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

Mashikko 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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

PPDP '18, September 3–5, 2018, Frankfurt am Main, Germany
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6441-6/18/09 . $15.00
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.

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
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 \((x_A \neq b \lor \neg y_A) \land (x_B \neq a \lor \neg y_B)\) 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.

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 state-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 \(\cup\) and \(\subseteq\) to represent the union and subset relation of multisets, respectively. For example, \([1, 1, 2, 3]\) \(\cup\) \([3, 4]\) = \([1, 1, 2, 3, 3, 4]\) and \([1, 1, 2, 3]\) \(\subseteq\) \([1, 1, 2, 3, 3, 4]\).

An assignment \(\alpha\) gives a value to each variable in its domain \(\text{Dom}(\alpha)\). 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 \(\alpha\) with \(\alpha(x_i) = v_i\) for \(x_i \in \text{Dom}(\alpha)\). If assignments \(\alpha\) and \(\beta\) satisfy that \(\alpha(x) = \beta(x)\) for common defined variables \(x \in \text{Dom}(\alpha) \cap \text{Dom}(\beta)\), we say that they are compatible. The union \(\alpha \cup \beta\) of compatible assignments \(\alpha\) and \(\beta\) is well defined as \((\alpha \cup \beta)(x) = \alpha(x)\) if \(x \in \text{Dom}(\alpha); (\alpha \cup \beta)(x) = \beta(x)\) otherwise.

If compatible assignments \(\alpha\) and \(\beta\) satisfies that \(\text{Dom}(\alpha) \supseteq \text{Dom}(\beta)\), we say that \(\alpha\) is an instance of \(\beta\), written as \(\beta \preceq \alpha\). We use \(\alpha|_W\) for \(\beta\) such that \(\beta \preceq \alpha\) and \(\text{Dom}(\beta) = W\).

We use a substitution for replacing an object in the sentences with some symbol or value. We write a substitution \(\theta\) as \([x_1 \leftarrow \ldots \leftarrow x_n \leftarrow v_1, \ldots, v_n]\).
Transformation of Extended SQL

\[ y_1, \ldots, x_n \leftarrow y_{\eta n}. \theta(T) \] 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 \((\{\text{product, STRING}\}, \{\text{size, INT}\})\). 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 \( \text{Col}(r) \) or \( \text{Col}(T) \). 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 \( \text{Img}(c_i) \) 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 \( \{\text{product} = A, \text{size} = 4\} \). We have \( \text{Col}(r_1) = \{\text{product}, \text{size}\} \), \( \text{size}(r_1) = 4 \), and \( \text{size}(PL) = \{1, 3, 4, 6\} \).

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 queries and integer queries are used similarly. For example, the following table query returns the table consisting of the records in \( PL \) in Table 1 (a) whose size is 3:

\[
\text{SELECT } * \text{ FROM } PL \text{ WHERE size } 3
\]

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 \( \leq \), \( <\), \( <\), 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 \( a \) 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, SUM is a function \( \Sigma \) 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 \( PL \).

\[
\text{SELECT SUM(size) FROM PL}
\]

**Table operators** are defined as follows:

1. \( R_1 \ \text{UNION ALL} \ R_2 \), which returns \( R_1 \cup R_2 \), where tables \( R_1 \) and \( R_2 \) must have the same number of columns without type mismatching.

2. \( R_1 \ \text{CROSS JOIN} \ R_2 \), which returns \( \{r_1 \cup r_2 \mid r_1 \in R_1, r_2 \in R_2\} \).

3. \( \text{SELECT } * \text{ FROM } R \text{ AS } tn \text{ WHERE } \text{const}, \) which returns \( \{r \mid r \in R, \text{const} \} \), where \( \text{const} \) is the value of an expression \( \text{const} \), which is obtained from \( \text{const} \) by applying the substitution \( \theta \): \( \theta = [m.n \leftarrow c(r) \mid c \in \text{Col}(r)] \).

4. \( \text{SELECT } c_1, \ldots, c_n \text{ FROM } R, \) which returns \( \{(c_1 = c_1(r), \ldots, c_n = c_n(r)) \mid r \in R\} \).

5. \( \text{SELECT } c, a(d) \text{ AS } dn \text{ FROM } R \text{ GROUP BY } c, \) which returns \( \{[c = v, dn = w] \mid v \in c(R), w = a(d(r)) \mid r \in R, c(r) = v\} \)

Here \( c,d \) is column names and \( a \) is an aggregate operator, \( c \) and \( a \) 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 \( \text{const} \) is decomposed to

   \[
   \text{SELECT columns FROM } (\text{SELECT } * \text{ FROM } R \text{ WHERE } \text{const}).
   \]

2. \( R_1 \ \text{INNER JOIN} \ R_2 \) ON \( R_1.c_1 = R_2.c_2 \) is decomposed to

   \[
   \text{SELECT } * \text{ FROM } (R_1 \ \text{CROSS JOIN} \ R_2) \text{ WHERE } R_1.c_1 = R_2.c_2.
   \]

