ are defined as follows. Suppose $c(\langle R^+, \varphi\rangle)$ represents a finite set that covers the image of $\langle R^+, \varphi\rangle$, that is, all possible values for the column c determined from $\langle R^+, \varphi\rangle$.

$$S^+ = \{\!\{\{c = v, \mathit{dn} = x_v, \mathit{ext} = y_v\} \mid v \in c(\langle R^+, \varphi\rangle)\}\!\}$$

where x_v,y_v are variables freshly introduced for each v.

$$\begin{aligned}\psi = &\bigwedge_{v \in c(\langle R^+, \varphi\rangle)} \left(y_v \Leftrightarrow \left(\bigvee_{r \in R^+} (\mathtt{ext}(r) \wedge d(r) = v)\right)\right)\\ &\wedge \bigwedge_{v \in c(\langle R^+, \varphi\rangle)} \left(x_v = \mathit{exp}_v\right)\end{aligned}$$

and exp_v is

$$\bar{a}\{\mathrm{if}(\mathtt{ext}(r) \wedge c(r) = v, d(r), \mathrm{id}_a) \mid r \in R^+\}$$

Remark that subqueries may appear in WHERE constraints, whose treatment is not described in the above definition for simplicity. Subqueries can be processed recursively by attaching the following glue. Suppose $\langle R^+, \varphi\rangle$ is a resulted constrained table for a subquery, where $R^+ = \{r_1, \ldots, r_n\}$ with columns c and ext. Then the value extracted from the first record is defined as $\langle z, \psi\rangle$, where ψ is

$$\begin{array}{rl} z \Leftrightarrow & \mathrm{if}(\mathtt{ext}(r_1), c(r_1), \\ & \quad \mathrm{if}(\mathtt{ext}(r_2), c(r_2), \ \ldots \\ & \quad\quad \mathrm{if}(\mathtt{ext}(r_{n-1}), c(r_{n-1}), c(r_n)) \cdots)) \\ & \wedge \bigvee_{i=\{1,\ldots,n\}} \mathtt{ext}(r_i). \end{array}$$

Theorem 5.3. *Suppose $Q(t_1, \ldots, t_n)$ a Boolean/integer/table query where each bound variable t_i representing a table is explicitly displayed, and the set W of variables appearing in the search space $\langle\langle R_1^+, \ldots, R_n^+\rangle, \varphi\rangle$. For an assignment β such that $\mathrm{Dom}(\beta) = W$,*

$$\begin{array}{l} \|Q^W(\langle R_1^+, \mathrm{true}\rangle, \ldots, \langle R_n^+, \mathrm{true}\rangle)\|_\beta \\ = \{Q(t_1, \ldots, t_n) \mid t_i \in \|\langle R_i^+, \mathrm{true}\rangle\|_\beta, i = 1, \ldots, n\}, \end{array}$$

The proof is found in the appendix.

5.3 Producing CP/SMT constraints

We show how CP/SMT constraints are produced by using the extended SQL operations introduced in the previous subsection.

Generation. Suppose a *choose_query*

SUBTABLE OF (SELECT $c_1, \ldots, c_n$ FROM R)

where each c_i is either a column name of the table R or a term in the form of CHOOSE(S_i) with a single columned table S_i having a column name s_i. This query is converted to the constrained table $\langle R^+, \varphi\rangle$ with

$$\begin{array}{rl} R^+ = & \{\!\{\langle F_1 = D_1, \ldots, F_n = D_n(r), \mathtt{ext} = y^r\rangle \\ & \quad \mid r \in R, \\ & \quad\ F_i = c_i \text{ and } D_i = c_i(r) \text{ if } c_i \text{ is a column name of } R, \\ & \quad\ F_i = s_i \text{ and } D_i = x_i^r \text{ otherwise } \}\!\}, \\ \varphi = & \bigwedge_{r\in R} \left(y^r \in \mathbb{B} \wedge \bigwedge_{\substack{1\le i\le n,\\ c_i = \mathrm{CHOOSE}(S_i)}} x_i^r \in \mathrm{Img}(S_i) \right), \end{array}$$

where each x_i^r (and each y^r) is a freshly introduced variable.

In the case that the query has no SUBTABLE OF, it is enough to replace $y^r \in \mathbb{B}$ with $y^r = \mathrm{true}$ in the definition of φ.

Lemma 5.4. *For constrained table $\langle R^+, \varphi\rangle$ produced from a choose-query, the set $\|\langle R^+, \varphi\rangle\|$ is equal to the expected set of tables.*

Proof. We consider the following simple *choose_query*:

SUBTABLE OF (SELECT c, CHOOSE(S) FROM R),

and show that $T \in \|\langle R^+, \varphi\rangle\|$ if and only if

$$T \subseteq \{\!\{\{c = c(r), s = f(r)\} \mid r \in R\}\!\} \text{ for some } f : R \to \mathrm{Img}(S),$$

where

$$\begin{array}{rl} R^+ = & \{\!\{\langle c = c(r), s = x^r, \mathtt{ext} = y^r\rangle \mid r \in R\}\!\}, \\ \varphi = & \bigwedge_{r\in R} (y^r \in \mathbb{B} \wedge x^r \in \mathrm{Img}(S)). \end{array}$$

(only-if): Assume $T \in \|\langle R^+, \varphi\rangle\|$. Then, there exists an assignment α such that $\alpha \models \varphi$ and $T = \alpha(R^+)$. Here $\alpha(R^+) = \{\!\{\{c = c(r), s = \alpha(x^r)\}\!\} \mid r \in R, \alpha(y^r) = \mathrm{true}\}$. Since $\alpha \models \varphi$, each $\alpha(x^r) \in \mathrm{Img}(S)$. By taking $f(r) = \alpha(x^r)$ for $r \in R$, this direction is shown.

(if): Assume $T \subseteq \{\!\{\{c = c(r), s = f(r)\} \mid r \in R\}\!\}$ for some $f : R \to \mathrm{Img}(S)$. We take an assignment α such that for for $r \in R$

- $\alpha(x^r) = f(r)$, and
- $\alpha(y^r) = \mathit{true}$ if and only if $\{c = c(r), s = f(r)\}$ is in T.

Then, $\alpha \models \varphi$, and hence $T \in \|\langle R^+, \varphi\rangle\|$.

A simpler case having no SUBTABLE OF is shown similarly. □

Let Q_i be an ordinary SQL query that returns a table, and $\langle S_i^+, \varphi_i\rangle$ be a constrained table created from *choose_query*, where we can assume the sets of variables for each constrained table are pairwise disjoint. Suppose a creating set statement

CREATE SET *sname*
HAS ($Q_1(t_1, \ldots, t_n), \ldots, Q_m(t_1, \ldots, t_n)$)
t_1 IN FOR $\langle S_1^+, \psi_1\rangle$, …, t_n IN $\langle S_n^+, \psi_n\rangle$.

