Extending Abstract Interpretation to Dependency Analysis of Database Applications

Angshuman Jana[1], Raju Halder[1], K. V. Abhishekh[1], S. D. Ganni[1], and Agostino Cortesi[2]
[1]Indian Institute of Technology Patna, India
{ajana.pcs13, halder, kalahasti.cs13, ganni.cs13}@iitp.ac.in
[2]Università Ca' Foscari Venezia, Italy
cortesi@unive.it

Abstract—Dependency information (data- and/or control-dependencies) among program variables and program statements is playing crucial roles in a wide range of software-engineering activities, e.g. program slicing, information flow security analysis, debugging, code-optimization, code-reuse, code-understanding. Most existing dependency analyzers focus on mainstream languages and they do not support database applications embedding queries and data-manipulation commands. The first extension to the languages for relational database management systems, proposed by Willmor et al. in 2004, suffers from the lack of precision in the analysis primarily due to its syntax-based computation and flow insensitivity. Since then no significant contribution is found in this research direction. This paper extends the Abstract Interpretation framework for static dependency analysis of database applications, providing a semantics-based computation tunable with respect to precision. More specifically, we instantiate dependency computation by using various relational and non-relational abstract domains, yielding to a detailed comparative analysis with respect to precision and efficiency. Finally, we present a prototype semDDA, a **sem**antics-based **D**atabase **D**ependency **A**nalyzer integrated with various abstract domains, and we present experimental evaluation results to establish the effectiveness of our approach. We show an improvement of the precision on an average of 6% in the interval, 11% in the octagon, 21% in the polyhedra and 7% in the powerset of intervals abstract domains, as compared to their syntax-based counterpart, for the chosen set of Java Server Page (JSP)-based open-source database-driven web applications as part of the GotoCode project.

Index Terms—Dependency Graphs, Static Analysis, Relational Databases, Structured Query Languages.

1 Introduction

Static analysis is recognized as a fundamental approach to collect information about the behavior of computer programs for all possible inputs, without performing any actual execution [1]. Over the past several decades, continuous and concerted research efforts in this direction make them powerful enough to solve many non-trivial questions about program's behavior, although they are undecidable in practice [1], [2]. Some notable and widely used static analysis techniques include Data-flow analysis [3], [4], Control-flow analysis [5], Type-based Theory [6], [7], [8], Abstract Interpretation [9], [10].

Observably most of the existing static analysis techniques in the literature make use, implicitly or explicitly, of dependency information among program statements and variables, solving a large number of software engineering tasks. Examples include information-flow security analysis [11], taint analysis [12], program slicing [13], optimization [14], [15], code-reuse [16], code-understanding [17]. A most common representation of these dependencies is *Dependency Graph* [18], [19], an intermediate form of programs which consists of both data- and control-dependencies among program components. Since the pioneer work by Ottenstein and Ottenstein [18], a number of variants of dependency graph for various programming languages are proposed by tuning them towards their suitable application domains. They are Program Dependency Graph (PDG) for intra-procedural programs [18], System Dependency Graph (SDG) for inter-procedural programs [20], Class Dependency Graph (ClDG) for object-oriented programs [21], Database-Oriented Program Dependency Graph (DOPDG) for database programs [22].

Although static analysis has been longly studied over the last several decades, researchers have not paid much attention to the case of database applications embedding database languages. In order to exploit the power of dependency graph in solving problems related to database applications, Willmor et al. [22] first introduced the notion of Database-Oriented Program Dependency Graph (DOPDG), considering the following two additional data dependencies due to the presence of database statements: (*i*) Program-Database dependency (PD-dependency) which represents dependency between an imperative statement and a database statement, and (*ii*) Database-Database dependency (DD-dependency) which represents a dependency between two database statements. However, since then no such notable contribution is found in this research direction. Some of the problems among many others which can effectively be addressed by using DOPDGs are:

(a) **Slicing of Database Applications.** Program slicing [23] is a well-known static analysis technique to address many software-engineering problems, including code understanding, debugging, maintenance, testing, parallelization, integration, software measurement [17], [24], [25], [26]. Existing program-slicing approaches have not considered external database states and therefore they are inapplicable to data-intensive programs in information system scenarios. It is imperative to say that slicing of database applications [27] based on their dependency information definitely serves as a powerful technique to solve the above-mentioned software-engineering problems relating query languages and underlying databases. In this context, preciseness of DOPDGs (hence slices) and their efficient computations are two prime factors which may affect the above-mentioned solutions to a great extent. This is yet to receive enough attention from the scientific community.

(b) **Database Leakage Analysis.** Language-based information-flow security analysis [28] has been longly studied during past decades to control illegitimate information leakage in software products. Needless to say, the confidentiality of sensitive database information can also possibly be compromised during their flow along database-applications accessing and processing them legitimately [29], [30]. The dependency information in the form of DOPDG can effectively capture any interference (if it exists) between sensitive and non-sensitive data. Of course, preciseness of dependency information highly matters to guarantee the absence of false security alarms in software products.

(c) **Data provenance.** Data provenance [31] is a static analysis technique which aids understanding and troubleshooting database queries by explaining the results in terms of input databases. Its intention is to show how (part of) the output of a query depended on (part of) its input. Precise dependency information among queries and identification of all parts of database information flowing along the program code are the basis of effective computations of data provenance.

(d) **Materialization View Creation.** Attribute dependencies are one of the prime factors for creating materialized views of databases [32]. The computation of precise static dependency information of database queries issued on a database over a certain period of time leads to a more precise materialized view creation.

A common challenge in all the above-mentioned application scenarios is to address the susceptibility of static dependency analysis to false positives, a main drawback of static analysis, which reduces development speed significantly. The best way to reduce false-positives is to allow tuning the analysis behavior towards specific needs. Our contribution in this paper on semantics-driven database dependency analyzer meets this challenge by facilitating precision control under various levels of abstractions.

To exemplify our motivation briefly, let us consider a small database code snippet, depicted in Figure 1, which increases salary of all employees by a common bonus amount *Cbonus* and by an additional special bonus amount *Sbonus* only for aged employees. Observe that the syntactic presence of *'sal'* as the defined-variable in Q_1 and as the used-variable in Q_3 makes Q_3 syntactically dependent on Q_1. However, a careful observation reveals that syntactic presence of variables as a way of dependency computation may often result in false positives, and thus fails to compute optimal set of dependencies. For instance, it is clear from the code that the values of *'sal'* referred in the "`WHERE`" clauses of Q_1 and Q_3 do not overlap with each other and this results in an independency between Q_1 and Q_3. This triggers a semantics-based approach to compute dependency where values instead of variables are considered. In this context, the following research question arises: *Are the values defined by one statement being used by another statement?* The problem to compute semantics-based dependency among statements in concrete domain is in general undecidable [2], [33]. This is also true in the case of database applications when the input database instance is *unknown*. Addressing similar problems in imperative languages, Mastroeni and Zanardini [34] introduced the notion of abstract semantics-based data dependency in the Abstract Interpretation framework. Abstract Interpretation [9], [10] is a widely used formal method which offers a sound approximation of the program's semantics to answer about the program's runtime behavior including undecidable ones. The intuition of Abstract Interpretation is to lift the concrete semantics to an abstract domain, by replacing concrete values by suitable properties of interests and simulating the operations in the abstract domain w.r.t. its concrete counterparts, in order to ensure sound semantic approximation.

```
Start;
Q0: Connection c =DriverManager.getConnection(......);
Q1: UPDATE emp SET sal = sal + Sbonus WHERE age ⩾ 60;
Q2: SELECT AVG(sal) FROM emp WHERE  age ⩾ 60;
Q3: SELECT AVG(sal) FROM emp WHERE  age < 60
Q4: UPDATE emp SET sal = sal + Cbonus;
Q5: SELECT AVG(sal) FROM emp;
Stop;
```

Fig. 1: An Introductory Example

Willmor's definition for DOPDG is not fully semantics-based [22]: although they define DD-dependency in terms of defined- and used-values of databases, their definition of PD-dependency relies on the syntactic presence of variables and attributes in statements. Intuitively, the precision of DOPDG depends on how precisely one can identify the overlapping of database-parts by various database operations (`INSERT`, `UPDATE`, `DELETE`). Although they refer to the Condition-Action rules [35] to compute the overlapping of database-parts, this fails to capture semantic independencies when the application contains more than one database statements defining (in sequence) the same attribute which is subsequently used by another database statement. The main reason behind this is the flow-insensitivity of the Condition-Action rules. For example, in Figure 1, Q_5 is semantically independent on Q_1 as the part of *sal*-values defined by Q_1 is fully redefined by Q_4 and never reaches Q_5. Unfortunately, Condition-Action rules can not capture this independency as the approach checks every pair of database statements independently, and as a result, this finds dependency when the pair Q_1 and Q_5 is encountered.

As the values of database attributes differ from that of imperative language variables, the computation of abstract semantics (and hence semantics-based dependency) of database applications is, however, challenging and requires different treatment. The key point here is the static identification of various parts of the database information possibly accessed or manipulated by database statements at various program points. Addressing these challenges, in this paper, we aim to answer the following two main research objectives:

- How to obtain more precise dependency information (hence more precise DOPDG)? and
- How to compute them efficiently?

To summarize, our contributions in this paper[1] are:

- Adapting the Abstract Interpretation framework to define computable abstract semantics of database applications, even in an undecidable scenario when the input database instance is unknown.
- Design of an algorithm to compute semantics-based independencies among database statements based on Abstract Semantics.
- A detailed analytical study on precision vs. efficiency when computing dependency in various well-suited non-relational and relational abstract domains, e.g. Interval, Octagon, Polyhedra, Powerset Domain.
- Development of a prototype "semDDA", Semantics-based Database Dependency Analyzer integrated with various abstract domains, which enables users to perform precise dependency computation in various abstract domains of interest.
- Experimental evaluation on a set of open-source database-driven JSP web applications as part of the GotoCode project [37] using our semDDA tool[2]. Experiments demonstrate the results in different abstract domains with a detailed comparison on precision and efficiency. This clearly shows that our technique improves precision w.r.t. the proposal by Willmor et al. [22].

Our preliminary theoretical proposal in [36] considered only polyherdra abstract domain. To be specific, the improvements in this paper compared to [36] are: (*i*) strengthening the approach by instantiating dependency computation using various non-relational and relational abstract domains, yielding a detailed comparative analysis with respect to precision and efficiency, (*ii*) implementation of semDDA which is tunable at various levels of abstractions, (*iii*) validation of our approach by performing experiments on a set of open-source database-driven JSP web applications using semDDA, establishing the effectiveness of our approach w.r.t. the literature.

The structure of the rest of paper is as follows: Section 2 illustrates a running example. In section 3 we describe the evolution of syntax-based dependency computation of database programs. Section 4 recalls some basics on syntax and concrete semantics of programs embedding SQL statements. In section 5 we describe semantics-based dependency computation in the concrete domain, whereas in section 6 we lift semantics-based dependency computation from the concrete domain to various non-relational and relational abstract domains. Section 7 illustrates the proposed framework on our running example. The soundness of our approach is proved in section 8. Section 9 describes the experimental results on a set of benchmark codes. We present in section 10 a case study on database code slicing, witnessing an improvement in the precision, by applying the proposed semantics-based dependency computation framework. We discuss in section 11 the current state-of-the-art in the literature. Finally, section 12 concludes our work.

1. This work is a revised and extended version of [36].
2. Available at: https://github.com/angshumanjana/SemDDA.

2 A Running Example

Consider the database code snippet "`Prog`" depicted in Figure 2. The code implements a module which provides a set of offers on various purchases made on an online shopping system.

The main method of the class saleOffer updates the purchase amount (stored in the attribute *purchase_amt*) depending on various discount offers. For instance, a customer will get 5% discount if the purchase amount is between 1000 USD and 3000 USD. Similarly, a 10% of discount is offered on the purchase amount more than 3000 USD. A special offer on waiving delivery charges is also given for all customers (program point 7). Finally, the module increments the points accumulated by its customers depending on both the purchase amount and the wallet balance at program points 15 and 16.

Observing the code carefully, we can identify a number of dependencies among the statements in "`Prog`". Some of them, although exist syntactically, may not be valid dependencies when we consider semantics of the program. For example, although statement 6 is syntactically DD-dependent on statement 5, but they are semantically independent as the values of the attribute *purchase_amt* defined by statement 5 can never be used by statement 6. In the subsequent sections, we pursue various existing approaches to refine dependency information, and finally we propose an abstract interpretation-based approach to approximate defined and used database parts by database statements (at various levels of abstractions) and hence to compute semantics-based dependencies among them based on the overlapping. We show, in section 7, how the proposed approach effectively identifies false DD-dependencies in '`Prog`".