3. \( \text{SELECT } c, a(d) \text{ FROM } R \text{ WHERE } \text{const}_1 \text{ GROUP BY } c \text{ HAVING } \text{const}_2 \) is decomposed to

   \[
   \text{SELECT } c, a(d) \text{ FROM } (\text{SELECT } c, a(d) \text{ FROM } (\text{SELECT } * \text{ FROM } R \text{ WHERE } \text{const}_1) \text{ GROUP BY } c) \text{ WHERE } \text{const}_2.
   \]

4. (SELECT DISTINCT \( c \) FROM \( R \) is decomposed to

   \[
   \text{SELECT } c \text{ FROM } R \text{ GROUP BY } c
   \]

5. (SELECT \( a(\text{DISTINCT } c_2) \) FROM \( R \) GROUP BY \( c_1 \) is decomposed to

   \[
   \text{SELECT } a(c_2) \text{ FROM } (\text{SELECT } c_1, c_2 \text{ FROM } R \text{ GROUP BY } c_1, c_2) \text{ GROUP BY } c_1
   \]
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 \((X, P, f)\), where \(X\) is a set of solution candidates, called a search space, \(P\) is a predicate on \(X\), and \(f : X \rightarrow \mathbb{Z}\) 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:

```
CREATE SET sname AS
SELECT R
FROM R
WHERE each cj is either a column name of the table R or a term in the form of \(\text{CHOICE}(S, S_i)\). Here \(S_i\) is a single column table. For a simple presentation, we focus on the form
```

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

\[
\{ [c = c(r), s = f(r)] \mid r \in R \} : f : R \rightarrow \text{Img}(S)
\]

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

```
SUBTABLE OF choose_query,
```

it gives the union of the power sets of \(R \in \text{SetTab}\)

\[
\{ R' \mid R' \subseteq R, R \in \text{SetTab} \},
\]

where \(\text{SetTab}\) 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.

```
CREATE SET sname
HAS \(Q_1(t_1, \ldots, t_n) \ldots Q_m(t_1, \ldots, t_n)\)
FOR \(t_1 \in \text{choose_query}_1, \ldots, t_n \in \text{choose_query}_n\)
```

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

\[
name := \{ (s_1, \ldots, s_m) \mid s_j = Q_i(t_1, \ldots, t_n), t_i \in \text{SetTab}_i, i = 1, \ldots, n, j = 1, \ldots, m\},
\]

where each \(\text{SetTab}_i\) is the set of tables induced by \(\text{choose_query}_i\). In case of \(n = m = 1\), it is in the form of

```
CREATE SET sname HAS \(Q(t)\) FOR \(t \in \text{choose_query}\).
```

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

```
CREATE SET sname AS 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.

```
1 CREATE SET SearchSpace AS
2 SUBTABLE OF (SELECT \text{product}, \text{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:

```
CREATE SET \(sname_1\) HAS \((t_1, \ldots, t_n) \in sname_2\) SUCH THAT \(\text{BoolQuery}(t_1, \ldots, t_n)\)
```

where \(\text{BoolQuery}(t_1, \ldots, t_n)\) 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

\[
\text{filter}(sname_1) := \{ (t_1, \ldots, t_n) \in sname_2 \mid \text{BoolQuery}(t_1, \ldots, t_n)\}.
\]

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 \(\text{GROUP BY}\) indicates that a record is possibly created for each basket, and \(\text{HAVING}\) 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.

```
CREATE TABLE \(tname_1, \ldots, tname_n\) AS \((t_1, \ldots, t_n) \in \text{name}\)
```

```
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 (t INNER JOIN Ban ON t.product = Ban.product AND t.basket = Ban.basket)
9 )
10 AND NOT EXISTS (SELECT * FROM t
11 GROUP BY basket
12 HAVING
13 SUM(lookup(t.product, PL.product, size)) > lookup(t.basket, BL.basket, capacity)
14 )
```

Figure 2: Filtering constraint for solutions of Example 1.1

\(^1\) CHOOSE-structure was proposed by Mancini et al. [4] to denote an arbitrary element in the given single-columned table.
where an integer query $Goal(t)$ is optional, and written as either

\[
\text{MAXIMIZING } \text{IntegerQuery}(t_1, \ldots, t_n), \text{ or}
\]

\[
\text{MINIMIZING } \text{IntegerQuery}(t_1, \ldots, t_n).
\]

The trivial meaning is

\[
(t_{name_1}, \ldots, t_{name_n}) := (t_1, \ldots, t_n) \in \text{name}
\]

such that

\[
\text{IntegerQuery}(t_1, \ldots, t_n) = \max \{ \text{IntegerQuery}(\bar{u}) \mid \bar{u} \in \text{name} \}.
\]