This statement binds *sname* to the constrained table $\langle\langle R_1^+, \ldots, R_m^+\rangle, \varphi\rangle$ generated in the following way. Assume that W is the set of variables used in $\langle S_1^+, \psi_1\rangle, \ldots, \langle S_n^+, \psi_n\rangle$. Let $\langle R_j^+, \varphi_j\rangle$ be the constraint table $Q_j^W(\langle S_1^+, \mathrm{true}\rangle, \ldots, \langle S_n^+, \mathrm{true}\rangle)$ ($j = 1, \ldots, m$), where we can assume the sets of variables in $\langle R_j^+, \varphi_j\rangle$ are pairwise disjoint except for those in W. Then, this create-set statement is converted to $\langle\langle R_1^+, \ldots, R_m^+\rangle, \varphi\rangle$, where φ is

$$\bigwedge_{i=1,\ldots,n} \psi_i \wedge \bigwedge_{j=1,\ldots,m} \varphi_j.$$

Lemma 5.5. *For a constrained table $\langle\langle R_1^+, \ldots, R_m^+\rangle, \varphi\rangle$ produced from a create-set statement, the set $\|\langle\langle R_1^+, \ldots, R_m^+\rangle, \varphi\rangle\|$ is equal to the expected set of (tuples of) tables.*

Proof. We consider the following simple statement:

CREATE SET *sname* HAS $Q(t)$ FOR t IN $\langle S^+, \psi\rangle$,

and show that

$$\|\langle R^+, \psi \wedge \varphi\rangle\| = \{Q(t) \mid t \in \|\langle S^+, \psi\rangle\|\},$$

where $\langle R^+, \varphi\rangle$ indicates $Q^W(\langle S^+, \mathrm{true}\rangle)$.

($\subseteq$): Let a table R be in the left hand side. Then there exists an assignment α such that $\alpha \models \psi \wedge \varphi$ and $R = \alpha(R^+)$. Let $\beta = \alpha|_W$. Then, $\beta \preceq \alpha$, $\alpha \models \varphi$ and also $R \in \|\langle R^+, \varphi\rangle\|_\beta = \|Q^W(\langle S^+, \mathrm{true}\rangle)\|_\beta$. By Theorem 5.3, $R \in \|\langle R^+, \varphi\rangle\|_\beta = \{Q(t) \mid t \in \|\langle S^+, \mathrm{true}\rangle\|_\beta\}$. Since $\|\langle S^+, \mathrm{true}\rangle\|_\beta \subseteq \|\langle S^+, \psi\rangle\|$ from $\beta \preceq \alpha$ and $\alpha \models \psi$, the table R is in the right hand side.

($\supseteq$): Let a table R be in the right hand side. Then there exists a table $S \in \|\langle S^+, \psi\rangle\|$ such that $R = Q(S)$. Thus, there exists an assignment β satisfying $\beta \models \psi$ and $S = \beta(S^+)$, where $\mathrm{Dom}(\beta) = W$, and hence $\|\langle S^+, \psi\rangle\| = \|\langle S^+, \mathrm{true}\rangle\|_\beta$. By Theorem 5.3, $\|\langle R^+, \varphi\rangle\|_\beta = \{Q(t) \mid t \in \|\langle S^+, \mathrm{true}\rangle\|_\beta\}$, hence $R \in \|\langle R^+, \varphi\rangle\|_\beta$. From this, there exists α such that $\alpha \models \varphi$, $\beta \preceq \alpha$, and $R = \alpha(R^+)$. Therefore, R is in the left hand side $\|\langle R^+, \psi \wedge \varphi\rangle\|$. □

Filtering. Suppose a filtering statement

CREATE SET *sname* HAS ($t_1, \ldots, t_n$) IN $\langle\langle R_1^+, \ldots, R_n^+\rangle, \psi\rangle$
SUCH THAT $Q(t_1, \ldots, t_n)$.

This statement binds *sname* to the constrained table

$$\langle\langle R_1^+, \ldots, R_n^+\rangle, \psi \wedge \varphi \wedge x\rangle$$

where $\langle x, \varphi\rangle$ indicates $Q^W(\langle R_1^+, \text{true}\rangle, \ldots, \langle R_n^+, \text{true}\rangle)$ for the set W of variables used in $\langle\langle R_1^+, \ldots, R_n^+\rangle, \psi\rangle$.

Lemma 5.6. *For a constrained table $\mathcal{R}$ produced from a filtering statement, the set $\|\mathcal{R}\|$ is equal to the expected set of (tuples of) tables.*

Proof. Supposing a simple statement

$$\text{CREATE SET } sname \text{ HAS } t \text{ IN } \langle R^+, \psi\rangle \text{ SUCH THAT } Q(t),$$

we show that $\|\langle R^+, \psi \wedge \varphi \wedge x\rangle\| = \{t \mid t \in \|\langle R^+, \psi\rangle\|, Q(t)\}$, where $\langle x, \varphi\rangle$ is $Q^W(\langle R^+, \text{true}\rangle)$ for the set W of variables used in $\langle R^+, \psi\rangle$.

($\subseteq$): Let a table R be in the left hand side. Then there exists an assignment α such that $\alpha \models \psi \wedge \varphi \wedge x$ and $R = \alpha(R^+)$. Let $\beta = \alpha|_W$ then $\beta \models \psi$ and also $R \in \|\langle R^+, \psi\rangle\|$. On the other hand, from $\beta \leq \alpha$, $\alpha \models \varphi$, and also true $\in \|\langle x, \varphi\rangle\|_\beta = \|Q^W(\langle R^+, \text{true}\rangle)\|_\beta$. By Theorem 5.3, true $\in \{Q(t) \mid t \in \|\langle R^+, \text{true}\rangle\|_\beta\}$. Since $R \in \|\langle R^+, \text{true}\rangle\|_\beta \subseteq \|\langle R^+, \psi\rangle\|$ from $\beta \leq \alpha$ and $\alpha \models \psi$, we obtain $Q(R)$. Therefore, the table R is in the right hand side.

($\supseteq$): Let a table R be in the right hand side. Then, there exists a table $R \in \|\langle R^+, \psi\rangle\|$ such that $Q(R)$. From the former, there exists an assignment β satisfying $\beta \models \psi$ and $R = \beta(R^+)$, where $\text{Dom}(\beta) = W$, and hence $R \in \|\langle R^+, \psi\rangle\| = \|\langle R^+, \text{true}\rangle\|_\beta$. Since $\|Q^W(\langle R^+, \text{true}\rangle)\|_\beta = \{Q(t) \mid t \in \|\langle R^+, \text{true}\rangle\|_\beta\}$ by Theorem 5.3, we have true $\in \|\langle x, \varphi\rangle\|_\beta = \|Q^W(\langle R^+, \text{true}\rangle)\|_\beta$. From this, there exists α such that $\beta \leq \alpha$, true $= \alpha(x)$, and $\alpha \models \varphi$, which means that $\alpha \models x$. Since $\beta \leq \alpha$, we obtain $R = \alpha(R^+)$, $\alpha \models \psi \wedge \varphi \wedge x$. Therefore, R is in the left hand side $\|\langle R^+, \psi \wedge \varphi \wedge x\rangle\|$. □

Selection. This step applies a CP/SMT solver to constraints obtained in the previous step with transformation for a goal function. Suppose a statement

$$\text{CREATE TABLE } tn_1, \ldots, tn_n \text{ AS } (t_1, \ldots, t_n) \text{ IN } \langle\langle R_1^+, \ldots, R_n^+\rangle, \psi\rangle$$
$$\text{MAXIMIZING } Q(t_1, \ldots, t_n)$$

For this statement, $\psi \wedge \varphi$ is given to a CP/SMT solver as constraints with the goal function 'maximizing x', where $\langle x, \varphi\rangle$ indicates $Q^W(\langle R_1^+, \text{true}\rangle, \ldots, \langle R_n^+, \text{true}\rangle)$. If $\psi \wedge \varphi$ is satisfiable and an assignment α causes this optimum goal value, then an optimal solution $(tn_1, \ldots, tn_n)$ is obtained as $(\alpha(R_1^+), \ldots, \alpha(R_n^+))$; if $\psi \wedge \varphi$ is not satisfiable then it is known that no solution exists. This is justified from (2) of the following Lemma.