3 Revisiting Syntax-based Dependency Computation in Database Applications

This section briefly discusses the evolution of syntax-based Database-Oriented Program Dependency Graph (DOPDG) construction and its limitations w.r.t. the literature. Throughout this paper we shall use the terms "Program" and "Database Program" synonymously. Similarly, we shall use the term "Statement" which synonymously refers to either "imperative statement" or "database statement" depending on the context.

3.1 Pure Syntax-based DOPDGs

The construction of pure syntax-based Database-Oriented Program Dependency Graph (DOPDG) is straightforward. It is an extension of traditional Program Dependency Graphs (PDGs) [18] to the case of database programs, considering the following three kinds of data-dependencies: (1) Program-Program dependency (PP-dependency) which represents a dependency between two imperative statements, (2) Program-Database dependency (PD-dependency) which represents a dependency between a SQL statement

```
0. public class saleOffer {
1.    public static void main(String[] args) throws SQLException {
2.      float x = 0.1;
3.      float y = 0.05;
4.      try { Statement con = DriverManager.getConnection("jdbc mysql: . . . ", "scott", "tiger").createStatement();
          /* 5% discount offer based on the purchase amount. */
5.          con.executeQuery("UPDATE Sales SET purchase_amt = purchase_amt − y ∗ purchase_amt WHERE purchase_amt BETWEEN 1000 AND 3000 ");
          /* 10% discount offer based on the purchase amount. */
6.          con.executeQuery("UPDATE Sales SET purchase_amt = purchase_amt − x ∗ purchase_amt WHERE purchase_amt > 3000 ");
          /* Free delivery offer. */
7.          con.executeQuery("UPDATE Sales SET purchase_amt = purchase_amt − delivery_charge ");
          . . .
          . . .
          . . .
11.         ResultSet rs=con.executeQuery("SELECT cust_name, purchase_amt FROM Sales WHERE purchase_amt ⩾ 200 ");
          . . .
          . . .
          . . .
          /* Points increment based on the purchase amount and wallet balance. */
15.         con.executeUpdate("UPDATE Sales SET point = point + 2 WHERE (purchase_amt + wallet_bal) ⩾ 5000 AND (purchase_amt + wallet_bal) < 10000 ");
16.         con.executeUpdate("UPDATE Sales SET point = point + 4 WHERE (purchase_amt + wallet_bal) ⩾ 10000 "); } catch (Exception e) { . . . } }}
```

Fig. 2: Database Code Snippet "Prog"

and an imperative statement, and (3) Database-Database dependency (DD-dependency) which represents a dependency between two SQL statements. This is to observe that syntax-based PP-dependencies and control dependencies in DOPDGs are the same as syntax-based data-dependencies and control-dependencies in PDGs respectively[3]. Let us define them below:

Definition 1 (Program-Program (PP) dependency [18]). An imperative statement I_2 is PP-dependent on another imperative statement I_1 if there exists an application variable x such that: (*i*) x is defined by I_1, (*ii*) x is used by I_2, and (*iii*) there is a x-definition free path from I_1 to I_2.

Definition 2 (Program-Database (PD) dependency [22]). A database statement Q is PD-dependent on an imperative statement I if there exists an application variable x such that: (*i*) x is defined by I, (*ii*) x is used as an input to Q, and (*iii*) there is a x-definition free path from I to Q. Similarly, an imperative statement I is PD-dependency on a database statement Q if there exists an application variable x such that: (*i*) the execution of Q sets x to be equal to one of the output of Q, (*ii*) x is used by I, and (*iii*) there is a x-definition free path from Q to I.

Definition 3 (Database-Database (DD) dependency). A database statement Q_2 is DD-dependent on another database statement Q_1 for an attribute a (denoted $Q_1 \xrightarrow{a} Q_2$) if the following conditions hold: (*i*) a is defined by Q_1, (*ii*) a is used by Q_2, and (*iii*) there is no rollback operation in between them, which undoes the effect of Q_1 on a.

The syntax-based dependency computation depends on the syntactic presence of one variable in the definition of another variable or on the control structure of the program. Let $\mathbb{C}$, $\mathbb{V}_a$ and $\mathbb{V}_d$ be the sets of statements, application-variables and database-attributes in database programs. Let $\mathbb{V} = \mathbb{V}_a \cup \mathbb{V}_d$ where $\mathbb{V}_a \cap \mathbb{V}_d = \emptyset$. The construction of syntax-based DOPDG can be formalized based on the two following functions:

$$\mathsf{USE}: \ \mathbb{C} \rightarrow \wp(\mathbb{V}) \tag{1}$$

$$\mathsf{DEF}: \ \mathbb{C} \rightarrow \wp(\mathbb{V}) \tag{2}$$

which extract the set of variables (either application-variables or database-attributes) *used* and *defined* in a statement $c \in \mathbb{C}$.

The following example illustrates the construction of pure syntax-based DOPDG using the above functions.

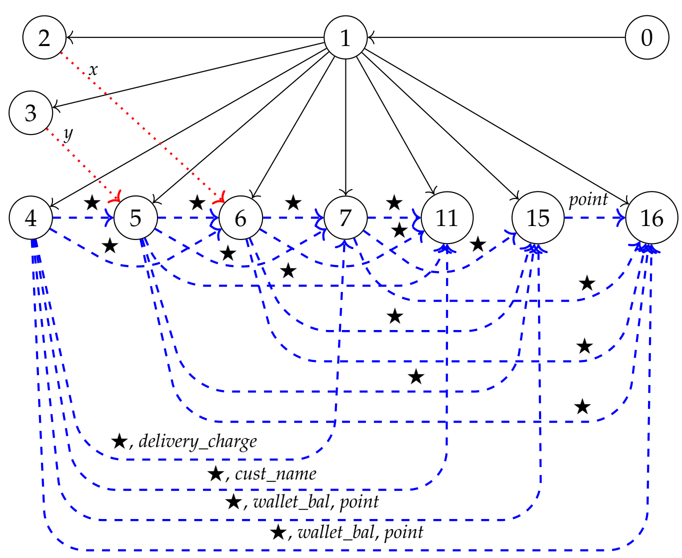


Fig. 3: Pure Syntax-based DOPDG (★ denotes attribute *purchase_amt*) of "Prog"

Example 1. Consider our running example "Prog" depicted in Figures 2. The control dependencies 1→2, 1→3, 1→4, etc. are computed in similar way as in the case of traditional PDG. The *used* and *defined* variables at each program point of "Prog" are computed as follows:

3. In the rest of the paper, we represent DD-, PD-, control-dependencies by blue dashed-line, red dotted-line and black line respectively.

DEF(2) $=\{x\}$ DEF(3) $=\{y\}$
DEF(4) $=\{purchase\_amt, delivery\_charge, cust\_name, wallet\_bal, point\}$
DEF(5) $=\{purchase\_amt\}$ USE(5) $=\{purchase\_amt,\ y\}$
DEF(6) $=\{purchase\_amt\}$ USE(6) $=\{purchase\_amt,\ x\}$
DEF(7) $=\{purchase\_amt\}$
USE(7) $=\{purchase\_amt,\ delivery\_charge\}$
USE(11)$=\{purchase\_amt,\ cust\_name\}$
DEF(15)$=\{point\}$
USE(15)$=\{purchase\_amt,\ wallet\_bal,\ point\}$
DEF(16)$=\{point\}$
USE(16)$=\{purchase\_amt,\ wallet\_bal,\ point\}$

Observe that statement 4 defines all database attributes as it connects to the database, resulting DEF(4) to contain all attributes. From the above information, the following data dependencies are identified:

- DD-dependencies for *purchase_amt*: $4 \dashrightarrow 5$, $4 \dashrightarrow 6$, $4 \dashrightarrow 7$, $4 \dashrightarrow 11$, $4 \dashrightarrow 15$, $4 \dashrightarrow 16$, $5 \dashrightarrow 6$, $5 \dashrightarrow 7$, $5 \dashrightarrow 11$, $5 \dashrightarrow 15$, $5 \dashrightarrow 16$, $6 \dashrightarrow 7$, $6 \dashrightarrow 11$, $6 \dashrightarrow 15$, $6 \dashrightarrow 16$, $7 \dashrightarrow 11$, $7 \dashrightarrow 15$, $7 \dashrightarrow 16$,
- DD-dependencies for other attributes: $4 \dashrightarrow 7$, $4 \dashrightarrow 11$, $4 \dashrightarrow 15$, $4 \dashrightarrow 16$, $15 \dashrightarrow 16$
- PD-dependencies for x and y: $2 \rightsquigarrow 6$, $3 \rightsquigarrow 5$

The syntax-based DOPDG of "`Prog`" is depicted in Figure 3.

Limitations. Syntax-based dependency computation often introduces false dependencies, leading to an imprecise analysis. For instance, in Example 1, although the statement 6 is syntactically DD-dependent on statement 5, however one can observe that the values of the attribute *purchase_amt* defined by statement 5 can never be used by statement 6. This is also true for $15 \dashrightarrow 16$. Similarly observe that the redefinition of all values of *purchase_amt* at program point 7 makes the statements 11, 15 and 16 data-independent on statements 4, 5 and 6 for *purchase_amt*, which is not captured here.

3.2 An Improved Syntax-driven Construction of DOPDGs

We proposed in [38] an improvement over the syntax-driven DOPDG construction algorithm by tagging variables with labels which indicate whether a variable is *fully-defined* or *partially-defined*. This enables us to (partially) identify a number of false dependencies.

The modified definitions of USE and DEF functions are as follows:

$$\text{USE}: \mathbb{C} \to \wp(\mathbb{V} \times \mathbb{L}) \tag{3}$$
$$\text{DEF}: \mathbb{C} \to \wp(\mathbb{V} \times \mathbb{L}) \tag{4}$$

where $\mathbb{L} = \{●, ◒\}$ is a set of labels. The label ● associated with an attribute a indicates that a is *fully-defined* – which means all values of a in the database are defined by the database statement. On other hand, the label ◒ associated with a indicates that a is *partially-defined* – which means only a subset of the values of a in the database are defined. Observe that these *fully-* and *partially-defined* distinctions are also applicable to program variables representing collections, such as arrays, lists, etc. For ordinary variable holding single value, the label is by default ● (i.e., *fully-defined*). Let us illustrate this on our running example.

***Example* 2.** Applying equations 3 and 4 on all statements in "`Prog`" of the running example, we get the following information:

DEF(2) $=\{(x, ●)\}$ DEF(3) $=\{(y, ●)\}$
DEF(4) $=\{(purchase\_amt, ●), (cust\_name, ●), (point, ●), (wallet\_bal, ●), (delivery\_charge, ●)\}$
DEF(5) $=\{(purchase\_amt, ◒)\}$
USE(5) $=\{(purchase\_amt, ◒), (y, ●)\}$
DEF(6) $=\{(purchase\_amt, ◒)\}$
USE(6) $=\{(purchase\_amt, ◒), (x, ●)\}$
DEF(7) $=\{(purchase\_amt, ●)\}$
USE(7) $=\{(purchase\_amt, ●), (delivery\_charge, ●)\}$
USE(11)$=\{(purchase\_amt, ◒), (cust\_name, ◒)\}$
DEF(15)$=\{(point, ◒)\}$
USE(15)$=\{(purchase\_amt, ◒), (wallet\_bal, ◒), (point, ◒)\}$
DEF(16)$=\{(point, ◒)\}$
USE(16)$=\{(purchase\_amt, ◒), (wallet\_bal, ◒), (point, ◒)\}$

The above information results in the following refined set of data dependencies:

- DD-dependencies for *purchase_amt*: $4 \dashrightarrow 5$, $4 \dashrightarrow 6$, $4 \dashrightarrow 7$, $5 \dashrightarrow 6$, $5 \dashrightarrow 7$, $6 \dashrightarrow 7$, $7 \dashrightarrow 11$, $7 \dashrightarrow 15$, $7 \dashrightarrow 16$,
- DD-dependencies for other attributes: $4 \dashrightarrow 7$, $4 \dashrightarrow 11$, $4 \dashrightarrow 15$, $4 \dashrightarrow 16$, $15 \dashrightarrow 16$
- PD-dependencies for x and y: $2 \rightsquigarrow 6$, $3 \rightsquigarrow 5$

The label ● associated with *purchase_amt* in DEF(7) indicates that all values of *purchase_amt* are defined at program point 7. This means that all definitions of *purchase_amt* before 7 does not reach any of its use after 7, identifying false DD-dependencies $4 \dashrightarrow 11$, $5 \dashrightarrow 11$, $6 \dashrightarrow 11$, $4 \dashrightarrow 15$, $5 \dashrightarrow 15$, $6 \dashrightarrow 15$, $4 \dashrightarrow 16$, $5 \dashrightarrow 16$ and $6 \dashrightarrow 16$ for *purchase_amt*. Observe that the DD-dependency $4 \dashrightarrow 11$ exists for *cust_name* and dependencies $4 \dashrightarrow 15$, $4 \dashrightarrow 16$ exist for both *wallet_bal* and *point*. The improved syntax-based DOPDG of "`Prog`" is depicted in Figure 4.