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 $\text{CHOOSE}$ and $\text{SUBTABLE}$ of fragments. In other words, we didn’t extend table handling operators such as $\text{JOIN}$, $\text{SELECT}$, $\text{GROUP BY}$, 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 $\text{CREATE TABLE}$ 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 $\text{CL}$ consists of records each of which specifies a student information including his/her affiliated laboratory name.

```
CREATE SET SearchSp AS
SELECT v,CHOOSE(CL) FROM V
WHERE lookup v from association (key,val) in asc
MACRO lookup(v,asc,key,val) AS
SELECT val FROM association WHERE v = key
CREATE SET Solutions HAS t IN SearchSp
SUCH THAT NOT EXISTS(
SELECT + FROM E
WHERE lookup(E.v1,t,v,c)=lookup(E.v2,t,v,c)
)
CREATE TABLE solution AS t IN Solutions
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 $\text{Students}$ consists of records each of which specifies a student information including his/her affiliated laboratory name.
- Lines 6–11: the table $\text{SubSessions}$ consists of records each of which specifies a sub-session information with the allowed number of presentations.
- Lines 12–15: the table $\text{Bans}$ consists of records specifying combinations of a laboratory and a session to be avoided.
- Lines 16–19: the table $\text{Labs}$ specifies a laboratory with the affiliated group.

```
CREATE TABLE Students (student_id TEXT, name TEXT, lab_name TEXT)
CREATE TABLE SubSessions (subsess_id INT KEY, room TEXT, session INT, presen_max INT)
CREATE TABLE Bans (lab_name TEXT, session INT)
CREATE TABLE Labs (lab_name TEXT KEY, lab_gr TEXT)
```

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

```
CREATE VIEW SubSessionIds AS
SELECT subsess_id FROM SubSessions;
CREATE VIEW StudentsWithGroup AS
SELECT * FROM Students CROSS JOIN Labs
WHERE Labs.lab_name = Students.lab_name;
CREATE SET SearchSpace AS
SELECT * FROM t CROSS JOIN SubSessions
WHERE subsess_id = t.session_id
FOR t IN SearchSpace;
/\ for H1 /\ 
CREATE SET ReducedH2Space HAS t IN ReducedH2Space
SUCH THAT NOT EXISTS(
SELECT * FROM t WHERE EXISTS ( 
SELECT * FROM Bans 
WHERE Bans.lab_name = t.lab_name 
AND Bans.session = t.session 
))
/\ for H2 /\ 
CREATE SET ReducedH3Space HAS t IN ReducedH3Space
SUCH THAT NOT EXISTS(
SELECT * FROM t AS t1 CROSS JOIN t AS t2
WHERE t1.lab_name = t2.lab_name 
AND t1.room <> t2.room 
AND t1.session = t2.session 
)
/\ for H3 /\ 
CREATE TABLE solution AS t IN ReducedH4Space
MAXIMIZING
/\ for S1 /\ 
(SELECT SUM(kind) AS s 
FROM (SELECT COUNT(DISTINCT lab_name) AS kind 
FROM t GROUP BY room, session))
/\ for S2 /\ 
- (SELECT SUM(kind) AS s 
FROM (SELECT COUNT(DISTINCT lab_gr) AS kind 
FROM t GROUP BY room, session))
/\ for S3 /\ 
- (SELECT SUM(kind) AS s 
FROM (SELECT COUNT(DISTINCT room) AS kind 
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 of a single table containing variables over finite domains and a constraint on them. Suppose that a constrained table \( \langle R, \sigma \rangle \) represents the search space, and that \( \text{BoolQuery}(t) \) is located after \( \text{SUCH THAT} \) in a filtering-step description. By the extended SQL operations, \( \text{BoolQuery}(\langle R, \sigma \rangle) \) results in a constrained table \( \langle y, \sigma' \rangle \), where the first item \( y \) is a Boolean variable because the resulted constrained table represents a set of Boolean values. Once a satisfiable solution \( \sigma \) for \( \psi \land \varphi \land \varphi' \) is found by a CP/SMT solver, the final solution is obtained from the first item \( R \) of the solution space by applying \( \sigma \).

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 \( \text{ext} \) with Boolean type. An constrained table \( R \) is a pair \( \langle R^*, \varphi \rangle \) of an extended table \( R^* \) and a constraint \( \varphi \), where constraints are on some fixed model.

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

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

The set \( \| R \|_\alpha \) represented by a constrained table \( R \) is defined as follows, where \( \emptyset \) is an empty assignment having empty domain.
We now extend SQL operations on constrained values, which are values that can be assumed that common variables among such constrained values, or integer values. We abbreviate the assignment in with respect to an assignment \( \beta \) is given by
\[
\| (R^*, \varphi) \|_\beta = (\alpha(R^*) \mid \alpha \models \varphi, \beta \preceq \alpha, \text{Var}(R^*) \subseteq \text{Dom}(\alpha)).
\]
The set represented by a constrained table in this paper is always finite. Thus the final domain property for every variable is (explicitly or implicitly) described in \( \varphi \), or somewhere outside as a context. For the latter case, \( \beta \) 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 \((x, \varphi)\) of a variable and a constraint, which represents the following set of values:
\[
\| (x, \varphi) \|_\beta = \{ \alpha(x) \mid \alpha \models \varphi, \beta \preceq \alpha, x \in \text{Dom}(\alpha) \}
\]
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 \( R \) to work correctly for any fixed assignment \( \beta \) with \( \text{Dom}(\beta) = W \):
\[
\| Q^W (R_1, \ldots, R_n) \|_\beta = \{ Q(t_1, \ldots, t_n) \mid t_i \in \| R_i \|_\beta, i = 1, \ldots, n \}.
\]