Lemma 5.7. *Consider a search space $\langle\langle R_1^+, \ldots, R_n^+\rangle, \psi\rangle$, and $\langle x, \phi\rangle = Q^W(\langle R_1^+, \text{true}\rangle, \ldots, \langle R_n^+, \text{true}\rangle)$ for the set W of variables that appear in $\langle\langle R_1^+, \ldots, R_n^+\rangle, \psi\rangle$,*

(1) *The following statements are equivalent.*
  (a) $(R_1, \ldots, R_n) \in \|\langle\langle R_1^+, \ldots, R_n^+\rangle, \psi\rangle\|$ *and* $k = Q(R_1, \ldots, R_n)$.
  (b) $\alpha \models \psi \wedge \varphi$, $R_i = \alpha(R_i^+)$ *and* $k = \|\langle x, \psi \wedge \varphi\rangle\|$ *for some assignment* α.

(2) *The following statements are equivalent.*
  (a) $(R_1, \ldots, R_n) \in \|\langle\langle R_1^+, \ldots, R_n^+\rangle, \psi\rangle\|$ *and*

$$Q(R_1, \ldots, R_n) = \max\{Q(S_1, \ldots, S_n) \mid (S_1, \ldots, S_n) \in \|\langle\langle R_1^+, \ldots, R_n^+\rangle, \psi\rangle\|\}.$$

  (b) $\alpha \models \psi \wedge \varphi$ *and*

$$Q(\alpha(R_1^+), \ldots, \alpha(R_n^+)) = \max\{\alpha'(x) \mid \alpha' \models \psi \wedge \varphi\}$$

  for some assignment α.

Proof. Suppose $Q^W(\langle R_1^+, \text{true}\rangle, \ldots, \langle R_n^+, \text{true}\rangle)$ is transformed to $\langle x, \psi\rangle$. Then, by Theorem 5.3

$$\|Q^W(\langle R_1^+, \text{true}\rangle, \ldots, \langle R_n^+, \text{true}\rangle)\|_\beta = \{Q(t_1, \ldots, t_n) \mid t_1 \in \|\langle R_i^+, \text{true}\rangle\|_\beta\} \quad (*)$$

(1) Assume (a). Then there exists an assignment β such that $R_i = \beta(R_i^+)$, $\beta \models \psi$, and $\text{Dom}(\beta) = W$. We obtain

$$(R_1, \ldots, R_n) \in \|\langle\langle R_1^+, \ldots, R_n^+\rangle, \text{true}\rangle\|_\beta.$$

It follows that $R_i \in \|\langle R_i^+, \text{true}\rangle\|_\beta$ for each i. We can obtain $k \in \|\langle x, \phi\rangle\|_\beta$ from (*), and hence there exist α such that $k = \alpha(x)$, $\alpha \models \phi$, and $\beta \leq \alpha$. Thus $k \in \|\langle x, \phi \wedge \varphi\rangle\|$ as well as $\alpha \models \psi \wedge \varphi$ and $R_i = \alpha(R_i^+)$. Therefore (b) holds.

Assume (b). Then $(R_1, \ldots, R_n) \in \|\langle\langle R_1^+, \ldots, R_n^+\rangle, \psi\rangle\|$ follows directly, and there exists α such that $k = \alpha(x)$, $\alpha \models \psi \wedge \varphi$. Let $\beta = \alpha|_W$. Then we can show that k is in the left hand side of (*). (a) follows, since $k = \alpha(x)$, $\alpha \models \phi$, and $\beta \leq \alpha$.

(2) is shown by using (1). □

In the simpler case that a statement without goal function, it is enough to check the satisfiability of ψ, whose correctness are induced from (1) of Lemma 5.7

5.4 Example of transformation

Consider the description shown in Figure 5 and an instance in Table 5. Here, Table 6(a) represents `SearchSp` by the integer variables $x_a, \ldots, x_d$. For a simple presentation, we assume that the subqueries `lookup` in the line 9 of Figure 5 are executed as an ordinary SQL, which is much simpler than transformed according to the definition. Let's explain more precisely. By the definition, the former subquery is expanded to

$$\text{SELECT}^W \text{ c FROM } \langle R^+, \varphi\rangle \text{ WHERE E.v1} = \text{v}.$$

Intuitively, the subquery returns the color $x_{\text{E.v1}}$ of the vertex E.v1. Since all value of the column `ext` is true and column v has concrete values a,. . .,d, the subquery can be replaced with

$$\text{SELECT c FROM } R^+ \text{ WHERE E.v1} = \text{v}.$$

This simple treatment is possible. In really, we employed this in the improved implementation in such possible cases. The transformed constrained table from lines 8–10 is $\langle S^+, \varphi_S\rangle$ illustrated in Table 6(b). Then, by the NOT EXISTS constraint in line 7, the following constrained table $\langle z, \varphi_z\rangle$ is produced, where φ_z is $(z \Leftrightarrow \neg w) \wedge (w \Leftrightarrow y_1 \vee y_2 \vee y_3 \vee y_4)$. Therefore the filtering step produces a constraint table $\langle R^+, \varphi_F\rangle$, where φ_F is $\varphi_R \wedge \varphi_S \wedge \varphi_z \wedge z$.

Table 5: Tables for coloring problem

(a) Vertices (V)

v
a
b
c
d

(b) Edges (E)

v1	v2
a	b
a	c
b	c
c	d

(c) Colors (CL)

c
1
2
3
4

Table 6: Constrained tables

(a) $\langle R^+, \varphi_R \rangle$ **for** `SearchSp`

$R^+ =$

v	c	ext
a	x_a	true
b	x_b	true
c	x_c	true
d	x_d	true

$\varphi_R =$
$x_a, x_b, x_c, x_d \in \{1, 2, 3, 4\}$

(b) $\langle S^+, \varphi_S \rangle$ **for lines 8–10 in Figure 5**

$S^+ =$

v1	v2	ext
a	b	y_1
a	c	y_2
b	c	y_3
c	d	y_4

$\varphi_S = y_1 \Leftrightarrow x_a = x_b \wedge y_2 \Leftrightarrow x_a = x_c \wedge$
$y_3 \Leftrightarrow x_b = x_c \wedge y_4 \Leftrightarrow x_c = x_d$

In the selection step, the goal function in line 12 produces a constrained table $\langle k, \varphi_k \rangle$, where φ_k is $k = \max\{x_a, x_b, x_c, x_d\}$. Finally, it produces a CP/SMT instance: $\varphi_F \wedge \varphi_k$ with minimizing k.