Limitations. This improved DOPDG construction approach also fails to compute optimal dependency results, because of its syntactic bound. For example, the false DD-dependencies $5 \dashrightarrow 6$ for *purchase_amt* and $15 \dashrightarrow 16$ for *point* still remain unidentified.

3.3 DOPDG Construction on Condition-Action Rules

Although Willmor et al. [22] defined PD-dependency (Definition 2) in terms of syntax, however interestingly they

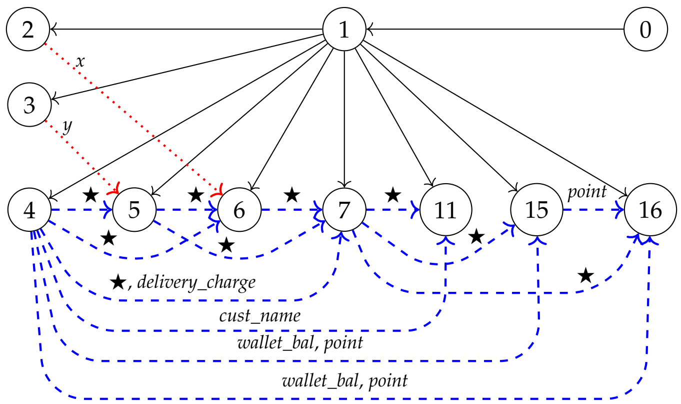


Fig. 4: Improved Syntax-based DOPDG of "`Prog`" (★ denotes attribute *purchase_amt*)

defined DD-dependency in terms of *defined* and *used* values (see Definition 4). This leads to an improvement in the precision of DD-dependency computation. However, the preciseness depends on how precisely one can identify the overlapping of database-parts by various database operations.

Definition 4 (Database-Database (DD) dependency [22]). Let $Q.SEL$, $Q.INS$, $Q.UPD$ and $Q.DEL$ denote the parts of database state which are selected, inserted, updated, and deleted respectively by Q. A database statement Q_1 is DD-dependent on another database statement Q_2 iff (*i*) the database-part defined by Q_2 overlaps the database-part used by Q_1, i.e. $Q_1.SEL \cap (Q_2.INS \cup Q_2.UPD \cup Q_2.DEL) \neq \emptyset$, and (*ii*) there is no roll-back operation in the execution path p between Q_2 and Q_1 (exclusive) which reverses back the effect of Q_2.

As a solution to compute this overlapping, Willmor et al. refer to the propagation algorithm in [35] designed for the static analysis of Condition-Action rules in expert database systems. The Condition-Action rules defined in an expert database system enable it to react automatically in some situations without the need of user access. These rules are, in general, expressed in the form $\mathsf{E}_{cond} \longrightarrow \mathsf{E}_{act}$, where E_{act} represents an action as data modification operation (e.g. `INSERT`, `UPDATE` and `DELETE`) and E_{cond} represents a condition. Formally, [35] considers an extended version of the relational algebra by introducing an additional operator ε, known as attribute extension operator, in case of database update. This operator is defined as $\varepsilon[x = expr]e$, where the expression $expr$ is evaluated over each tuple t of e and the resulting value is entered into the new attribute x for t under the new schema $schema(e) \cup \{x\}$. Let us illustrate this with running example.

Example 3. Consider our running example in Section 2. Following the extended relational algebra, we get the following Condition-Action rules at program points 5 and 6:

$$\mathsf{E}^5_{cond} \rightarrow \pi_{purchase\_amt}(\sigma_{purchase\_amt \geqslant 1000 \wedge purchase\_amt \leqslant 3000}\ \text{Sales})$$
$$\mathsf{E}^5_{act} \rightarrow \varepsilon[purchase\_amt' = purchase\_amt\ -\ 0.05 \times purchase\_amt](\sigma_{purchase\_amt \geqslant 1000 \wedge purchase\_amt \leqslant 3000}\ \text{Sales})$$
$$\mathsf{E}^6_{cond} \rightarrow \pi_{purchase\_amt}(\sigma_{purchase\_amt > 3000}\ \text{Sales})$$
$$\mathsf{E}^6_{act} \rightarrow \varepsilon[purchase\_amt' = purchase\_amt\ -\ 0.1 \times purchase\_amt](\sigma_{purchase\_amt > 3000}\ \text{Sales})$$

where π and σ are basic relational algebra operators for attribute projection and attribute selection respectively.

The propagation algorithm predicts how the action of one rule can affect the condition of another. In other words, the analysis checks whether a condition in one rule sees any data inserted or deleted or modified due to an action in another. This considers following three possibilities: (*i*) both the pre-defined part (i.e., database-part before performing the action E_{act}) and the post-defined part (i.e., database-part obtained after performing the action E_{act}) are in use by the condition E_{cond}; (*ii*) the pre-defined part is not in use by E_{cond} whereas the post-defined part is in use by E_{cond}, (*iii*) the pre-defined part is in use by E_{cond} whereas the post-defined part is not in use by E_{cond}. Let us illustrate this by recalling the rules already defined in Example 3. This is worthwhile to note here that this kind of conditions verification makes the computational complexity exponential w.r.t. the number of defining statements.

Example 4. Consider the Condition-Action rules at program points 5 and 6 of our running example expressed in Example 3. Observe that the predicates ($1000 \leqslant purchase\_amt \leqslant 3000$) in E^5_{act} and ($purchase\_amt > 3000$) in E^6_{cond} are contradictory – meaning that E^5_{act} operates on a part of data which is not accessed by E^6_{cond}. In other words, the action E^5_{act} does not affect the condition E^6_{cond}. Therefore, DD-dependency $5 \dashrightarrow 6$ is false. Similarly we can also identify another false DD-dependency $15 \dashrightarrow 16$. The refined set of data dependencies are:

- DD-dependencies for *purchase_amt*: $4 \dashrightarrow 5$, $4 \dashrightarrow 6$, $4 \dashrightarrow 7$, $4 \dashrightarrow 11$, $4 \dashrightarrow 15$, $4 \dashrightarrow 16$, $5 \dashrightarrow 7$, $5 \dashrightarrow 11$, $5 \dashrightarrow 15$, $5 \dashrightarrow 16$, $6 \dashrightarrow 7$, $6 \dashrightarrow 11$, $6 \dashrightarrow 15$, $6 \dashrightarrow 16$, $7 \dashrightarrow 11$, $7 \dashrightarrow 15$, $7 \dashrightarrow 16$
- DD-dependencies for other attributes: $4 \dashrightarrow 7$, $4 \dashrightarrow 11$, $4 \dashrightarrow 15$, $4 \dashrightarrow 16$
- PD-dependencies for x and y: $2 \dashrightarrow 6$, $3 \dashrightarrow 5$

Figure 5 depicts the refined DOPDG based on the above result.

Limitations. The Condition-Action rules can be applied only on a single *def-use* pair at a time. This fails to capture semantic independencies when a code contains more than one defining database statements (in sequence) for an attribute which is subsequently used by another database statement. The main reason behind this is the flow-insensitivity of this approach. For instance, the approach fails to identify false DD-dependencies $4 \dashrightarrow 11$, $4 \dashrightarrow 15$, $4 \dashrightarrow 16$,

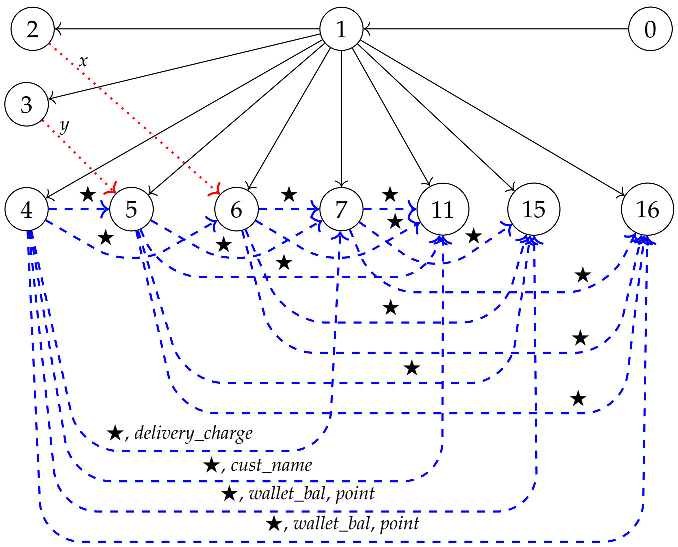


Fig. 5: Condition-Action Rules-based DOPDG of "`Prog`" (★ denotes attribute *purchase_amt*)

$5 \dashrightarrow 11$, $5 \dashrightarrow 15$, $5 \dashrightarrow 16$, $6 \dashrightarrow 11$, $6 \dashrightarrow 15$ and $6 \dashrightarrow 16$ in "`Prog`" due to the presence of multiple definitions of *purchase_amt* by the statements 5, 6 and 7 in sequence. Moreover, this approach incurs a high computational overhead w.r.t. program size. Observe that the algorithm combining from sections 3.2 and 3.3 will identify a set of false dependencies which is same as the union of the results obtained from both of the algorithms when applied individually.

The subsequent sections are dedicated to semantic-based DD-dependency (in concrete domain) and DD-independency (in abstract domain) computation of database programs.

4 Formal Syntax and Concrete Semantics of Database Query Languages

In this section, we recall from [39] the formal syntax and concrete semantics of database query languages.

Abstract syntax of database statement is denoted by $\langle A, \phi\rangle$ where A represents an action-part and ϕ represents a conditional-part. For instance, the query "`UPDATE` t `SET` sal=sal + 100 `WHERE` $age \geqslant 35$" is denoted by $\langle A, \phi\rangle$ where A represents "`UPDATE` t `SET` sal=sal + 100" and ϕ represents "$age \geqslant 35$".

Table 1 depicts the syntactic sets and the abstract syntax of database statements. The SQL clauses `GROUP BY`, `ORDER BY`, `DISTINCT/ALL`, and the aggregate functions are denoted by different functions $g()$, $f()$, $r()$, and $h()$ respectively. A SQL action A is either "`SELECT`" or "`UPDATE`" or "`DELETE`" or "`INSERT`". For example, the abstract syntax of the query above is denoted by

$$\langle \texttt{UPDATE}(\vec{v_d}, \vec{e}),\ \phi\rangle$$

where $\phi = (age \geqslant 35)$ and $\vec{v_d} = \langle sal\rangle$ and $\vec{e} = \langle sal + 100\rangle$.

It is worthwhile to mention that our defined abstract syntax, which we recall from our previous work [39], has limitation in the sense that it considers only numerical attributes. However, the syntax is consistent with the SQL definition given by ANSI [40]. In fact, we have shown its equivalence with relational algebra and its extension to support nested queries in sections 8 and 10 of [39] respectively. Therefore, our formalism supports different RDBMS implementations, like Oracle, MySQL or IBM DB2.

Application Environment. Given the set of application variables $\mathbb{V}_a$ and the domain of values `Val`, let $\mathfrak{E}_a : \mathbb{V}_a \mapsto \texttt{Val}$ be the set of all functions with domain $\mathbb{V}_a$ and range included in `Val`. An application environment $\rho_a \in \mathfrak{E}_a$ maps application variables to their values in `Val`.

Database Environment. A database d is a set of tables $\{t_i \mid i \in I_x\}$ for a given set of indexes I_x. A database environment is defined as a function ρ_d whose domain is I_x, such that for $i \in I_x$, $\rho_d(i) = t_i$.