Note that this complex treatment of \( \beta \) is necessary in order to synchronize assignments in \( R_1, \ldots, R_n \), because of a search space is given in the form of \( \langle R_1^1, \ldots, R_n^r \rangle, \varphi \rangle \) for \( R_i = (R_i^1, \varphi_i) \). (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. \( E^W \) is \( \langle x, x \models E \rangle \) for a Boolean constant \( E \).
2. \( \text{NOT}^W (y, \varphi) = \langle x, \varphi \land (x \models \neg y) \rangle \).
3. \( (y_1, \varphi_1) \text{ bop } (y_2, \varphi_2) = \langle x, \varphi_1 \land \varphi_2 \land \psi \rangle \), where \( \psi \) is \( x \models y_1 \text{ bop } y_2 \) for a Boolean operator \( \text{ bop } \).
4. \( (k_1, \varphi_1) \text{ cop } (k_2, \varphi_2) = \langle x, \varphi_1 \land \varphi_2 \land \psi \rangle \), where \( \psi \) is \( x \models k_1 \text{ cop } k_2 \) for a comparison operator \( \text{ cop } \).
5. \( \text{EXISTS}^W (R^*, \varphi) = \langle x, \varphi \land \psi \rangle \), where \( \psi \) is \( x \models \lor_{r \in R^*} \text{ext}(r) \).

Integer operations.

1. \( E^W = \langle x, x \models E \rangle \), for an integer constant \( E \).
2. \( (y_1, \varphi_1) \text{ iop } (y_2, \varphi_2) = \langle x, \varphi_1 \land \varphi_2 \land \psi \rangle \), where \( \psi \) is \( x = y_1 \text{ iop } y_2 \) for an integer operator \( \text{ iop } \).
3. \( \text{SELECT}^W \alpha(c) \) from \( (R^*, \varphi) = \langle x, \varphi \land \psi \rangle \), where \( \psi \) is a constraint \( x = \bar{a} \text{ if } \text{ext}(r) \land c(r), \text{id}_a \rangle \land \{ r \in R^* \} \), and if \( (e_1, e_2, e_3) \) denotes \( (e_1 \models e_2) \land (\neg e_1 \models e_3) \), \text{id}_a \) is the identity element of function \( \bar{a} \).

Note that the average function has no identity element, but can be rephrased as \( \text{id}_A = \Sigma(A)/|A| \).

Table queries.

1. \( R^W = \langle R^*, \text{true} \rangle \) for an ordinary table \( R \), where \( R^* \) is obtained from \( R \) by adding a column name \( \text{ext} \) in which all values are true.
2. \( (R_1^1, \varphi_1) \cup \text{ ALL}^W (R_2^1, \varphi_2) = (R_1^1 \cup R_2^1, \varphi_1 \land \varphi_2) \).
3. \( (R_1^1, \varphi_1) \text{ CROSS JOIN}^W (R_2^1, \varphi_2) = (R_1^1, \varphi_1 \land \varphi_2 \land \psi) \), where \( R^* \) is a extended table obtained from \( R^* \) by removing columns \( \text{ext}_1 \) and \( \text{ext}_2 \).
4. \( \text{SELECT}^W \phi \text{ FROM } (R^*, \varphi) \text{ AS } tn \text{ WHERE } \text{ const} = (R^*, \varphi) \land \psi \), where \( R^* \) is obtained from \( S^* \) by removing column \( \text{ext}_1 \).
5. \( \text{SELECT}^W \phi_1, \ldots, \phi_n \text{ FROM } (R^*, \varphi) = (S^*, \varphi) \), where \( S^* \) is

\[
\| (\phi_1 = \phi_2(\cdots) \mid r \in R^*) \|
\]

(6) \( \text{SELECT}^W \phi \text{ FROM } (R^*, \varphi) \text{ AS } tn \text{ GROUP BY } \phi = (S^*, \varphi) \text{ GROUP BY } \phi \), where \( S^* \) and \( \psi \) are defined as follows. Suppose \( c((R^*, \varphi)) \) represents a finite set that covers the image of \( (R^*, \varphi) \), that is, all possible values for the column \( c \) determined from \( (R^*, \varphi) \).

\[
S^* = \{ (c = v, d = x_v, \text{ext} = y_v) \mid v \in c((R^*, \varphi)) \}
\]

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

\[
\psi = \bigwedge_{v \in c((R^*, \varphi))} \left( y_v \models \left( \bigvee_{r \in R^*} (\text{ext}(r) \land d(r) = v) \right) \land \bigwedge_{v \in c((R^*, \varphi))} (x_v = \text{exp}_v) \right)
\]

and \( \text{exp}_v \) is
\[
\bar{a}(\text{ext}(r) \land c(r) = v, d(r), \text{id}_a) \mid r \in R^*)
\]

Remark that subqueries may appear in \( \text{WHERE} \) constraints, whose treatment is not described in the above definition for simplicity. Subqueries can be processed recursively by attaching the following
We show how CP/SMT constraints are produced by using the example and show that $T \parallel \langle \cdot \rangle$. For an assignment $\alpha$ such that for $r \in R$

- $\alpha(x') = f(r)$, and
- $\alpha(y') = true$ if and only if $c = c(r), s = f(r)$ is in $T$.