6 IMPLEMENTATION ISSUES

We are working on an implementation accepting CombSQL+ presented in Section 5, and have partially completed. We use Ruby in the implementation, and follow SQL 92 [5] as a base SQL language. The overview is shown in Figure 8, where SQLite is a lightweight

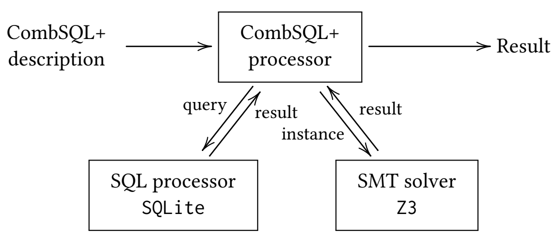

Figure 8: Overview of an implementation

database, which is executable as a standalone application [9]. Z3 [1] is one of famous SMT solvers, which accepts logical formulas over theories and an optimization function.

In order to generate a simpler constraints, we used ordinary SQL processors, if possible as in the example in Section 5.4, in implementing operators on constrained values. For example,

$$\mathsf{SELECT}^W\ columns\ \mathsf{FROM}\ \langle R^+, \varphi \rangle$$

can always be represented as

$$\langle \mathsf{SELECT}\ columns \mathsf{FROM}\ R^+, \varphi \rangle$$

by regarding a given extended table R^+ as an ordinary table with `text` type column `ext`. The similar calculation is possible for SELECT queries if the WHERE constrains contain no constrained tables. Such treatment is extremely effective to reduce the size of CP/SMT constraints.

We prepared an operation INSERT as shown in lines 4–6 in Figure 9, which reads records from csv-files.

We experimented on graph vertex coloring problem with minimizing the kinds k of used colors. We have shown the problem description in written in CombSQL+ in Section 4.1. The used instances are selected from a benchmark [18] so that the sizes are at most 561 vertices and 6656 edges and also they are solvable by

```
1 CREATE TABLE V  (  v      TEXT );
2 CREATE TABLE E  (  v1,v2  TEXT );
3 CREATE TABLE CL (  c      INT  );
4 INSERT INTO V   FILE "Vertices.csv";
5 INSERT INTO E   FILE "Edges.csv";
6 INSERT INTO CL  FILE "ColorList.csv";
```

Figure 9: Reading data from csv-files

Z3 within one hour via the hand-coded translator. The result is shown in Table 7. Here the first three columns are statistics of the used instances: name, number of vertices, and number of edges. The other columns are the results: the minimum k obtained, time in seconds to solve by our implementation 'Ours–' that strictly follows the transformation presented in Section 5, time in seconds to solve by our implementation 'Ours' that improved to reduce unnecessary constraints, and time in seconds to solve via hand-coded translator that produce a reasonable constraints for Z3 from an input graph.

It appears that 'Ours' and hand-coded are competitive. The translator 'Ours–' spent most of the time for creating and sending constraints to Z3, not for solving in Z3. Our improved translator generated twice in number of constraints against hand-coded one, and created fresh variables whose number is almost the same as the number of produced constraints.

As far as this experiment, it seems that the efficiency of our improved translator depends on the backend CP/SMT solver.

Table 7: Experiment on minimizing colors for graph vertices

Instances	Vert.	Edg.	k	Ours–	Ours	HandCd.
myciel3	11	20	4	0.4	0.2	0.1
myciel4	23	71	5	3.4	1.5	1.0
queen5_5	25	160	5	13.7	0.5	0.2
queen6_6	36	290	7	122.	58.9	84.8
queen7_7	49	476	7	182.	5.6	5.8
miles250	128	387	8	1456.	26.5	29.8
games120	120	638	9	(3600.<)	12.6	114.

(in seconds)

7 CONCLUDING REMARKS

We have proposed an extension of the SQL language for describing optimization problems, and a transformation scheme of descriptions and instance tables into CP/SMT constraints. We have completed the implementation except for GROUP BY. From the current knowledge, the performance is competitive against hand-coded translator, and depends on the backend CP/SMT solver. Thus, it is promising to incorporate CP-specific efficient operations, such as all-different constraints, into CombSQL+, which will accelerate its efficiency.

A PROOF OF THEOREM 5.3

We provide a technical lemma.

Lemma A.1. *For an extended table R^+ and an assignment α,*

$$\mathit{SELECT}\ a(c)\ \mathit{FROM}\ \alpha(R^+)$$
$$= \bar{a}\{if(\alpha(\mathrm{ext}(r)), \alpha(c(r)), \mathrm{id}_a) \mid r \in R^+\}$$

Proof. $\alpha(R^+)$ is

$$\{\!\{\{c_1 = \alpha(d_1), \ldots, c_n = \alpha(d_n)\} \mid \{c_1 = d_1, \ldots, c_n = d_1, \mathtt{ext} = y\} \in R^+, \alpha(y) = \mathrm{true}\}\!\}.$$

Hence, "SELECT $a(c)$ FROM $\alpha(R^+)$" is

$$\bar{a}\{\alpha(c(r)) \mid r \in R^+, \alpha(\mathtt{ext}(r)) = \mathrm{true}\}$$

Since id_a is identity element of a, this is equal to

$$\bar{a}\{if(\alpha(\mathtt{ext}(r)), \alpha(c(r)), \mathrm{id}_a) \mid r \in R^+\}$$ □

We show that a general form of Theorem 5.3, that is

$$\|Q^W(\langle R_1^+, \varphi_1\rangle, \ldots, \langle R_n^+, \varphi_n\rangle)\|_\beta = \{Q(t_1, \ldots, t_n) \mid t_i \in \|\langle R_i^+, \varphi_i\rangle\|_\beta, i = 1, \ldots, n\}.$$

We indicate this for the simple case that Q consists of a single operator. The case for complex queries is easily derived by induction on the structure of the query Q.

Boolean operators. (1) $\|E^W\|_\beta = \{\alpha(x) \mid \alpha \models (x \Leftrightarrow E), \beta \le \alpha\} = \{E\}$.

(2) We show $\|\mathrm{NOT}^W\langle y, \varphi\rangle\|_\beta = \{\mathrm{NOT}\ v \mid v \in \|\langle y, \varphi\rangle\|_\beta\}$.

($\subseteq$) For $b \in \|\langle x, \varphi \wedge (x \Leftrightarrow \neg y)\rangle\|_\beta$, there exists an assignment α such that $\alpha(x) = b$, $\alpha \models (\varphi \wedge (x \Leftrightarrow \neg y))$, and $\beta \le \alpha$. Since $\alpha \models (x \Leftrightarrow \neg y)$, it holds that $b \Leftrightarrow \neg\alpha(y)$. Therefore b is also in the right hand side.