Table Environment. Given a database table t with attributes $\texttt{attr}(t)$=$\{a_1, a_2, \ldots, a_k\}$. So, $t \subseteq D_1 \times D_2 \times \ldots. \times D_k$ where a_i is the attribute corresponding to the typed domain D_i. A *table environment* ρ_t for a table t is defined as a function such that for any attribute $a_i \in \texttt{attr}(t)$, $\rho_t(a_i) = \langle \pi_i(l_j) \mid l_j \in t\rangle$ where π is the projection operator and $\pi_i(l_j)$ represents the i^{th} element of the l_j-th row. In other words, ρ_t maps a_i to the ordered set of values over the rows of the table t.

Concrete Semantics. Let Σ_{dba} be the set of states for the database language under consideration, defined by $\Sigma_{dba} \triangleq \mathfrak{E}_{dbs} \times \mathfrak{E}_a$ where $\mathfrak{E}_{dbs}$ and $\mathfrak{E}_a$ denote the set of all database environments and the set of all application environments respectively. Therefore, a state $\rho \in \Sigma_{dba}$ is denoted by a tuple (ρ_d, ρ_a) where $\rho_d \in \mathfrak{E}_{dbs}$ and $\rho_a \in \mathfrak{E}_a$. The transition relation

$$\mathscr{T}_{dba} : (\mathbb{C} \times \Sigma_{dba}) \mapsto \wp(\Sigma_{dba}) \tag{5}$$

specifies which successor states $(\rho_{d'}, \rho_{a'}) \in \Sigma_{dba}$ can follow when a statement $c \in \mathbb{C}$ executes on state $(\rho_d, \rho_a) \in \Sigma_{dba}$. Let us illustrate the concrete semantics of an update statement.

Example 5.

Consider the database table t in Table 2(a) and the following update statement:

$$Q_{upd}:\ \texttt{UPDATE}\ t\ \texttt{SET}\ sal = sal + 100\ \texttt{WHERE}\ age \geqslant 35$$

The abstract syntax is denoted by $\langle \texttt{UPDATE}(\vec{v_d}, \vec{e}),\ \phi\rangle$ where $\phi = (age \geqslant 35)$ and $\vec{v_d} = \langle sal\rangle$ and $\vec{e} = \langle sal + 100\rangle$.

The table targeted by Q_{upd} is $target(Q_{upd})$= $\{t\}$. The semantics of Q_{upd} is:

$$\begin{aligned}
&\mathscr{T}_{dba}[\![Q_{upd}]\!](\rho_d, \rho_a)\\
&\quad = \mathscr{T}_{dba}[\![\Big\langle \texttt{UPDATE}(\langle sal\rangle, \langle sal + 100\rangle),\ (age \geqslant 35)\Big\rangle]\!](\rho_d, \rho_a)\\
&\quad = \mathscr{T}_{dba}[\![\Big\langle \texttt{UPDATE}(\langle sal\rangle, \langle sal + 100\rangle),\ (age \geqslant 35)\Big\rangle]\!](\rho_t, \rho_a)\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{Since, } target(Q_{upd})=\{t\}\\
&\quad = \mathscr{T}_{dba}[\![\texttt{UPDATE}(\langle sal\rangle, \langle sal + 100\rangle)]\!](\rho_{t\downarrow(age\geqslant 35)}, \rho_a)\\
&\qquad\quad \sqcup\\
&\quad (\rho_{t\downarrow\neg(age\geqslant 35)}, \rho_a) \qquad\qquad \textbf{Absorbing } \phi = (age \geqslant 35)
\end{aligned}$$

Constants:			
k	$\in$	$\mathbb{R}$	Set of Numerical Constants
Variables:			
v_a	$\in$	$\mathbb{V}_a$	Set of Application Variables
v_a	$::=$	$x \mid y \mid z \mid \ldots$	
v_d	$\in$	$\mathbb{V}_d$	Set of Database Attributes
v_d	$::=$	$a_1 \mid a_2 \mid a_3 \mid \ldots$	
$\mathbb{V}$	$::=$	$\mathbb{V}_a \cup \mathbb{V}_d$	
Expressions:			
e	$\in$	$\mathbb{E}$	Set of Arithmetic Expressions
e	$::=$	$k \mid v_d \mid v_a \mid op_u\ e \mid e_1\ op_b\ e_2$	where $op_u \in \{+, -\}$ and $op_b \in \{+, -, *, /\}$
b	$\in$	$\mathbb{B}$	Set of Boolean Expressions
b	$::=$	$\text{true} \mid \text{false} \mid e_1\ op_r\ e_2 \mid \neg b \mid b_1 \oplus b_2$	where $op_r \in \{\leq, \geq, ==, >, \neq, \ldots\}$ and $\oplus \in \{\vee, \wedge\}$
SQL Pre-conditions:			
τ	$\in$	$\mathbb{T}$	Set of Terms
τ	$::=$	$k \mid v_a \mid v_d \mid f_n(\tau_1, \tau_2, ..., \tau_n)$	where f_n is an n-ary function.
a_f	$\in$	$\mathbb{A}_f$	Set of Atomic Formulas
a_f	$::=$	$R_n(\tau_1, \tau_2, ..., \tau_n) \mid \tau_1 == \tau_2$	where $R_n(\tau_1, \tau_2, ..., \tau_n) \in \{true, false\}$
ϕ	$\in$	$\mathbb{W}$	Set of Pre-conditions
ϕ	$::=$	$a_f \mid \neg\phi \mid \phi_1 \oplus \phi_2 \mid \otimes v\ \phi$	where $\oplus \in \{\vee, \wedge\}$ and $\otimes \in \{\forall, \exists\}$
SQL Functions:			
$g(\vec{e})$	$::=$	$\texttt{GROUP BY}(\vec{e}) \mid id$	where $\vec{e} = \langle e_1, ..., e_n \mid e_i \in \mathbb{E}\rangle$ and id denotes identity function
r	$::=$	$\texttt{DISTINCT} \mid \texttt{ALL}$	
s	$::=$	$\texttt{AVG} \mid \texttt{SUM} \mid \texttt{MAX} \mid \texttt{MIN} \mid \texttt{COUNT} \mid id$	
$h(e)$	$::=$	$s \circ r(e)$	
$h(*)$	$::=$	$\texttt{COUNT(*)}$	where * represents the list of all database attributes denoted by $\vec{v_d}$.
$\vec{h}(\vec{x})$	$::=$	$\langle h_1(x_1), ..., h_n(x_n)\rangle$	where $\vec{h} = \langle h_1, ..., h_n\rangle$ and $\vec{x} = \langle x_1, ..., x_n \mid x_i = e \vee x_i = *\rangle$
$f(\vec{e})$	$::=$	$\texttt{ORDER BY ASC}(\vec{e}) \mid \texttt{ORDER BY DESC}(\vec{e}) \mid id$	
Commands:			
Q	$\in$	$\mathbb{Q}$	Set of SQL Statements
Q	$::=$	$Q_{sel} \mid Q_{upd} \mid Q_{ins} \mid Q_{del}$	
Q_{sel}	$::=$	$\langle A_{sel},\ \phi\rangle$	
	$::=$	$\Big\langle \texttt{SELECT}\Big(f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2,\ g(\vec{e})\Big),\ \phi_1\Big\rangle$	
Q_{upd}	$::=$	$\langle A_{upd},\ \phi\rangle$	
	$::=$	$\langle \texttt{UPDATE}(\vec{v_d}, \vec{e}),\ \phi\rangle$	
Q_{ins}	$::=$	$\langle A_{ins},\ \phi\rangle$	
	$::=$	$\langle \texttt{INSERT}(\vec{v_d}, \vec{e}),\ false\rangle$	
Q_{del}	$::=$	$\langle A_{del},\ \phi\rangle$	
	$::=$	$\langle \texttt{DELETE}(\vec{v_d}),\ \phi\rangle$	
c	$\in$	$\mathbb{C}$	Set of Commands
c	$::=$	$skip \mid v_a = e \mid Q \mid \text{if } b \text{ then } c \text{ endif}$	
	$\mid$	$\text{if } b \text{ then } c_1 \text{ else } c_2 \text{ endif}$	
	$\mid$	$\text{while } b \text{ do } c \text{ done}$	
$\mathcal{P}$	$::=$	$c \mid c\ ;\ \mathcal{P}$	Program

TABLE 1: Abstract Syntax of Programs embedding SQL

eid	*sal*	*age*	*dno*
1	1500	35	10
2	800	28	20
3	2500	50	10
4	3000	62	10

(a) table t

eid	*sal*	*age*	*dno*
1	1600	35	10
2	800	28	20
3	2600	50	10
4	3100	62	10

(b) table t''

TABLE 2: Database before and after the update operation

$$
\begin{aligned}
&=(\rho_{t'}, \rho_a)\ \sqcup\ (\rho_{t\downarrow\neg(age\geqslant 35)}, \rho_a)\\
&=(\rho_{t'} \sqcup \rho_{t\downarrow\neg(age\geqslant 35)},\ \rho_a)\\
&=(\rho_{t''}, \rho_a)
\end{aligned}
$$

where

$$
\begin{aligned}
\rho_{t'} \equiv& \rho_{t\downarrow\,(age\,\geqslant\,35)}\Big[sal \ \leftarrow E[\![sal\ +\ 100]\!](\rho_{t\downarrow\,(age\,\geqslant\,35)},\ \ \rho_a)\Big]\\
=& \rho_{t\downarrow\,(age\,\geqslant\,35)}\,[sal \ \leftarrow \langle 1600,\ \ 2600,\ \ 3100\rangle]
\end{aligned}
$$

The notation $(t \downarrow (age \geqslant 35))$ denotes the set of tuples in t for which $(age \geqslant 35)$ is true (denoted by red part in t of Table 2(a)). $E[\![.]\!]$ is the semantic function for arithmetic expression which maps "$sal + 100$" to a list of values $\langle 1600, 2600, 3100\rangle$ on the table environment $\rho_{t\downarrow(age\geqslant 35)}$. The notation $\leftarrow$ denotes a substitution by new values. Observe that the substitution of 'sal' by the list of values in $\rho_{t\downarrow(age\geqslant 35)}$ results in a new table environment $\rho_{t'}$ (denoted by red part in Table 2(b)). Finally, the least upper bound (denoted $\sqcup$) which is defined over the lattice of table environments partially ordered by $\subseteq$, results in a new state $(\rho_{t''}, \rho_a)$ where t'' is depicted in Table 2(b).

5 Semantics-based Dependency: A Formalization in Concrete Domain

As witnessed in section 3, the DOPDG construction approaches based on the syntax often fail to compute optimal set of dependencies. This motivates researchers towards semantics-based dependency computation considering values rather than variables [34]. For instance, consider an arithmetic expression "$e = x^2 + 4w \bmod 2 + z$". Although in this expression e syntactically depends on w, semantically there is no dependency as the evaluation of "$4w \bmod 2$" is always zero.

We, in our previous work [36], formalized the notion of semantics-based dependencies of database programs. Let us first recall this and then in the subsequent sections we build a computational framework considering this as the basis.

Given a SQL statement $Q = \langle A, \phi\rangle$ and its target table t. Suppose $\vec{x} = \texttt{USE}(A)$, $\vec{y} = \texttt{USE}(\phi)$ and $\vec{z} = \texttt{DEF}(\text{Q})$. According to the concrete semantics, suppose $\mathscr{T}_{dba}[\![Q]\!](\rho_t, \rho_a) = (\rho_{t'}, \rho_a)$.

The *used* and *defined* part of t by Q are computed according to the following equations:

$$\mathbf{A}_{def}(Q, t) = \Delta(\rho_{t'}(\vec{z}), \rho_t(\vec{z})) \tag{6}$$

$$\mathbf{A}_{use}(Q, t) = \rho_{t\downarrow\phi}(\vec{x}) \cup \rho_{t\downarrow\phi}(\vec{y}) \tag{7}$$

where

$t \downarrow \phi$	:	Set of tuples in table t which satisfies the condition-part ϕ.
$\rho_{t\downarrow\phi}(\vec{x})$	:	Values of $\vec{x}$ in $(t \downarrow \phi)$.
$\rho_{t\downarrow\phi}(\vec{y})$	:	Values of $\vec{y}$ in $(t \downarrow \phi)$.
Δ	:	Computes the difference between the original database state on which Q operates and the new database state obtained after performing the action-part A.