Then, $\alpha \models \varphi$, and hence $T \in \parallel (R^+, \varphi)\parallel$.

A simpler case having no SUBTABLE $OF$ is shown similarly. $\square$

Let $Q_i$ be an ordinary SQL query that returns a table, and $(S^+_i, \varphi_i)$ 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**

**NAME**

HAS $(Q_i(t_1, \ldots, t_n), \varphi_i)$

$t_i$ IN FOR $(S^+_i, \varphi_i), \ldots, t_n IN (S^+_n, \varphi_n)$.

This statement binds $\text{name}$ to the constrained table $(R^+_1, \ldots, R^+_n, \varphi)$ generated in the following way. Assume that $W$ is the set of variables used in $(S^+_i, \varphi_i), \ldots, (S^+_n, \varphi_n)$. Let $(R^+_i, \varphi_j)$ be the constraint table $Q^W_i ((S^+_i, true), \ldots, (S^+_n, true)) (j = 1, \ldots, m)$, where we can assume the sets of variables in $(R^+_j, \varphi_j)$ are pairwise disjoint except for those in $W$. Then, this create-set statement is converted to $(\langle R^+_1, \ldots, R^+_m, \varphi \rangle)$, where $\varphi$ is

$$\bigwedge_{j=1,\ldots,m} \varphi_j \wedge \bigwedge_{i=1,\ldots,n} \varphi_i.$$
where \((x, \phi)\) indicates \(Q^W((R_0^+, \text{true}), \ldots, (R_n^+, \text{true}))\) for the set \(W\) of variables used in \((R_1^+, \ldots, R_n^+), \psi\).

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

**Proof.** Supposing a simple statement

\[
\text{CREATE SET } \text{sn} \text{ AS } \{t \mid t \in \{(R^+, \psi)\}, Q(t)\}, \text{where} \langle x, \phi \rangle = Q^W((R^+, \text{true})), \text{for the set } W \text{ of variables used in } (R^+, \psi).
\]

(3): Let a table \(R\) be in the left hand side. Then there exists an assignment \(\alpha\) such that \(\alpha \models \psi \land \phi \land x\) and \(R = \alpha R^\phi\). Then \(\beta = \alpha|_W\) and also \(R \in \{(R^+, \psi)\}\). On the other hand, from \(\beta \leq \alpha, \alpha \models \neg \phi\), and also true \(\in \{\langle x, \phi \rangle\}_\beta = \{Q^W((R^+, \text{true})))_\beta\}\) for \(\alpha\). We obtain \((\{R^+, \psi\})\). Therefore, the table \(R\) is in the right hand side.

(2): Let a table \(R\) be in the right hand side. Then, there exists a table \(R \in \{(R^+, \psi)\}\) such that \(Q(R)\). From the former, there exists an assignment \(\beta\) satisfying \(\beta \models \psi\) and \(R = \beta R^\phi\), where \(\text{Dom}(\beta) = W\), and hence \(R \in \{(R^+, \psi)\}\). Then \(\beta \models \psi\) and \(\beta = \alpha|_W\) for some assignment \(\alpha\). This follows directly, and there exists \(\alpha\) such that \(k = \alpha(x), \alpha \models \psi \land \phi\) and \(R = \alpha R^\phi\). Therefore \(b\) holds.

Assume \((a)\), then \(R_1 \in \{(R^+, \psi)\}\) for each \(i\). We can obtain \(\{\langle x, \phi \rangle\}_\beta = \{Q^W((R^+, \text{true})))_\beta\}\) from the former. Then we can show that \(k\) is in the left hand side of \((a)\), if, then \(k = \alpha(x), \text{therefore} \text{the filtering step produces}\) \((\{R^+, \psi\})\).

\[
\text{SELECT}^W c \text{ FROM } (R^+, \psi) \quad \text{WHERE } E. v_1 = v.
\]

Therefore, the subquery returns the color \(E. v_1\) of the vertex \(E. v_1\). Since all value of the column \(\text{ext}\) is true and column \(v\) has concrete values \(a, \ldots, d\), the subquery can be replaced with

\[
\text{SELECT} c \text{ FROM } R^+ \quad \text{WHERE } E. v_1 = v.
\]

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 \((S^+, \phi_S^+))\) illustrated in Table 6(b). Then, by the NOT EXISTS constraint in line 7, the following constrained table \((\langle z, \varphi_2 \rangle)\) is produced, where \(\varphi_2\) is \((z \leftrightarrow \neg w) \\
\lor (w \lor y_1 \lor y_2) \lor y_3 \lor y_4\) therefore the filtering step produces a constraint table \((R^+, \varphi_F)\), where \(\varphi_F = \varphi_R \land \varphi_S \land \varphi_2 \land z\).
In the selection step, the goal function in line 12 produces a constrained table \( \langle k, \varphi_k \rangle \), where \( \varphi_k \) is \( k = \max(x_a, x_b, x_c, x_d) \). Finally, it produces a CP/SMT instance: \( \varphi_F \land \varphi_k \) 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 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,

\[
\text{SELECT}^W \text{columns FROM } (R^+ , \varphi)
\]

can always be represented as

\[
(\text{SELECT columns FROM } R^+ , \varphi)
\]

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

We prepared an operation \text{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 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 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.