($\supseteq$) For $b \in \mathbb{B}$ in the right hand side, there exists an assignment α such that $\alpha(y) = \neg b$, $\alpha \models \varphi$, and $\beta \le \alpha$. Letting $\alpha' = \alpha \cup [x \leftarrow b]$, we obtain $\alpha' \models (\varphi \wedge (x \Leftrightarrow \neg y))$ and $\beta \le \alpha \le \alpha'$. Therefore b is also in $\|\langle x, \varphi \wedge (x \Leftrightarrow \neg y)\rangle\|_\beta$.

(3) We show $\|\langle y_1, \varphi_1\rangle\ bop^W\ \langle y_2, \varphi_2\rangle\|_\beta = \{b_1\ bop\ b_2 \mid b_i \in \|\langle y_i, \varphi_i\rangle\|_\beta, i = 1, 2\}$. Let ψ be $x \Leftrightarrow y_1\ bop\ y_2$.

($\subseteq$) For $b \in \|\langle x, \varphi_1 \wedge \varphi_2 \wedge \psi\rangle\|_\beta$, there exists an assignment α such that $\alpha(x) = b$, $\alpha \models \varphi_1 \wedge \varphi_2 \wedge \psi$, and $\beta \le \alpha$. Since $\alpha \models \psi$, we have $b \Leftrightarrow \alpha(y_1)\ bop\ \alpha(y_2)$. Therefore b is in the right hand side.

($\supseteq$) For $b \in \mathbb{B}$ in the right hand side, there exists an assignment α_i ($i = 1, 2$) such that $b \Leftrightarrow \alpha(y_1)\ bop\ \alpha(y_2)$, $\alpha_i \models \varphi_i$, and $\beta \le \alpha_i$. From the construction, common variables in domains $\mathrm{Dom}(\alpha_i)$ ($i = 1, 2$) are in W, hence $\alpha = \alpha_1 \cup \alpha_2 \cup [x \leftarrow b]$ is well defined. Thus, we obtain $\alpha \models \varphi \wedge \psi$ and $\beta \le \alpha$. Therefore b is also in $\|\langle x, \varphi_1 \wedge \varphi_2 \wedge \psi\rangle\|_\beta$.

(4) We show

$$\|\langle y_1, \varphi_1\rangle\ cop^W\ \langle y_2, \varphi_2\rangle\|_\beta = \{k_1\ cop\ k_2 \mid k_i \in \|\langle y_i, \varphi_i\rangle\|_\beta, i = 1, 2\}.$$

Let ψ be $x \Leftrightarrow y_1\ cop\ y_2$.

($\subseteq$) For $b \in \|\langle x, \varphi_1 \wedge \varphi_2 \wedge \psi\rangle\|_\beta$, there exists an assignment α such that $\alpha(x) = b$, $\alpha \models \varphi_1 \wedge \varphi_2 \wedge \psi$, and $\beta \le \alpha$. Since $\alpha \models \psi$, we have $b \Leftrightarrow \alpha(y_1)\ cop\ \alpha(y_2)$. Therefore b is in the right hand side.

($\supseteq$) For $b \in \mathbb{B}$ in the right hand side, there exists an assignment α_i ($i = 1, 2$) such that $b \Leftrightarrow \alpha(y_1)\ cop\ \alpha(y_2)$, $\alpha_i \models \varphi_i$, and $\beta \le \alpha_i$. From the construction, common variables in domains $\mathrm{Dom}(\alpha_i)$ ($i = 1, 2$) are in W, hence $\alpha = \alpha_1 \cup \alpha_2 \cup [x \leftarrow b]$ is well defined. Thus, we obtain $\alpha \models \varphi \wedge \psi$ and $\beta \le \alpha$. Therefore b is also in $\|\langle x, \varphi_1 \wedge \varphi_2 \wedge \psi\rangle\|_\beta$.

(5) We show $\|\mathrm{EXISTS}^W\langle R^+, \varphi\rangle\|_\beta = \{\mathrm{EXISTS}\ R \mid R \in \|\langle R^+, \varphi\rangle\|_\beta\}$. Let ψ be $x \Leftrightarrow \bigvee_{r \in R^+} \mathtt{ext}(r)$.

($\subseteq$) For $b \in \mathbb{B}$ in $\|\langle x, \varphi \wedge \psi\rangle\|_\beta$, there exists an assignment α such that $\alpha(x) = b$ and $\alpha \models \varphi \wedge \psi$. Hence $\alpha(x)$ if and only if $\alpha(\mathtt{ext}(r))$ for some $r \in R^+$.

1. If $b = \mathrm{true}$, there exists $r \in R^+$ such that $\alpha(\mathtt{ext}(r)) = \mathrm{true}$. This means that $\alpha(R^+)$ has at least one record.
2. If $b = \mathrm{false}$, there exists no $r \in R^+$ such that $\alpha(\mathtt{ext}(r)) = \mathrm{true}$. This means that $\alpha(R^+)$ has no records.

Therefore b is also in the right hand side.

($\supseteq$) For $b \in \mathbb{B}$ in the right hand side, there exists an assignment α such that EXISTS $\alpha(R^+) = b$ and $\alpha \models \varphi$. Let $\alpha' = \alpha \cup [x \leftarrow b]$. We obtain $b \in \|\langle x, \varphi \wedge \psi\rangle\|_\beta$ if $\alpha' \models \psi$.

1. If $b = \mathrm{true}$, $\alpha(R^+)$ has at least one record. This means that $\alpha(\mathtt{ext}(r)) = \mathrm{true}$ for some $r \in R^+$, hence $\alpha' \models \psi$.
2. If $b = \mathrm{false}$, $\alpha(R^+)$ has no records. This means that $\alpha(\mathtt{ext}(r)) = \mathrm{false}$ for any $r \in R^+$, hence $\alpha' \models \psi$.

Integer operators. (1) $\|E^W\|_\beta = \{\alpha(x) \mid \alpha \models (x = E), \beta \le \alpha\} = \{E\}$.

(2) We show $\|\langle y_1, \varphi_1\rangle\ iop^W\ \langle y_2, \varphi_2\rangle\|_\beta = \{k_1\ iop^W\ k_2 \mid k_i \in \|\langle k_i, \varphi_i\rangle\|_\beta, i = 1, 2\}$. Let ψ be $x = k_1\ iop\ k_2$.

($\subseteq$) For $k \in \|\langle x, \varphi_1 \wedge \varphi_2 \wedge \psi\rangle\|_\beta$, there exists an assignment α such that $\alpha(x) = k$, $\alpha \models \varphi_1 \wedge \varphi_2 \wedge \psi$, and $\beta \le \alpha$. Since $\alpha \models \psi$, we have $k = \alpha(y_1)\ iop\ \alpha(y_2)$. Therefore k is also in the right hand side.

($\supseteq$) For $k \in \mathbb{Z}$ in the right hand side, there exists an assignment α_i ($i = 1, 2$) such that $k \Leftrightarrow \alpha(y_1)\ iop\ \alpha(y_2)$, $\alpha_i \models \varphi_i$, and $\beta \le \alpha_i$. From the construction, common variables in domains $\mathrm{Dom}(\alpha_i)$ ($i = 1, 2$) are in W, hence $\alpha = \alpha_1 \cup \alpha_2 \cup [x \leftarrow k]$ is well defined. Thus, we obtain $\alpha \models \varphi \wedge \psi$ and $\beta \le \alpha$. Therefore $k \in \|\langle x, \varphi_1 \wedge \varphi_2 \wedge \psi\rangle\|_\beta$.