In other words, the function $\mathbf{A}_{use}$ maps a query Q to the part of the database information used by it, whereas the function $\mathbf{A}_{def}$ defines the changes occurred in the database states when data is updated or deleted or inserted by Q. The following example illustrates this.

Example 6. Let us consider the concrete database table t shown in Table 2(a) and the following update statement:

$$Q_{upd}: \texttt{UPDATE}\ t\ \texttt{SET}\ sal = sal + 100\ \texttt{WHERE}\ age \geqslant 35$$

where $A = \texttt{UPDATE}(\langle sal\rangle, \langle sal + 100\rangle)$ and $\phi = age \geqslant 35$. According to equations 6 and 7, the *used*-part and *defined*-part are as follows:

$$\mathbf{A}_{use}(Q_{upd}, t) = \rho_{t\downarrow(age\geqslant 35)}(sal) \cup \rho_{t\downarrow(age\geqslant 35)}(age)$$

$$\mathbf{A}_{def}(Q_{upd}, t) = \Delta(\rho_{t'}(sal), \rho_t(sal))$$

These are depicted in Tables 3(a) and 3(b) respectively where we have denoted $\mathbf{A}_{use}(Q_{upd}, t)$ and $\mathbf{A}_{def}(Q_{upd}, t)$ by red color.

eid	*sal*	*age*	*dno*
1	1500	35	10
2	800	28	20
3	2500	50	10
4	3000	62	10

(a) $\mathbf{A}_{use}(Q_{upd}, t)$

eid	*sal*	*age*	*dno*
1	1600	35	10
2	800	28	20
3	2600	50	10
4	3100	62	10

(b) $\mathbf{A}_{def}(Q_{upd}, t)$

TABLE 3: The *used* and *defined* part of t by Q_{upd} (marked with red color)

Given two database statements $Q_1 = \langle A_1, \phi_1\rangle$ and $Q_2 = \langle A_2, \phi_2\rangle$ such that $target(Q_1) = t$ and $\mathscr{T}_{dba}[\![Q_1]\!](\rho_t, \rho_a) = (\rho_{t'}, \rho_a)$ and $target(Q_2) = t'$. Following the equations 6 and 7, we can compute the *defined* part of t by Q_1 as $\mathbf{A}_{def}(Q_1, t)$ and *used*-part of t' by Q_2 as $\mathbf{A}_{use}(Q_2, t')$. Therefore, we can say Q_2 is DD-dependent on Q_1 when $\mathbf{A}_{def}(Q_1, t)$ and $\mathbf{A}_{use}(Q_2, t')$ overlap with each other, i.e. $\mathbf{A}_{use}(Q_2, t') \cap \mathbf{A}_{def}(Q_1, t) \neq \emptyset$. Observe that Q_1 is either `UPDATE`, `INSERT` and `DELETE` statement which defines the database. This is defined in Definition 5.

Definition 5 (Semantics-based DD-dependency [27]). A SQL statement $Q_2 = \langle A_2, \phi_2\rangle$ with $target(Q_2) = t'$ is DD-dependent for Υ on another SQL statement $Q_1 = \langle A_1, \phi_1\rangle$ with $target(Q_1) = t$ (denoted $Q_1 \xrightarrow{\Upsilon} Q_2$) if $Q_1 \in \{Q_{upd}, Q_{ins}, Q_{del}\}$ and $\mathscr{T}_{dba}[\![Q_1]\!](\rho_t, \rho_a) = (\rho_{t'}, \rho_a)$ and the overlapping-part $\Upsilon = \mathbf{A}_{use}(Q_2, t') \cap \mathbf{A}_{def}(Q_1, t) \neq \emptyset$.

When an initial database instance is unknown, due to infiniteness of the concrete domains, the computation of concrete semantics of database programs and hence $\mathbf{A}_{use}$, $\mathbf{A}_{def}$ and Υ become undecidable problem. Nevertheless, in case of finite large scale databases, these semantics-based dependency computations also incur in high computational overhead. To ameliorate this performance bottleneck, we apply the Abstract Interpretation theory [9] to compute abstract semantics of database languages, in a decidable way, as a sound approximation of its concrete counterparts.

6 Semantics-based Abstract dependency: A Sound Approximation

In this section, we first briefly introduce the Abstract Interpretation framework [9], [10]. Then we define abstract semantics of database statements in various non-relational and relational abstract domains. Finally, we present the computation of abstract dependencies among statements identifying their approximated *used* and *defined* database-parts based on the abstract semantics.

6.1 The Abstract Interpretation Framework: Preliminaries

Abstract Interpretation is a method of sound approximation of the program's concrete semantics which enables to provide sound answers to questions about the program's run-time behaviour. The idea is to lift concrete semantics to an abstract setting by replacing concrete values by suitable properties of interest, and simulating the concrete operations by sound abstract operations. The concrete and the abstract domains are partially ordered sets (lattices, or complete lattices, possibly), where the ordering relations describe the relative precision of the denotations including the top elements representing no information. The mapping between concrete and abstract semantics domains is usually established by a Galois Connection [4]:

Definition 6 (Galois Connections [9]). Consider two partial orders $(\mathbb{D}, \leqslant)$ and $(\overline{\mathbb{D}}, \sqsubseteq)$ where the first one represents a concrete domain and the second one represents an abstract domain. The Galois Connection between $\mathbb{D}$ and $\overline{\mathbb{D}}$ is denoted by $\left\langle (\mathbb{D}, \leqslant), \alpha, \gamma, (\overline{\mathbb{D}}, \sqsubseteq) \right\rangle$ or $(\mathbb{D}, \leqslant) \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} (\overline{\mathbb{D}}, \sqsubseteq)$ where α: $\mathbb{D} \rightarrow \overline{\mathbb{D}}$ and γ: $\overline{\mathbb{D}} \rightarrow \mathbb{D}$ holds iff:

- $\forall v \in \mathbb{D}.\ v \sqsubseteq \gamma \circ \alpha(v)$.
- $\forall \overline{v} \in \overline{\mathbb{D}}.\ \alpha \circ \gamma(\overline{v}) \sqsubseteq \overline{v}$.
- α and γ are monotonic.

In other words, iff $\forall v \in \mathbb{D}, \overline{v} \in \overline{\mathbb{D}}.\ \alpha(v) \sqsubseteq \overline{v} \iff v \leqslant \gamma(\overline{v})$.

A number of abstract domains, non-relational and relational, exist in the literature [9], [10], [41], [42], [43]. Let us briefly illustrate them below:

Non-relational Abstract Domains.

An abstract domain is said to be non-relational if it does not preserve any relation among program variables. Non-relational abstract domains care only about the actual variables being updated, rather than having potential to change multiple values at once [10]. Some widely used non-relational abstract domains for program analysis include sign domain for sign property analysis, parity domain for parity property analysis, interval domain for division-by-zero or overflows [9]. Analyses in these domains are, although efficient, but imprecise w.r.t. relational abstract domains. Figure 6 pictorially depicts a scenario where a set of points SP (indicated by $\bullet$) on the xy-plane are abstracted by sign and interval properties.

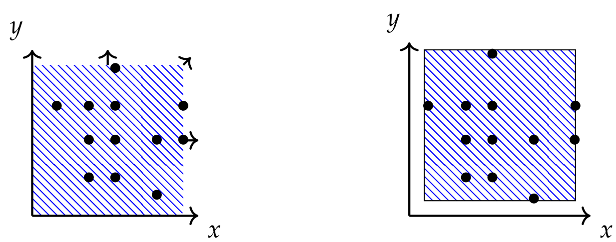


Fig. 6: Abstractions of SP by Sign (left) and Interval Properties (right)

Relational Abstract Domains.

Unlike non-relational abstract domains, the relational abstract domains preserve relations among program variables [41]. Analyses in these domains are more precise as compared to the non-relational abstract domains, in particular, for large number of relations among variables in the code. Widely used relational abstract domains are the domains of Polyhedra, Octagons, Difference-Bound Matrices (DBM), etc [41], [42], [43]. Abstractions of the same set of points in the octagon and polyhedra domains are exemplified in Figure 7.

4. Notice that for some abstract domains only a concretization function exists, like in the case of Polyhedra.

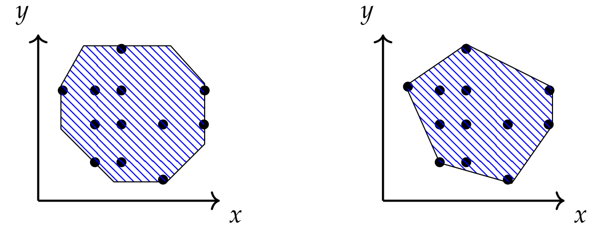


Fig. 7: Abstractions of SP in Octagon (left) and Polyhedra Domains (right)

6.2 Defining Abstract Semantics of Database Statements

We are now in a position to define the abstract semantics of database statements in various abstract domains. To this aim, let us first define abstract database states and the abstract semantic transition relation in an abstract domain of interest w.r.t. its concrete counterpart (section 4).

Definition 7 (Abstract Table). Given a concrete table $t \in \wp(D)$ where $D = D_1 \times D_2 \times \dots \times D_k$ such that $\mathtt{attr}(t)$=$\{a_1, a_2, \dots, a_k\}$ and a_i is the attribute corresponding to the typed domain D_i. Let $\overline{D}$ be an abstract domain which represents properties of the attributes of t establishing the Galois Connection [5] $\left\langle (\wp(D), \subseteq), \alpha, \gamma, (\overline{D}, \sqsubseteq) \right\rangle$. An element $\overline{t} \in \overline{D}$ is said to be a sound abstraction of the concrete table t if for all tuples $l \in t$, $l \in \gamma(\overline{t})$.

Definition 8 (Abstract Table Environment). Given an abstract table $\overline{t}$, an abstract table environment $\rho_{\overline{t}}$ is defined as $\rho_{\overline{t}}(a_i) = \overline{\pi}_i(\overline{t})$ for any attribute $a_i \in \mathtt{attr}(\overline{t})$, where $\overline{\pi}$ is the projection operator in the abstract domain and $\overline{\pi}_i(\overline{t})$ represents the projected abstract values corresponding to the i^{th} attribute in $\overline{t}$.

Definition 9 (Abstract Database States). An abstract database $\overline{d}$ is a set of abstract tables $\{\overline{t_i} \mid i \in I_x\}$ for a given set of indexes I_x. An abstract database environment is defined as a function $\rho_{\overline{d}}$ whose domain is I_x, such that for $i \in I_x$, $\rho_{\overline{d}}(i) = \overline{t_i}$.

Definition 10 (Abstract States). An abstract state $\overline{\rho} \in \overline{\Sigma}_{dba}$ for database applications is defined as a tuple $(\rho_{\overline{d}}, \rho_{\overline{a}})$ where $\rho_{\overline{d}} \in \overline{\mathfrak{E}}_{dbs}$ and $\rho_{\overline{a}} \in \overline{\mathfrak{E}}_a$ are an abstract database environment and an abstract application environment respectively.

Observe that, as any constraint defined at database level has no role in the dependency computation at application-code level, we consider database abstraction without taking these constraints into consideration.

In order to formalize the abstract semantics of database applications, we define the following sound abstract transition relation corresponding to its concrete counterpart $\mathscr{T}_{dba}$ (defined in equation 5 of section 4):

$$\overline{\mathscr{T}}_{dba} :\ \mathbb{C} \times \overline{\Sigma}_{dba} \mapsto \overline{\Sigma}_{dba} \tag{8}$$

which specifies the successor abstract state $(\rho_{\overline{d'}}, \rho_{\overline{a'}}) \in \overline{\Sigma}_{dba}$ when a statement $c \in \mathbb{C}$ executes on an abstract state

5. Notice that for some abstract domain the abstraction function may not exist.

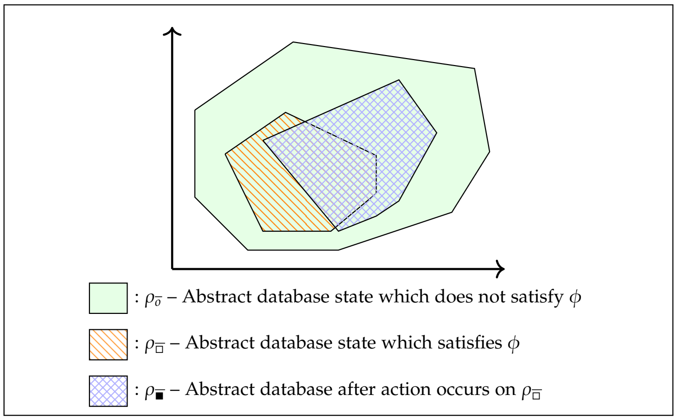


Fig. 8: Representation of abstract database state $\langle \rho_{\overline{o}}, \rho_{\overline{\square}}, \rho_{\overline{\blacksquare}} \rangle$

$(\rho_{\overline{d}}\ ,\ \rho_{\overline{a}}) \in \overline{\Sigma}_{dba}$. The soundness of the abstract semantics relies on the soundness of $\overline{\mathscr{T}}_{dba}$ w.r.t. $\mathscr{T}_{dba}$ (see the proof in section 8).