## 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 \text{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 \( a \),

\[
\text{SELECT } a(c) \text{ FROM } a(R^+)
\]

\[
= \bar{a}(\text{if}(a(\text{ext}(r)), a(c(r)), \text{id}_a) \mid r \in R^+)
\]
We indicate this for the simple case that
\[ \| (c_1 = \alpha(d_1), \ldots, c_n = \alpha(d_n)) \|_{R^+}, \alpha(y) = y \| \]
Hence, “SELECT \( a(c) \) FROM \( R^{+} \)” is
\[ \tilde{a}(\alpha(c(r))) \mid r \in R^{+}, \alpha(\text{ext}(r)) = \text{true} \]
Since \( id_a \) is identity element of \( a \), this is equal to
\[ \tilde{a}(\text{if}(\alpha(\text{ext}(r)), \alpha(c(r)), \text{id}_a)) \mid r \in R^{+} \]
We show that a general form of Theorem 5.3, that is
\[ \| Q^{W}((R_{1}^{+}, \varphi_{1}), \ldots, (R_{n}^{+}, \varphi_{n})) \|_{\beta} = (Q(d_{1}, \ldots, d_{n}) \mid t_{i} \in \| R_{i}^{+} \|_{\beta}, i = 1, \ldots, n). \]
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) \( \| E^{W} \|_{\beta} = \{ \alpha(x) \mid \alpha \models (x \in E), \beta \leq \alpha \} = \{ E \}. \)

(2) We show \( \| \neg W((y, \varphi)) \|_{\beta} = \{ \text{not } \neg \circ v \mid v \in \| (y, \varphi) \|_{\beta} \}. \)

(3) For \( b \in \{ (x, \varphi \land (x \lor \neg y)) \|_{\beta} \}, \) there exists an assignment \( \alpha \) such that \( \alpha(x) = b, \alpha \models (\varphi \land (x \lor \neg y)), \) and \( \beta \leq \alpha. \) Since \( \alpha(y) = \neg \varphi \), it holds that \( b \models \neg \alpha(y). \) Therefore \( b \) is also in the right hand side.

(4) For \( b \in \beta \) in the right hand side, there exists an assignment \( \alpha \) such that \( \alpha(y) = \neg b, \alpha \models \varphi, \) and \( \beta \leq \alpha. \) Letting \( \alpha' = \alpha \cup \{ x \leftarrow b \}, \) we obtain \( \alpha' \models (\varphi \land (x \lor \neg y)) \) and \( \beta \leq \alpha' \leq \beta. \) Therefore \( b \) is also in \( \| (x, \varphi \land (x \lor \neg y)) \|_{\beta} \).

(5) We show \( \| (y, \varphi_{1}) \circ_{W} (y, \varphi_{2}) \|_{\beta} = \{ k_{1} \circ_{op} k_{2} \mid k_{1} \in \| (y, \varphi_{1}) \|_{\beta}, k_{2} \in \| (y, \varphi_{2}) \|_{\beta}, i = 1, 2 \}. \) Let \( \psi \models x \in y_{1} \circ_{op} y_{2}. \)

(6) For \( b \in \beta \) in the right hand side, there exists a domain \( \alpha_{i} (i = 1, 2) \) such that \( b \models \alpha_{i}(y_{1}) \circ_{op} \alpha_{i}(y_{2}), \alpha_{i} \models \varphi_{i}, \) and \( \beta \leq \alpha_{i}. \) From the construction, common variables in domains \( \text{Dom} \) \( \alpha_{i} \) \( i = 1, 2 \) in \( W, \) hence \( \alpha_{i} = \alpha_{1} \cup \alpha_{2} \cup \{ x \leftarrow b \} \) is well defined.