(3) We show $\|\mathrm{SELECT}^W a(c)\ \mathrm{FROM}\ \langle R^+, \varphi\rangle\|_\beta = \{\mathrm{SELECT}\ a(c)\ \mathrm{FROM}\ t \mid t \in \|\langle R^+, \varphi\rangle\|_\beta\}$. Let ψ be $x = \bar{a}\{\mathrm{if}(\mathtt{ext}(r), c(r), \mathrm{id}_a) \mid r \in R^+\}$.

($\subseteq$) For $k \in \|\langle x, \varphi \wedge \psi\rangle\|_\beta$, there exists an assignment α such that $\alpha(x) = k$, $\alpha \models \varphi \wedge \psi$, and $\beta \le \alpha$. Since $\alpha \models \psi$, k is equal to

$$\bar{a}\{if(\alpha(\mathtt{ext}(r)), \alpha(c(r)), \mathrm{id}_a) \mid r \in R^+\}$$

By lemma A.1, it is equal to "SELECT $a(c)$ FROM $\alpha(R^+)$". Therefore k is in the right hand side.

($\supseteq$) For $k \in \mathbb{Z}$ in the right hand side, there exists an assignment α such that $\alpha \models \varphi$ and $k =$ SELECT $a(c)$ FROM $\alpha(R^+)$, where by lemma A.1 the latter is equal to

$$\bar{a}\{if(\alpha(\mathtt{ext}(r)), \alpha(c(r)), \mathrm{id}_a) \mid r \in R^+\}.$$

Let α' be $\alpha \cup [x \leftarrow k]$. Then, $\beta \le \alpha \le \alpha'$, and $\alpha' \models \varphi \wedge \psi$ because

$$\alpha'(x) = k = \bar{a}\{if(\alpha(\mathtt{ext}(r)), \alpha(c(r)), \mathrm{id}_a) \mid r \in R^+\}.$$

Therefore $k \in \|\langle x, \varphi \wedge \psi\rangle\|_\beta$.

Table operators. (1) Follows from $\|R^W\|_\beta = \|\langle R^+, \emptyset\rangle\|_\beta$ and $\beta(R^+) = R$.

(2) We show $\|\langle R_1^+, \varphi_1\rangle\ \mathrm{UNION\ ALL}^W\langle R_2^+, \varphi_2\rangle)\|_\beta = \{t_1\ \mathrm{UNION\ ALL}\ t_2 \mid t_i \in \|\langle R_i^+, \varphi_i\rangle\|_\beta$.

($\subseteq$) For a table $R \in \|\langle R_1^+ \uplus R_2^+, \varphi_1 \wedge \varphi_2\rangle\|_\beta$, there exists an assignment α such that $R = \alpha(R_1^+ \uplus R_2^+)$, $\alpha \models \varphi_1 \wedge \varphi_2$, and $\beta \le \alpha$. Since $R = \alpha(R_1^+ \uplus R_2^+) = \alpha(R_1^+) \uplus \alpha(R_2^+)$, there exists $R_i \in \alpha(R_i^+)$ such that $R = R_1 \uplus R_2$. Here, $R_i \in \|\langle R_i^+, \varphi_i\rangle\|_\beta$. Therefore the table R is also in the right hand side.

($\supseteq$) For a table R in the right hand side, there exists assignments $R_i \in \|\langle R_i^+, \varphi_i\rangle\|_\beta$ ($i = 1, 2$) such that $R = R_1 \uplus R_2$. Here there exists α_i such that $R_i = \alpha(R_i^+)$, $\alpha_i \models \varphi_i$, and $\beta \leq \alpha_i$. From the construction, common variables in domains $\mathrm{Dom}(\alpha_i)$ are in W, hence $\alpha = \alpha_1 \cup \alpha_2$ is well defined. Thus, we obtain $\alpha \models \varphi_1 \wedge \varphi_2$, and $\beta \leq \alpha$. Since $R = \alpha(R_1^+) \uplus \alpha(R_2^+) = \alpha(R_1^+ \uplus R_2^+) \in \|\langle R_1^+, \varphi_1\rangle\ \text{UNION ALL}^W \langle R_2^+, \varphi_2\rangle\|_\beta$.

(3) We show $\|\langle R_1^+, \varphi_1\rangle\ \text{CROSS JOIN}^W \langle R_2^+, \varphi_2\rangle\|_\beta = \{t_1\ \text{CROSS JOIN}\ t_2 \mid t_i \in \|\langle R_i^+, \varphi_i\rangle\|_\beta, i = 1, 2\}$. Let R^+ be an extended table obtained from S^+ by removing columns $\mathtt{ext}_1$ and $\mathtt{ext}_2$,

$$\begin{array}{rcl} \psi &=& \bigwedge_{r\in S^+} (\mathtt{ext}(r) \Leftrightarrow \mathtt{ext}_1(r) \wedge \mathtt{ext}_2(r))\,, \\ S^+ &=& \{\!\{r_1 \cup r_2 \cup \{\mathtt{ext} = x_{r_1,r_2}\} \mid r_1 \in R_1^+,\ r_2 \in R_2^+\}\!\}, \end{array}$$

assuming the column name $\mathtt{ext}$ in R_i is renamed as $\mathtt{ext}_i$ for $i = 1, 2$.

($\subseteq$): Suppose $R \in \|\langle R^+, \varphi_1 \wedge \varphi_2 \wedge \psi\rangle\|_\beta$. Then there exists α such that $R = \alpha(R^+)$, $\alpha \models \varphi_1 \wedge \varphi_2 \wedge \psi$, and $\beta \leq \alpha$.

We show that $\alpha(R^+) = (\alpha(R_1^+)\ \text{CROSS JOIN}\ \alpha(R_2^+))$. For $r \in R = \alpha(R^+)$, we have $r' \cup \{\mathtt{ext} = x\} \in R^+$, $\alpha(r') = r$, and $\alpha(x) = \text{true}$. Thus, $r'_1 \cup r'_2 \cup \{\mathtt{ext} = x, \mathtt{ext}_1 = x_1, \mathtt{ext}_2 = x_2\} \in S^+$, $r' = r'_1 \cup r'_2$, and $\alpha(x_i) = \text{true}$ from the construction of R^+ and ψ. This means that $\alpha(r'_i) \in \alpha(R_i^+)$ and $\alpha(r'_1) \cup \alpha(r'_2) = r$, hence r is the right hand side. By tracing reversely, we obtain $r \in \alpha(R^+)$ from $r \in (\alpha(R_1^+)\ \text{CROSS JOIN}\ \alpha(R_2^+))$.

Moreover $\alpha(R_i^+) \in \|\langle R_i^+, \varphi_i\rangle\|_\beta$ since $\alpha \models \varphi_i$ and $\beta \leq \alpha$. Therefore R is in the right hand side.