Since our objective is to compute semantics-based DD-independencies, it is important to identify database-parts (identified by the condition ϕ) before and after performing the action A. With this objective, unlike equation 8 which results in a single abstract state $\overline{\rho}$, we define a variant of the abstract transition relation as follows:

$$\overline{\mathscr{T}}_{dep} : \ \mathbb{C} \times \overline{\Sigma}_{dba} \mapsto (\overline{\mathfrak{E}}_{dbs} \times \overline{\mathfrak{E}}_{dbs} \times \overline{\mathfrak{E}}_{dbs}) \tag{9}$$

which results in a three-tuple $\langle \rho_{\overline{o}}, \rho_{\overline{\square}}, \rho_{\overline{\blacksquare}} \rangle$ of abstract database states, where $\rho_{\overline{o}}, \rho_{\overline{\square}}, \rho_{\overline{\blacksquare}} \in \overline{\mathfrak{E}}_{dbs}$. The first component $\rho_{\overline{o}}$ represents an abstract database state which does not satisfy ϕ, whereas the second component $\rho_{\overline{\square}}$ represents an abstract database state which satisfies ϕ. Observe that an abstract database-part which may or may not satisfy ϕ (due to abstraction) will be included in both $\rho_{\overline{o}}$ and $\rho_{\overline{\square}}$. The third component $\rho_{\overline{\blacksquare}}$ is obtained after performing an action A on $\rho_{\overline{\square}}$. These are depicted in Figure 8.

The abstract semantics of database statements in various abstract domains following equations 8 and 9 are defined in the subsequent sections.

6.2.1 *Domain of Intervals*

Let $\mathsf{L}_c = \langle \wp(\mathbb{R}), \subseteq, \emptyset, \mathbb{R}, \cap, \cup \rangle$ be a concrete lattice of the powerset of numerical values $\mathbb{R}$. Let $\mathbb{I} = \{[l,\ h] \mid l \in \mathbb{R} \cup \{-\infty\},\ h \in \mathbb{R} \cup \{+\infty\},\ l \leq h\} \cup \bot$ be the abstract domain of intervals forming an abstract lattice $\mathsf{L}_a = \langle \mathbb{I}, \sqsubseteq, \bot, [-\infty, +\infty], \sqcap, \sqcup \rangle$, such that:

- $[l_1,\ h_1] \sqsubseteq [l_2,\ h_2] \iff l_2 \leqslant l_1 \wedge h_2 \geqslant h_1$
- $[l_1,\ h_1] \sqcap [l_2,\ h_2] = [max(l_1\ l_2),\ min(h_1\ h_2)]$
- $[l_1,\ h_1] \sqcup [l_2,\ h_2] = [min(l_1,\ l_2),\ max(h_1\ h_2)]$

The correspondence between L_c and L_a is formalized as the Galois connection $\langle \mathsf{L}_c, \alpha_{\mathbb{I}},\ \gamma_{\mathbb{I}},\ \mathsf{L}_a \rangle$ where $\forall S \in \wp(\mathbb{R})$ and $\forall \overline{v} \in \mathbb{I}$:

$$\alpha_{\mathbb{I}}(S) = \begin{cases} \bot & \text{if } S = \emptyset \\ [l, h] & \text{if } min(S) = l\ \wedge\ max(S) = h \\ [-\infty, h] & \text{if } \nexists min(S)\ \wedge\ max(S) = h \\ [l, +\infty] & \text{if } min(S) = l\ \wedge\ \nexists max(S) \\ [+\infty,\ -\infty] & \text{if } \nexists min(S)\ \wedge\ \nexists max(S); \end{cases}$$

$$\gamma_{\mathbb{I}}(\overline{v}) = \begin{cases} \emptyset & \text{if } \overline{v} = \bot \\ \{k \in \mathbb{R} \mid\ l \leq k \leq h\} & \text{if } \overline{v} = [l,\ h] \\ \{k \in \mathbb{R} \mid\ k \leq h\} & \text{if } \overline{v} = [-\infty,\ h] \\ \{k \in \mathbb{R} \mid\ l \leq k\} & \text{if } \overline{v} = [l,\ +\infty] \\ \mathbb{R} & \text{if } \overline{v} = [+\infty,\ -\infty]. \end{cases}$$

The pictorial representation of the Galois connections $\langle \mathsf{L}_c,\ \alpha_{\mathbb{I}},\ \gamma_{\mathbb{I}},\ \mathsf{L}_a \rangle$ is shown in Figure 9.

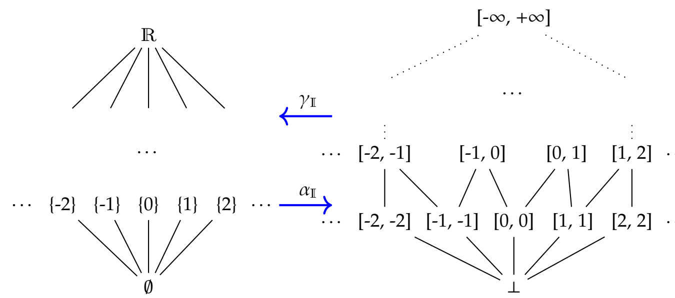


Fig. 9: Galois Connection between L_c and L_a

Abstract Semantics of Imperative Language in Interval: A Quick Tour [44].

Consider the set of concrete states $\Sigma\ :\ \ \mathbb{V}_a\ \mapsto\ \mathbb{R}$ representing the mapping of imperative program variables to their semantic domain values. Given the set of arithmetic expressions $\mathbb{E}$, boolean expressions $\mathbb{B}$ and commands $\mathbb{C}$. The concrete denotational semantics functions $\mathscr{T}_e\ :\ \ (\mathbb{E} \cup \mathbb{B})\ \mapsto\ (\Sigma\ \mapsto\ \mathbb{R} \cup \{true, false\})$ for expressions evaluation, $\mathscr{T}_f :\ \mathbb{B} \mapsto (\wp(\Sigma) \mapsto \wp(\Sigma))$ for state-filtering based on boolean satisfiability and $\mathscr{T}_c :\ \mathbb{C} \mapsto (\wp(\Sigma) \mapsto \wp(\Sigma))$ specifying effects of commands on states, are defined in Figure 10.

Given an abstract domain $\mathbb{I}$ of intervals, the set of abstract states is defined as $\overline{\Sigma}\ :\ \mathbb{V}_a\ \mapsto\ \mathbb{I}$ which respects the Galois Connection, i.e. $\forall\ \rho \in \Sigma, \forall\ \overline{\rho} \in \overline{\Sigma}$: $\alpha(\rho) \sqsubseteq \overline{\rho} \iff \rho \subseteq \gamma(\overline{\rho})$.

The corresponding sound abstract semantics function $\overline{\mathscr{T}_e}\ :\ \ (\mathbb{E} \cup \mathbb{B})\ \mapsto\ (\overline{\Sigma}\ \mapsto\ \mathbb{I} \cup \{true, false, \top_B\})$ for expression evaluation where $\top_B$ denotes " *may be true or may be false*", is defined as:

$$\begin{aligned}
\overline{\mathscr{T}_e}[\![k]\!] &= \{(\overline{\rho}, [k, k]) \mid \overline{\rho} \in \overline{\Sigma}\} \\
\overline{\mathscr{T}_e}[\![x]\!] &= \{(\overline{\rho}, \overline{\rho}(x)) \mid \overline{\rho} \in \overline{\Sigma}\} \\
\overline{\mathscr{T}_e}[\![e_1\ \oplus\ e_2]\!] &= \{(\overline{\rho}, \overline{v}_1 \mathbin{\overline{\oplus}} \overline{v}_2) \mid (\overline{\rho}, \overline{v}_1) \in \overline{\mathscr{T}_e}[\![e_1]\!], (\overline{\rho}, \overline{v}_2) \in \overline{\mathscr{T}_e}[\![e_2]\!]\} \\
\overline{\mathscr{T}_e}[\![e_1\ \circledcirc\ e_2]\!] &= \{(\overline{\rho}, \overline{v}_1 \mathbin{\overline{\circledcirc}} \overline{v}_2) \mid (\overline{\rho}, \overline{v}_1) \in \overline{\mathscr{T}_e}[\![e_1]\!], (\overline{\rho}, \overline{v}_2) \in \overline{\mathscr{T}_e}[\![e_2]\!]\}
\end{aligned}$$

Examples of sound abstract arithmetic and relational operations $\overline{\oplus}$ and $\overline{\circledcirc}$ respectively in the domain of intervals are:

$$\begin{aligned}
[l_1,\ h_1] \mathbin{\overline{+}} [l_2,\ h_2] &= [l_1 + l_2,\ h_1 + h2] \\
[l_1,\ h_1] \mathbin{\overline{\times}} [l_2,\ h_2] &= [\min(l_1 \times l_2, l_1 \times h_2, h_1 \times l_2, h_1 \times h_2), \\
&\qquad \max(l_1 \times l_2, l_1 \times h_2, h_1 \times l_2, h_1 \times h_2)]
\end{aligned}$$

Abstract versions of other arithmetic and relational operations are also defined this way, ensuring the soundness in $\mathbb{I}$.

Similarly, the abstract semantics functions $\overline{\mathscr{T}_f}\ :\ \ \mathbb{B}\ \mapsto\ (\overline{\Sigma} \mapsto \overline{\Sigma})$ for abstract state-filtering and $\overline{\mathscr{T}_c} :\ \mathbb{C} \mapsto (\overline{\Sigma} \mapsto \overline{\Sigma})$ for commands are:

$$\overline{\mathscr{T}_f}[\![x \leqslant k]\!] = \Big\{(\overline{\rho}, \overline{\rho}[x \leftarrow [l, \min(h,k)]]) \mid \overline{\rho} \in \overline{\Sigma}, \overline{\rho}(x) = [l,\ h],\ l \leqslant k\Big\}$$

$$
\begin{aligned}
\mathscr{T}_e[\![k]\!] =& \{(\rho,k) \mid \rho \in \Sigma\}\\
\mathscr{T}_e[\![x]\!] =& \{(\rho,v) \mid \rho \in \Sigma,\ \rho(x) = v\}\\
\mathscr{T}_e[\![e_1 \oplus e_2]\!] =& \{(\rho, v_1 \oplus v_2) \mid\ (\rho,v_1) \in \mathscr{T}_e[\![e_1]\!], (\rho,v_2) \in \mathscr{T}_e[\![e_2]\!],\\
& \oplus \in \{+,-,\times\} \vee (\oplus \in \{/,\%\} \wedge v_2 \neq 0)\}\\
\mathscr{T}_e[\![e_1 \circledcirc e_2]\!] =& \{(\rho, u_1 \circledcirc u_2) \mid (\rho,u_1) \in \mathscr{T}_e[\![e_1]\!], (\rho,u_2) \in \mathscr{T}_e[\![e_2]\!],\\
& \circledcirc \in \{\geqslant, \leqslant, <, >, ==\}\}\\
\mathscr{T}_e[\![\neg b]\!] =& \{(\rho, \neg w) \mid (\rho,w) \in \mathscr{T}_e[\![b]\!]\}\\
\mathscr{T}_e[\![b_1 \circledast b_2]\!] =& \{(\rho, w_1 \circledast w_2) \mid (\rho,w_1) \in \mathscr{T}_e[\![b_1]\!], (\rho,w_2) \in \mathscr{T}_e[\![b_2]\!],\\
& \circledast \in \{\vee, \wedge\}\}\\
\mathscr{T}_f[\![b]\!] =& \{(\rho,\rho)\ \mid\ \rho \in \Sigma,\ (\rho, true) \in \mathscr{T}_e[\![b]\!]\}\\
\mathscr{T}_c[\![skip]\!] =& \{(\rho,\rho)\ \mid\ \rho \in \Sigma\}\\
\mathscr{T}_c[\![x = e]\!] =& \{(\rho, \rho[x \leftarrow v])\ \mid\ \rho \in \Sigma,\ (\rho,v) \in \mathscr{T}_e[\![e]\!]\}\\
\mathscr{T}_c[\![c_1;c_2]\!] =& \mathscr{T}_c[\![c_1]\!] \circ \mathscr{T}_c[\![c_2]\!]\\
\mathscr{T}_c[\![&\text{if } b \text{ then } c_1 \text{ else } c_2]\!]\\
& = \{(\rho\ ,\ \rho')\ \mid\ (\rho\ ,\ \rho) \in \mathscr{T}_f[\![b]\!],\ (\rho\ ,\ \rho') \in \mathscr{T}_c[\![c_1]\!]\}\\
& \quad \cup\\
& \quad \{(\rho\ ,\ \rho'')\ \mid\ (\rho,\ \rho) \in \mathscr{T}_f[\![\neg b]\!],\ (\rho,\ \rho'') \in \mathscr{T}_c[\![c_2]\!]\}\\
\mathscr{T}_c[\![&\text{while } b \text{ do } c]\!]\\
& = \{(\rho\ ,\ \rho')\ \mid\ (\rho\ ,\ \rho') \in\\
& \qquad \mathscr{T}_f[\![\neg b]\!] \circ \text{lfp}\ \lambda Y.(\rho \cup \mathscr{T}_c[\![c]\!] \circ \mathscr{T}_f[\![b]\!]Y)\}
\end{aligned}
$$