(7) We show \( \| \exists \{ (y, \varphi_{1}) \circ_{W} (y, \varphi_{2}) \} \|_{\beta} = \{ k_{1} \circ_{op} k_{2} \mid k_{1} \in \| (y, \varphi_{1}) \|_{\beta}, i = 1, 2 \}. \) Let \( \psi \models x \in y_{1} \circ_{op} y_{2}. \)

(8) For \( b \in \beta \) in the right hand side, there exists a domain \( \alpha_{i} (i = 1, 2) \) such that \( b \models \alpha_{i}(y_{1}) \circ_{op} \alpha_{i}(y_{2}), \alpha_{i} \models \varphi_{i}, \) and \( \beta \leq \alpha_{i}. \)

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

(2) We show \( \| (R_{1}^{+}, \varphi_{1}) \cup \cup_{W} (R_{2}^{+}, \varphi_{2}) \|_{\beta} = \{ t_{1} \cup \cup_{op} t_{2} \mid t_{1} \in \| (R_{1}^{+}, \varphi_{1}) \|_{\beta} \). \)

(3) For a table \( R \in \| (R_{1}^{+} \cup R_{2}^{+}, \varphi_{1} \cup \varphi_{2}) \|_{\beta} \), there exists an assignment \( \alpha \) such that \( R = \alpha((R_{1}^{+} \cup R_{2}^{+}), \alpha \models \varphi_{1} \cup \varphi_{2}, \) and \( \beta \leq \alpha. \)

(4) For a table \( R \in \| (R_{1}^{+}, \varphi_{1}) \|_{\beta} \) such that \( R = R_{1} \cup R_{2} \), hence \( R_{1} \in \| (R_{1}^{+}, \varphi_{1}) \|_{\beta} \). Therefore the table \( R \) is also in the right hand side.

(5) We show \( \| (R_{1}^{+}, \varphi_{1}) \|_{\beta} \) such that \( R_{1} \in \| (R_{1}^{+}, \varphi_{1}) \|_{\beta} \) (i = 1, 2).
exists $a_1$ such that $R_1 = \alpha(R_1^*)$, $a_1 \models \varphi_1$, and $\beta \leq a_1$. From the construction, common variables in domains $\text{Dom}(a_i)$ are in $W$, hence $\alpha = a_1 \cup a_2$ is well defined. Thus, we obtain $\alpha \models \varphi_1 \land \varphi_2$, and $\beta \leq \alpha$. Since $R = \alpha(R_1^+) \cup \alpha(R_2^+) = \alpha(R_1^+ \cup R_2^+) \in \{(R_1^+ \cup R_2^+), \psi_1\} \cup \text{UNION ALL} \{(R_1^+, \psi_1)\} \beta$.

(3) We show $(\alpha(R_1^+), \psi_1) \psi \alpha (R_2^+, \psi_2)\psi \beta = \{(t_1 \text{ CROSS JOIN} t_2) | t_1 \in (R_1^+, \psi_1), t_2 \in (R_2^+, \psi_2), i = 1, 2\}$. Let $R^*$ be an extended table obtained from $S^*$ by removing columns $ext_1$ and $ext_2$, 

$$\psi = \bigwedge_{r \in S^*} (\text{ext}(r) \Rightarrow \text{ext}_1(r) \land \text{ext}_2(r)),$$

$$S^* = \{(r_1 \cup r_2 \cup \{\text{ext} = x_{n_1}, x_{n_2}\} | r_1 \in R_1^+, r_2 \in R_2^+)\}.$$ 

assuming the column name $\text{ext}$ in $R_1$ is renamed as $\text{ext}_1$ for $i = 1, 2$.

(3): Suppose $R \in \{(R_1^+, \varphi_1 \land \varphi_2)\} \beta$ since $\alpha \models \varphi_1$ and $\beta \leq \alpha$. Therefore $R$ is in the right hand side.

(2): 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 $R_1 \in \{(R_1^+, \psi_1)\} \beta$. Thus, there exists $a_1$ such that $R_1 = \alpha(R_1^*)$, $a_1 \models \varphi_1$, and $\beta \leq a_1$. We fix an assignment $\delta$ from $S^*$ and $\alpha \models \psi$ by $\delta(\text{ext}) \Rightarrow (\alpha(\text{ext}(r)) \Leftrightarrow \alpha_i(\text{ext}(r)) \land \alpha_j(\text{ext}(r))$ for $r \in S^*$. From the construction, $\alpha = a_1 \cup \delta$ is well defined. In similar to the (3) case, we can show that

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

We obtain $R \in \{(R_1^+, \varphi \land \varphi)\} \beta$ since $\alpha \models \varphi \land \varphi$ and $\beta \leq \alpha$. (5) We show that $\{(\text{SELECT} c_1, \ldots, c_n \text{ FROM} (R_1^+, \varphi)\} \beta$ is equal to $\{(c_1, \ldots, c_n) \in \{(R_1^+, \varphi)\} \beta\}$ for $\alpha \models \varphi$ and $\beta \leq \alpha$. Therefore $R$ is in the right hand side.