($\supseteq$): Suppose R is in the right hand side. Then there exists R_1 and R_2 such that $R = (R_1\ \text{CROSS JOIN}\ R_2)$ and $R_i \in \|\langle R_i^+, \varphi_i\rangle\|_\beta$. Thus, there exists α_i such that $R_i = \alpha_i(R_i^+)$, $\alpha_i \models \varphi_i$, and $\beta \leq \alpha_i$.

We fix an assignment δ from S^+ and $\alpha \models \psi$ as $\delta(\mathtt{ext(r)}) \Leftrightarrow \alpha_1(\mathtt{ext_1(r)}) \wedge \alpha_2(\mathtt{ext_2(r)})$ for $r \in S^+$. Here $\alpha = \alpha_1 \cup \alpha_2 \cup \delta$ is well defined, since common variables in domain of α_i are in W from the construction.

In similar to the ($\subseteq$) case, we can show that

$$\alpha(R^+) = (\alpha(R_1^+)\ \text{CROSS JOIN}\ \alpha(R_2^+)).$$

We obtain $R = \alpha(R^+) \in \|\langle R^+, \varphi_1 \wedge \varphi_2 \wedge \psi\rangle\|_\beta$ since $\alpha \models \varphi_1 \wedge \varphi_2 \wedge \psi$ and $\beta \leq \alpha$.

(4) We show $\|\text{SELECT}^W * \text{ FROM } \langle R^+, \varphi\rangle \text{ AS } tn \text{ WHERE } const\|_\beta$ is equal to $\{\text{SELECT } * \text{ FROM } t \text{ AS } tn \text{ WHERE } const \mid t \in \|\langle R_1^+, \varphi\rangle\|_\beta\}$. Let $\psi = \bigwedge_{r\in S^+}\{\mathtt{ext}(r) \Leftrightarrow \mathtt{ext}_1(r) \wedge \overline{const}^{\,r}\}$, $S^+ = \{\!\{r_1 \cup \{\mathtt{ext} = x_{r_1}\} \mid r_1 \in R_1^+\}\!\}$, and $\overline{const}^{\,r}$ be an expression $\theta(const)$ obtained from $const$ by applying the substitution $\theta = [tn.c \leftarrow c(r) \mid c \in \mathrm{Col}(r)]$.

($\subseteq$): Suppose $R \in \|\langle R^+, \varphi \wedge \psi\rangle\|_\beta$. Then there exists α such that $R = \alpha(R^+)$, $\alpha \models \varphi \wedge \psi$, and $\beta \leq \alpha$.

We show that $\alpha(R^+) = \text{SELECT} * \text{FROM } \alpha(R_1^+) \text{ WHERE } const$. For $r \in \alpha(R^+)$, we have $r' \cup \{\mathtt{ext} = x\} \in R^+$, $\alpha(r') = r$, and $\alpha(x) = \text{true}$. Thus, $r' \cup \{\mathtt{ext} = x, \mathtt{ext}_1 = x_1\} \in S^+$, $\alpha(x) = \alpha(x_1) = \text{true}$, and $\overline{const}^{\alpha(r')}$ from the construction of R+ and ψ. This means that $\alpha(r') \in \alpha(R_1^+)$, hence r is in the right hand side. By tracing reversely, we obtain $r \in \alpha(R^+)$ from $r \in$ SELECT $*$ FROM $\alpha(R_1^+)$ WHERE $const$.

Moreover $\alpha(R^+) \in \|\langle R_1^+, \varphi\rangle\|_\beta$ since $\alpha \models \varphi$ and $\beta \leq \alpha$. Therefore R is in the right hand side.

($\supseteq$): Suppose R is in the right hand side. Then there exists $R_1 \in \|\langle R_1^+, \varphi\rangle\|_\beta$ such that SELECT $*$ FROM R_1 WHERE $const$. Thus, there exists α_1 such that $R_1 = \alpha_1(R_1^+)$, $\alpha_1 \models \varphi$, and $\beta \leq \alpha$.

We fix an assignment δ from S^+ and $\alpha \models \psi$ as $\delta(\mathtt{ext}) \Leftrightarrow \alpha_1(\mathtt{ext}) \wedge \overline{const}^{\alpha_1(r)}$ for $r \in S^+$. From the construction, $\alpha = \alpha_1 \cup \delta$ is well defined. In similar to the ($\subseteq$) case, we can show that

$$\alpha(R^+) = \text{SELECT } * \text{ FROM } \alpha_1(R_1^+) \text{ WHERE } const.$$

We obtain $R = \alpha(R^+) \in \|\langle R^+, \varphi \wedge \psi\rangle\|_\beta$ since $\alpha \models \varphi \wedge \psi$ and $\beta \leq \alpha$.

(5) We show that $\|\text{SELECT}^W c_1, \ldots, c_n \text{ FROM } \langle R^+, \varphi\rangle\|_\beta$ is equal to $\{\text{SELECT } c_1, \ldots, c_n \text{ FROM } t \mid t \in \|\langle R^+, \varphi\rangle\|_\beta\}$. Let S^+ be

$$\{\!\{\langle c_1 = c_1(r), \ldots, c_n = c_n(r), \mathtt{ext} = \mathtt{ext}(r)\rangle \mid r \in R^+\}\!\}.$$

($\subseteq$): Suppose $R \in \|\langle S^+, \varphi\rangle\|_\beta$. Then there exists α such that $R = \alpha(S^+)$, $\alpha \models \varphi$, and $\beta \leq \alpha$. It is easy to show that $\alpha(S^+) =$ SELECT $c_1, \ldots, c_n$ FROM $\alpha(R^+)$. Moreover $\alpha(R^+) \in \|\langle R_1^+, \varphi\rangle\|_\beta$ since $\alpha \models \varphi$ and $\beta \leq \alpha$. Therefore R is in the right hand side.

($\supseteq$): Suppose R is in the right hand side. Then there exists $R_1 \in \|\langle R^+, \varphi\rangle\|_\beta$ such that SELECT $c_1, \ldots, c_n$ FROM R_1. Thus, there exists α such that $R_1 = \alpha(R^+)$, $\alpha \models \varphi$, and $\beta \leq \alpha$. Since $\alpha(S^+) =$ SELECT $c_1, \ldots, c_n$ FROM $\alpha(R^+)$, we obtain $R = \alpha(S^+) \in \|\langle R^+, \varphi\rangle\|_\beta$ since $\alpha \models \varphi$ and $\beta \leq \alpha$.

(6) We show $\|\text{SELECT}^W c, a(d) \text{ AS } dn \text{ FROM } \langle R^+, \varphi\rangle \text{ GROUP BY } c\|_\beta = \{\text{SELECT } c, a(d) \text{ AS } dn \text{ FROM } t \text{ GROUP BY } c \mid t \in \|\langle R^+, \varphi\rangle\|_\beta\}$. Let

$$\begin{array}{rcl} S^+ &=& \{\!\{\{c = v, dn = x_v, ext = y_v\} \mid v \in c(\langle R^+, \varphi\rangle)\}\!\}, \\ \psi &=& \displaystyle\bigwedge_{v\in c(\langle R^+,\varphi\rangle)} \left(y_v \Leftrightarrow \left(\bigvee_{r\in R^+} (\mathtt{ext}(r) \wedge d(r) = v)\right)\right) \\ && \wedge \displaystyle\bigwedge_{v\in c(\langle R^+,\varphi\rangle)} \left(x_v = exp_v\right) \end{array}$$

and exp_v be $\bar{a}\{\mathrm{if}(\mathtt{ext}(r) \wedge c(r) = v, d(r), \mathrm{id}_a) \mid r \in R^+\}$

($\subseteq$): Suppose $R \in \|\langle S^+, \varphi \wedge \psi\rangle\|_\beta$. Then there exists α such that $R = \alpha(S^+)$, $\alpha \models \varphi \wedge \psi$, and $\beta \leq \alpha$.