Fig. 10: Concrete Denotational Semantics of simple Imperative Language

$$\overline{\mathscr{T}_f}[\![x \geqslant k]\!] = \Big\{(\overline{\rho}, \overline{\rho}[x \leftarrow [\max(l,k), h]]) \mid \overline{\rho} \in \overline{\Sigma}, \overline{\rho}(x) = [l,\ h],\ h \geqslant k\Big\}$$

$$\overline{\mathscr{T}_f}[\![x == k]\!] = \Big\{(\overline{\rho}, \overline{\rho}[x \leftarrow [k,k]]) \mid \overline{\rho} \in \overline{\Sigma}, \overline{\rho}(x) = [l,\ h],\ l\ \leqslant\ k \leqslant\ h\Big\}$$

$$
\begin{aligned}
\overline{\mathscr{T}_c}[\![skip]\!] =& \Big\{(\overline{\rho}, \overline{\rho})\ \mid\ \overline{\rho} \in \overline{\Sigma}\Big\}\\
\overline{\mathscr{T}_c}[\![x = e]\!] =& \Big\{(\overline{\rho}, \overline{\rho}[x \leftarrow \overline{v}]) \mid (\overline{\rho}, \overline{v}) \in \overline{\mathscr{T}_e}[\![e]\!]\Big\}\\
\overline{\mathscr{T}_c}[\![&\text{if } b \text{ then } c_1 \text{ else } c_2]\!]\\
& = \{(\overline{\rho},\ \overline{\rho}_3)\ \mid\ (\overline{\rho},\ \overline{\rho}_1) \in \overline{\mathscr{T}_f}[\![b]\!],\ (\overline{\rho}_1,\ \overline{\rho}_3) \in \overline{\mathscr{T}_c}[\![c_1]\!]\}\\
& \quad \sqcup\\
& \quad \{(\overline{\rho},\ \overline{\rho}_4)\ \mid\ (\overline{\rho},\ \overline{\rho}_2) \in \overline{\mathscr{T}_f}[\![\neg b]\!],\ (\overline{\rho}_2,\ \overline{\rho}_4) \in \overline{\mathscr{T}_c}[\![c_2]\!]\}\\
& = \{(\overline{\rho},\ \overline{\rho}_3 \sqcup \overline{\rho}_4)\ \mid \overline{\rho} \in \overline{\Sigma}\}
\end{aligned}
$$

where $\sqcup$ denotes component-wise join operation in the abstract lattice L_a.

$$
\begin{aligned}
\overline{\mathscr{T}_c}[\![&\text{while } b \text{ do } c]\!]\\
& = \{(\overline{\rho}\ ,\ \overline{\rho}_1)\ \mid\ (\overline{\rho}\ ,\ \overline{\rho}_1) \in\\
& \qquad \overline{\mathscr{T}}_f[\![\neg b]\!] \circ \text{lfp}\ \lambda Y.(Y \nabla (\overline{\rho}\ \sqcup\ \overline{\mathscr{T}}_c[\![c]\!] \circ \overline{\mathscr{T}}_f[\![b]\!]Y))\}
\end{aligned}
$$

where $\nabla : (\mathbb{I} \times \mathbb{I}) \rightarrow \mathbb{I}$ is a widening operator, if:

- for each $x, y \in \mathbb{I}$: $x \sqsubseteq x \nabla y$ and $y \sqsubseteq x \nabla y$.
- for each increasing chain $x_0 \sqsubseteq x_1 \sqsubseteq \ldots$, the increasing chain defined by $y_0 = x_0$, $y_{n+1} = y_n \nabla x_{n+1}$ for $n \in \mathbb{N}$, is not strictly increasing.

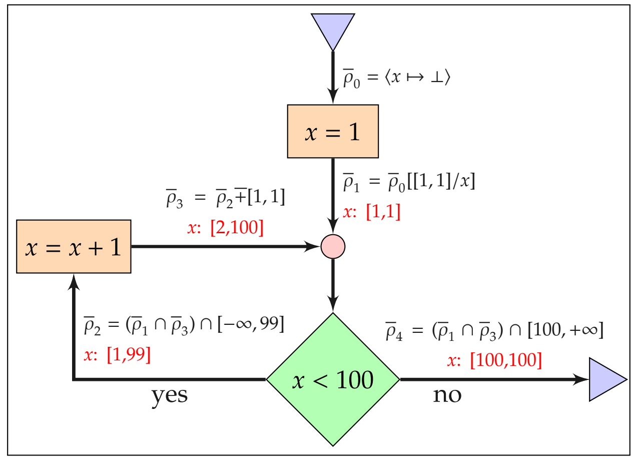


Fig. 11: An example of interval analysis

Example 7. Consider the statement $c ::=$ if $x \geqslant 5$ then $x = x+y$ else $x = x - y$. Consider an abstract state in the domain of intervals $\overline{\rho}=\langle x\ \mapsto\ [2,10], y\ \mapsto\ [1,1]\rangle$. The abstract semantics of c w.r.t. $\overline{\rho}$ is illustrated below:

$$
\begin{aligned}
\overline{\mathscr{T}_f}[\![x \geqslant 5]\!](\overline{\rho}) =& \overline{\rho}[x \leftarrow [5,10]] = \overline{\rho}_1\\
\overline{\mathscr{T}_f}[\![\neg(x \geqslant 5)]\!](\overline{\rho}) =& \overline{\rho}[x \leftarrow [2,4]] = \overline{\rho}_2\\
\overline{\mathscr{T}_e}[\![x + y]\!](\overline{\rho}_1) =& \overline{\rho}_1(x) \mathbin{\overline{+}} \overline{\rho}_1(y) = [6,11]\\
\overline{\mathscr{T}_e}[\![x - y]\!](\overline{\rho}_2) =& \overline{\rho}_2(x) \mathbin{\overline{-}} \overline{\rho}_2(y) = [1,3]\\
\overline{\mathscr{T}_c}[\![x = x + y]\!](\overline{\rho}_1) =& \overline{\rho}_1[x \leftarrow [6,11]] = \overline{\rho}_3\\
\overline{\mathscr{T}_c}[\![x = x - y]\!](\overline{\rho}_2) =& \overline{\rho}_2[x \leftarrow [1,3]] = \overline{\rho}_4\\
\overline{\mathscr{T}_c}[\![&\text{if } x \geqslant 5 \text{ then } x = x + y \text{ else } x = x - y]\!](\overline{\rho})\\
& = (\overline{\rho}_3 \sqcup \overline{\rho}_4) = \langle [6,11] \sqcup [1,3],\ [1,1] \sqcup [1,1] \rangle\\
& = \langle [1,11],\ [1,1] \rangle = \overline{\rho}_5
\end{aligned}
$$

Example 8. Consider the simple code fragment $x = 1$; while$(x < 100)\{x = x + 1;\}$. Figure 11 illustrates a data flow-based analysis of the code in $\mathbb{I}$ for the absence of runtime errors. The data-flow equation for each node is mentioned on the controlling edge of the corresponding node. The fix-point solution of these equations represent the abstract collecting semantics (denoted by red color).

Defining Abstract Semantics of Database Language in Interval Domain.

Let us recall the semantic function $\overline{\mathscr{T}}_{dba}$ defined in equation 8 which specifies the successor abstract state $(\rho_{\overline{d'}}\ ,\ \rho_{\overline{a'}}) \in \overline{\Sigma}_{dba}$ when a statement $c \in \mathbb{C}$ executes on an abstract state $(\rho_{\overline{d}}\ ,\ \rho_{\overline{a}}) \in \overline{\Sigma}_{dba}$.

Given a database statement $Q = \langle A, \phi \rangle$ and an abstract database state $\overline{\rho} = (\rho_{\overline{d}}\ ,\ \rho_{\overline{a}})$, the abstract semantics of Q w.r.t. $\overline{\rho}$ is defined below:

$$
\begin{aligned}
&\overline{\mathscr{T}}_{dba}[\![\langle A, \phi \rangle]\!]\overline{\rho}\\
&= \overline{\mathscr{T}}_{dba}[\![\langle A, \phi \rangle]\!](\rho_{\overline{d}}\ ,\ \rho_{\overline{a}})\\
&= \overline{\mathscr{T}}_{dba}[\![\langle A, \phi \rangle]\!](\rho_{\overline{t}}\ ,\ \rho_{\overline{a}})\\
&\qquad \text{where } t = target(\langle A, \phi \rangle) \text{ and } \exists \overline{t} \in \overline{d} :\ t \in \gamma(\overline{t})\\
&= \overline{\mathscr{T}}_{dba}[\![\langle A \rangle]\!](\rho_{\overline{TM}}\ ,\ \rho_{\overline{a}})\ \sqcup\ (\rho_{\overline{FM}}\ ,\ \rho_{\overline{a}})\\
&= (\rho_{\overline{TM'}}\ ,\ \rho_{\overline{a}})\ \sqcup\ (\rho_{\overline{FM}}\ ,\ \rho_{\overline{a}})
\end{aligned}
$$

$$=(\rho_{\overline{TM'}} \sqcup \rho_{\overline{FM}}\ ,\ \rho_{\overline{a}} \sqcup \rho_{\overline{a}})$$
$$=(\rho_{\overline{t'}}\ ,\ \rho_{\overline{a}}) \qquad (10)$$

where

$(\rho_{\overline{TM}}\ ,\ \rho_{\overline{a}}) \in \overline{\mathscr{T}_f}[\![\phi]\!](\rho_{\overline{t}}\ ,\ \rho_{\overline{a}})$ and $(\rho_{\overline{FM}}\ ,\ \rho_{\overline{a}}) \in \overline{\mathscr{T}_f}[\![\neg\phi]\!](\rho_{\overline{t}}\ ,\ \rho_{\overline{a}})$

Observe that $\overline{TM}$ and $\overline{FM}$ are the abstract database states obtained by using the filtering semantics function $\overline{\mathscr{T}_f}$ based on the satisfaction of ϕ. In particular, $\overline{TM}$ denotes the part of the abstract database state for which ϕ is true, whereas $\overline{FM}$ denotes the abstract database state for which ϕ is false. After performing the update action A on $\overline{TM}$, the resultant abstract state $\overline{TM'}$ is obtained. Finally, component-wise join operation between $\overline{TM'}$ and $\overline{FM}$ yields the resultant abstract state $\overline{t'}$. Observe that, in order to ensure the soundness, both $\overline{TM}$ and $\overline{FM}$ include the information for which ϕ results in "*may be true or false*". We illustrate this in Example 9.