(2): Suppose $R$ is in the right hand side. Then there exists $R_1 \in \{(R_1^+, \varphi)\} \beta$ such that $\alpha \models \varphi$ and $\beta \leq \alpha$. We fix an assignment $\delta$ from $\alpha \models \psi$ as $\delta(\text{ext}) \Rightarrow \delta(\text{ext}(r) \land \alpha_i(\varphi) \land \alpha_j(\varphi) )$ for $r \in S^*$. From the construction, $\alpha = a_1 \cup \delta$ is well defined. In similar to the (3) case, we can show that

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

We obtain $R \in \{(R_1^+, \varphi \land \varphi)\} \beta$ since $\alpha \models \varphi \land \varphi$ and $\beta \leq \alpha$. (4) We show $[\text{SELECT} W \ast \text{ FROM} (R_1^+, \varphi) \text{ WHERE const}] \beta$ is equal to $\{(R_1^+, \varphi)\} \beta$ for $\alpha \models \varphi$ and $\beta \leq \alpha$. We fix an assignment $\delta$ from $\alpha \models \psi$ as $\delta(\text{ext}) \Rightarrow \delta(\text{ext}(r) \land \alpha_i(\varphi) \land \alpha_j(\varphi) )$ for $r \in S^*$. From the construction, $\alpha = a_1 \cup \delta$ is well defined. In similar to the (3) case, we can show that

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

We obtain $R \in \{(R_1^+, \varphi \land \varphi)\} \beta$ since $\alpha \models \varphi \land \varphi$ and $\beta \leq \alpha$. (2): Suppose $R$ is in the right hand side. Then there exists $R_1 \in \{(R_1^+, \varphi)\} \beta$ such that $\alpha \models \varphi$ and $\beta \leq \alpha$. We fix an assignment $\delta$ from $\alpha \models \psi$ as $\delta(\text{ext}) \Rightarrow \delta(\text{ext}(r) \land \alpha_i(\varphi) \land \alpha_j(\varphi) )$ for $r \in S^*$. From the construction, $\alpha = a_1 \cup \delta$ is well defined. In similar to the (3) case, we can show that

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

We obtain $R \in \{(R_1^+, \varphi \land \varphi)\} \beta$ since $\alpha \models \varphi \land \varphi$ and $\beta \leq \alpha$. (3): Suppose $R \in \{(S^*, \varphi)\} \beta$. Then there exists $\alpha$ such that $R = \alpha(S^*)$, $\alpha \models \varphi$, and $\beta \leq \alpha$. It is easy to show that $\alpha(S^*) = \text{SELECT} c_1, \ldots, c_n \text{ FROM} \alpha(R^*)$. Moreover $\alpha(R^*) \in \{(R_1^+, \varphi)\} \beta$ since $\alpha \models \varphi$ and $\beta \leq \alpha$. Therefore $R$ is in the right hand side.

(2): Suppose $R$ is in the right hand side. Then there exists $R_1 \in \{(R_1^+, \varphi)\} \beta$ such that $\alpha \models \varphi$ and $\beta \leq \alpha$. We fix an assignment $\delta$ from $\alpha \models \psi$ as $\delta(\text{ext}) \Rightarrow \delta(\text{ext}(r) \land \alpha_i(\varphi) \land \alpha_j(\varphi) )$ for $r \in S^*$. From the construction, $\alpha = a_1 \cup \delta$ is well defined. In similar to the (3) case, we can show that

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

We obtain $R \in \{(R_1^+, \varphi \land \varphi)\} \beta$ since $\alpha \models \varphi \land \varphi$ and $\beta \leq \alpha$. (2): Suppose $R$ is in the right hand side. Then there exists $R_1 \in \{(R_1^+, \varphi)\} \beta$ such that $\alpha \models \varphi$ and $\beta \leq \alpha$. We fix an assignment $\delta$ from $\alpha \models \psi$ as $\delta(\text{ext}) \Rightarrow \delta(\text{ext}(r) \land \alpha_i(\varphi) \land \alpha_j(\varphi) )$ for $r \in S^*$. From the construction, $\alpha = a_1 \cup \delta$ is well defined. In similar to the (3) case, we can show that

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

We obtain $R \in \{(R_1^+, \varphi \land \varphi)\} \beta$ since $\alpha \models \varphi \land \varphi$ and $\beta \leq \alpha$. (2): Suppose $R$ is in the right hand side. Then there exists $R_1 \in \{(R_1^+, \varphi)\} \beta$ such that $\alpha \models \varphi$ and $\beta \leq \alpha$. We fix an assignment $\delta$ from $\alpha \models \psi$ as $\delta(\text{ext}) \Rightarrow \delta(\text{ext}(r) \land \alpha_i(\varphi) \land \alpha_j(\varphi) )$ for $r \in S^*$. From the construction, $\alpha = a_1 \cup \delta$ is well defined. In similar to the (3) case, we can show that

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

We obtain $R \in \{(R_1^+, \varphi \land \varphi)\} \beta$ since $\alpha \models \varphi \land \varphi$ and $\beta \leq \alpha$. (2): Suppose $R$ is in the right hand side. Then there exists $R_1 \in \{(R_1^+, \varphi)\} \beta$ such that $\alpha \models \varphi$ and $\beta \leq \alpha$. We fix an assignment $\delta$ from $\alpha \models \psi$ as $\delta(\text{ext}) \Rightarrow \delta(\text{ext}(r) \land \alpha_i(\varphi) \land \alpha_j(\varphi) )$ for $r \in S^*$. From the construction, $\alpha = a_1 \cup \delta$ is well defined. In similar to the (3) case, we can show that

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

The work is supported by JSPS KAKENHI Grant Number JP17H01721.

REFERENCES