We show $\alpha(S^+) = \text{SELECT } c, a(d) \text{ AS } dn \text{ FROM } \alpha(R^+) \text{ GROUP BY } c$. For $\{c = v, dn = k\} \in R = \alpha(S^+)$, we have $\{c = v, dn = x_v, \mathtt{ext} = y_v\} \in S^+$, $\alpha(x_v) = k$, and $\alpha(y_v) = \text{true}$. From $\alpha \models \psi$, there exists $r \in R^+$ such that $\alpha(\mathtt{ext}(r)) = \text{true}$ and $\alpha(d(r)) = v$, and also $k = \bar{a}\{\mathrm{if}(\alpha(\mathtt{ext}(r)) \wedge \alpha(c(r)) = v, \alpha(d(r)), \mathrm{id}_a) \mid r \in R^+\}$. This means that $\alpha(R^+)$ has at least one record having v in the column c, and $k = \bar{a}\{\alpha(d(r)) \mid r \in R^+, \alpha(\mathtt{ext}(r)) = \text{true}, \alpha(c(r)) = v\}$, hence r is the right hand side. By tracing reversely, we obtain $r \in \alpha(S^+)$ from $r \in$ SELECT $c, a(d)$ AS dn FROM $\alpha(R^+)$ GROUP BY c.

Moreover $\alpha(R^+) \in \|\langle R^+, \varphi\rangle\|_\beta$ since $\alpha \models \varphi$ and $\beta \leq \alpha$. Therefore R is in the right hand side.

($\supseteq$): Suppose that R is in the right hand side. Then there exists $R_1 \in \|\langle R^+, \varphi\rangle\|_\beta$ such that

$$R = \text{SELECT } c, a(d) \text{ AS } dn \text{ FROM } R_1 \text{ GROUP BY } c.$$

Thus, there exists α_1 such that $R_1 = \alpha_1(R^+)$, $\alpha_1 \models \varphi$, and $\beta \leq \alpha_i$.

We fix an assignment δ from $\alpha \models \psi$ as

$$\delta(y_v) \Leftrightarrow \bigvee_{r\in R^+} (\alpha_1(\mathtt{ext(r)}) \wedge \alpha_1(d(r)) = v),$$

and $\delta(x_v) = \alpha(exp_v)$. From the construction, $\alpha = \alpha_1 \cup \delta$ is well defined.

In similar to the ($\subseteq$) case, we can show that

$$\alpha(S^+) = \text{SELECT } c, a(d) \text{ AS } dn \text{ FROM } \alpha(R^+) \text{ GROUP BY } c.$$

We obtain $R = \alpha(R^+) \in \|\langle R^+, \varphi \wedge \psi\rangle\|_\beta$ since $\alpha \models \varphi \wedge \psi$ and $\beta \leq \alpha$.

ACKNOWLEDGMENTS

The work is supported by JSPS KAKENHI Grant Number JP17H01721.

REFERENCES

[1] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. nu-Z: An Optimizing SMT Solver. *Proc. of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (TACAS 2015), LNCS 9035, pp.194–199, 2015.

[2] Marco Cadoli and Toni Mancini. Combining relational algebra, SQL, and constraint programming. *Proc. of the 4th International Symposium on Frontiers of Combining Systems* (FroCoS 2002), LNCS 2309, pp. 147–161, 2002.

[3] Marco Cadoli and Toni Mancini. Combining relational algebra, SQL, and constraint modelling, and local search. *Theory and Practice of Logic Programming*, Vol. 7, pp. 37–65, 2007.

[4] Toni Mancini, Pierre Flener, and Justin Pearson. Combinatorial problem solving over relational databases: view synthesis through constraint-based local search. *Pro. of ACM Symposium on Applied Computing*, (SAC 2012), pp. 80–87, 2012.

[5] C. J. Date and H. Drawen. A guide to the SQL standard (3rd ed.), Addison-Welsley, 1993.

[6] Pierre Flener, Justin Pearson, and Magnus Ågren. Introducing ESRA, a relational language for modelling combinatorial problems. *Proc. of the 13th International Symposium on Logic Based Program Synthesis and Transformation* (LOPSTR 03), LNCS 3018, pp. 214–232, 2004.

[7] Alan M. Frisch, Matthew Grum, Chris Jefferson, Bernadette Martínez Hernández, and Ian Miguel. The design of ESSENCE: A constraint language for specifying combinatorial problems. *Proc, of the 20th International Joint Conference on Artificial Intelligence* (IJCAI), pp. 80-–87, 2007.

[8] P. Van Hentenryck. The OPL Optimization Programming Language. *MIT Press*, Cambridge, Mass., 1999.

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

[10] Gerald Lach and Marco E. Lübbecke. Curriculum based course timetabling: new solutions toudine benchmark instances. *Annals of Operations Research*, Vol. 194, No. 1, pp. 255–272, 2010.

[11] Patrick Mills, Edward Tsang, Richard Williams, John Ford, and James Borrett. EaCL 1.5: An Easy abstract Constraint optimization Programming Language. *Technical Report CSM-324*, University of Essex, 1999.

[12] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a Standard CP Modelling Language. *Proc. of 13th International Conference on Principles and Practice of Constraint Programming* (CP), LNCS 4741, Springer, pp. 529–543, 2007.

[13] Naoyuki Tamura, Tomoya Tanjo, and Mutsunori Banbara. Domain-Specific Language Copris for Constraint Programming in Scala. Computer Software, Vol. 29, No. 4, pp. 4114–4129, 2012 (in Japanese).

[14] G. S. Tseitin. On the complexity of derivation in propositional calculus. *Leningrad Seminar on Mathematical Logic*, 1970. Found in *Automation of Reasoning 2: Classical Papers on Computational Logic*, Springer Verlag, Berlin, pp. 466–483, 1983.

[15] Edward Tsang, Patrick Mills, Richard Williams, John Ford, and James Borrett. A computer-aided constraint programming system. *Proc. of The First International Conference on The Practical Application of Constraint Technologies and Logic Programming* (PACLP), pp. 81–93, 1999.

[16] Yusaku Uchida, Masahiko Sakai, and Naoki Nishida. An extension of SQL for specifying combinatorial optimization problems. *Technical Report of IEICE on Software and Science*, Vol. 115, No. 508, pp. 25–30, 2016 (in Japanese).

[17] J. Zhou. Introduction to the constraint language NCL. *Journal of Logic Programming*, Vol. 45, pp. 71–103, 2000.

[18] Graph Coloring Instances, http://mat.gsia.cmu.edu/COLOR/instances.html.