Example 9. Consider the abstract domain of intervals $\mathbb{I}$. Given the concrete database table t shown in Table 4(a), its corresponding abstract version $\overline{t}$ replacing concrete values by their properties from $\mathbb{I}$ is depicted in Table 4(b). Similarly, given an application environment

eid	*sal*	*age*	*dno*
1	1500	35	10
2	800	28	20
3	2500	50	10
4	3000	62	10

(a) Concrete table t

eid	*sal*	*age*	*dno*
[1,4]	[800,3000]	[28,62]	[10, 20]

(b) Abstract table $\overline{t}$ in interval domain

TABLE 4: Concrete and its corresponding Abstract Database

$\rho_a = \langle x \mapsto 100\rangle$ where x is an application variable, its corresponding abstract application environment in $\mathbb{I}$ is $\rho_{\overline{a}} = \langle x \mapsto [100, 100]\rangle$.

Now consider the following `UPDATE` statement:

$$Q_{upd}:\ \texttt{UPDATE}\ t\ \texttt{SET}\ sal = sal + x\ \texttt{WHERE}\ sal \geqslant 1500$$

Here $A = \texttt{UPDATE}(\langle sal\rangle, \langle sal + x\rangle)$ and $\phi = sal \geqslant 1500$. The concrete semantics yields the resultant table t' shown in Table 5.

eid	*sal*	*age*	*dno*
1	1600	35	10
2	800	28	20
3	2600	50	10
4	3100	62	10

TABLE 5: Execution result t' by Q_{upd} on t

The abstract semantics of Q_{upd} w.r.t. $\overline{\rho} = (\rho_{\overline{t}}\ ,\ \rho_{\overline{a}})$ is

$$\overline{\mathscr{T}}_{dba}[\![\langle A, \phi\rangle]\!](\rho_{\overline{t}}\ ,\ \rho_{\overline{a}})$$
$$=\overline{\mathscr{T}}_{dba}\Big[\!\!\Big[\Big\langle \texttt{UPDATE}(\langle sal\rangle, \langle sal + x\rangle),\ sal \geqslant 1500\Big\rangle\Big]\!\!\Big](\rho_{\overline{t}}\ ,\ \rho_{\overline{a}})$$
$$=\overline{\mathscr{T}}_{dba}\Big[\!\!\Big[\Big\langle \texttt{UPDATE}(\langle sal\rangle, \langle sal + x\rangle)\Big\rangle\Big]\!\!\Big](\rho_{\overline{TM}}, \rho_{\overline{a}}) \sqcup (\rho_{\overline{FM}}, \rho_{\overline{a}})$$
$$=(\rho_{\overline{TM'}}\ ,\ \rho_{\overline{a}}) \sqcup (\rho_{\overline{FM}}\ ,\ \rho_{\overline{a}})$$
$$=(\rho_{\overline{TM'}} \sqcup \rho_{\overline{FM}}\ ,\ \rho_{\overline{a}} \sqcup \rho_{\overline{a}})$$
$$=(\rho_{\overline{t'}}\ ,\ \rho_{\overline{a}})$$

where

$$\rho_{\overline{TM}} = \overline{\mathscr{T}_f}[\![sal \geqslant 1500]\!](\rho_{\overline{t}}) = \rho_{\overline{t}}\Big[sal \leftarrow [1500,\ 3000]\Big]$$

$$\rho_{\overline{FM}} = \overline{\mathscr{T}_f}[\![\neg(sal \geqslant 1500)]\!](\rho_{\overline{t}}) = \rho_{\overline{t}}\Big[sal \leftarrow [800,\ 1499]\Big]$$

$$\rho_{\overline{TM'}} = \overline{\mathscr{T}}_{dba}[\![\texttt{UPDATE}(\langle sal\rangle, \langle sal + x\rangle)]\!](\rho_{\overline{TM}}\ ,\ \rho_{\overline{a}})$$
$$= \overline{\mathscr{T}_c}[\![sal = sal + x]\!](\rho_{\overline{TM}}\ ,\ \rho_{\overline{a}})$$
$$= \overline{\mathscr{T}_c}[\![sal = sal + [100, 100]]\!](\rho_{\overline{TM}}\ ,\ \rho_{\overline{a}})$$
$$= \rho_{\overline{TM}}\Big[sal \leftarrow [1600, 3100]\Big]$$

Tables 6(a) and 6(b) depict $\overline{TM}$ and $\overline{FM}$ respectively. After performing the update action A on $\overline{TM}$, the resultant abstract table $\overline{TM'}$ is shown in Table 6(c). Finally, component-wise join operation between $\overline{TM'}$ and $\overline{FM}$ yields the resultant table $\overline{t'}$ depicted in Table 6(d). Observe that the abstract semantics is sound, i.e. $t' \in \gamma(\overline{t'})$.

eid	*sal*	*age*	*dno*
[1,4]	[1500,3000]	[28,62]	[10, 20]

(a) Abstract table $\overline{TM}$

eid	*sal*	*age*	*dno*
[1,4]	[800,1499]	[28,62]	[10, 20]

(b) Abstract table $\overline{FM}$

eid	*sal*	*age*	*dno*
[1,4]	[1600,3100]	[28,62]	[10, 20]

(c) Abstract table $\overline{TM'}$

eid	*sal*	*age*	*dno*
[1,4]	[800,3100]	[28,62]	[10, 20]

(d) Abstract table $\overline{t'} = \overline{TM'} \sqcup \overline{FM}$

TABLE 6: Execution results of Q_{upd} on $\overline{t}$

Limitations. Let us illustrate the limitation of the analysis in $\mathbb{I}$ by the following example.

Example 10. Consider the abstract state $\overline{\rho} = (\rho_{\overline{t}}\ ,\ \rho_{\overline{a}})$ defined in Example 9. Consider the following statement:

$$Q_{upd}:\ \texttt{UPDATE}\ t\ \texttt{SET}\ sal = sal + sal * 0.1\ \texttt{WHERE}\ sal + age \geqslant 1550$$

where $A = \texttt{UPDATE}(\langle sal\rangle, \langle sal + sal * 0.1\rangle)$ and $\phi = sal + age \geqslant 1550$. Observe that, as the semantics of ϕ on $\rho_{\overline{t}}$ results in "*may be true or false*", therefore both $\rho_{\overline{TM}}$ and $\rho_{\overline{FM}}$ include all abstract database-parts as depicted in Table 7. Therefore, analysis-results in interval domain of such cases are highly approximated, which may lead to the computation of false dependency.

eid	*sal*	*age*	*dno*
[1,4]	[800,3000]	[28,62]	[10, 20]

(a) Abstract table $\overline{TM}$

eid	*sal*	*age*	*dno*
[1,4]	[800,3000]	[28,62]	[10, 20]

(b) Abstract table $\overline{FM}$

TABLE 7: Part of execution results of Q_{upd} on $\overline{t}$

Abstract Semantics towards Independency Computation. In order to compute semantics-based DD-independencies, we define $\overline{\mathscr{T}}_{dep}$, according to equations 9, in $\mathbb{I}$ for database statements as follows:

$$\overline{\mathscr{T}}_{dep}[\![\langle A, \phi\rangle]\!]\overline{\rho}$$
$$=\overline{\mathscr{T}}_{dep}[\![\langle A, \phi\rangle]\!](\rho_{\overline{d}}\ ,\ \rho_{\overline{a}})$$
$$=\overline{\mathscr{T}}_{dep}[\![\langle A, \phi\rangle]\!](\rho_{\overline{t}}\ ,\ \rho_{\overline{a}})$$
$$\text{where } t = target(\langle A, \phi\rangle) \text{ and } \exists \overline{t} \in \overline{d}:\ t \in \gamma(\overline{t})$$
$$=\Big\langle \rho_{\overline{FM}}\ ,\ \rho_{\overline{TM}}\ ,\ \rho_{\overline{TM'}}\Big\rangle \qquad (11)$$

where

- $\overline{\mathscr{T}_f}[\![\phi]\!](\rho_{\overline{t}}\ ,\ \rho_{\overline{a}}) = (\rho_{\overline{TM}}\ ,\ \rho_{\overline{a}})$

- $\overline{\mathscr{T}_f}[\![\neg\phi]\!](\rho_{\bar{t}}\ ,\ \rho_{\bar{a}}) = (\rho_{\overline{FM}}\ ,\ \rho_{\bar{a}})$
- $\overline{\mathscr{T}_c}[\![A]\!](\rho_{\overline{TM}}\ ,\ \rho_{\bar{a}}) = (\rho_{\overline{TM'}}\ ,\ \rho_{\bar{a}})$

Let us now define $\overline{\mathscr{T}}_{dep}$ for UPDATE, DELETE, INSERT and SELECT.

UPDATE statement:

$$\overline{\mathscr{T}}_{dep}[\![\Big\langle \texttt{UPDATE}(\vec{v}_d,\ \vec{e}),\ \ \phi\Big\rangle]\!](\rho_{\bar{t}}\ ,\ \rho_{\bar{a}}) \ = \ \Big\langle \rho_{\overline{FM}}\ ,\ \rho_{\overline{TM}}\ ,\ \rho_{\overline{TM'}}\Big\rangle$$

where $\overline{\mathscr{T}_f}[\![\neg\phi]\!](\rho_{\bar{t}}\ ,\ \rho_{\bar{a}}) = (\rho_{\overline{FM}}\ ,\ \rho_{\bar{a}})$

$\overline{\mathscr{T}_f}[\![\phi]\!](\rho_{\bar{t}}\ ,\ \rho_{\bar{a}}) = (\rho_{\overline{TM}}\ ,\ \rho_{\bar{a}})$

$\overline{\mathscr{T}_c}[\![\texttt{UPDATE}(\vec{v}_d,\vec{e})]\!](\rho_{\overline{TM}}\ ,\ \rho_{\bar{a}})$

$= (\rho_{\overline{TM}}[\vec{v}_d\ \leftarrow \text{-}\mathscr{T}_e[\![\vec{e}]\!](\rho_{\overline{TM}}\ ,\ \rho_{\bar{a}})],\ \rho_{\bar{a}}) = (\rho_{\overline{TM'}}\ ,\ \rho_{\bar{a}})$

INSERT statement:

$$\overline{\mathscr{T}}_{dep}[\![\Big\langle \texttt{INSERT}(\vec{v}_d,\ \vec{e}),\ \ false\Big\rangle]\!](\rho_{\bar{t}}\ ,\ \rho_{\bar{a}}) = \Big\langle \rho_{\bar{t}}\ ,\ \rho_{\perp}\ ,\ \rho_{\overline{new}}\Big\rangle$$

where $\overline{\mathscr{T}_f}[\![\neg false]\!](\rho_{\bar{t}}\ ,\ \rho_{\bar{a}}) = (\rho_{\bar{t}}\ ,\ \rho_{\bar{a}})$

$\overline{\mathscr{T}_f}[\![false]\!](\rho_{\bar{t}}\ ,\ \rho_{\bar{a}}) = (\rho_{\perp}\ ,\ \rho_{\bar{a}})$

$\overline{\mathscr{T}_c}[\![\texttt{INSERT}(\vec{v}_d\ ,\ \vec{e})]\!](\rho_{\perp}\ ,\ \rho_{\bar{a}})$

$= (\rho_{\perp}[\vec{v}_d\ \leftarrow\ \overline{\mathscr{T}_e}[\![\vec{e}]\!](\rho_{\perp}\ ,\ \rho_{\bar{a}})]\ ,\ \rho_{\bar{a}}) \ = \ (\rho_{\overline{new}}\ ,\ \rho_{\bar{a}})$

where $\rho_{\perp}$ maps the attributes to the bottom element in the abstract domain which represents "*undefined*" values.

DELETE statement:

$$\overline{\mathscr{T}}_{dep}[\![\Big\langle \texttt{DELETE}(\vec{v_d}),\ \ \phi\Big\rangle]\!](\rho_{\bar{t}}\ ,\ \rho_{\bar{a}}) \ = \ \Big\langle \rho_{\overline{FM}}\ ,\ \rho_{\overline{TM}}\ ,\ \rho_{\perp}\Big\rangle$$

SELECT statement:

$$\overline{\mathscr{T}}_{dep}[\![\Big\langle \texttt{SELECT}\Big(f(\vec{e}),\ \ r(\vec{h}(\vec{x})),\ \ \phi_2,\ \ g(\vec{e})\Big),\ \ \ \phi_1\Big\rangle]\!](\rho_{\bar{t}}\ ,\ \ \rho_{\bar{a}}) = \Big\langle \rho_{\overline{FM}}\ ,\ \rho_{\overline{TM}}\ ,\ \rho_{\overline{TM}}\Big\rangle$$

Observe that the select operation does not change any information.

Example 11. Consider the abstract state $\overline{\rho} = \langle\rho_{\bar{t}}\ ,\ \rho_{\bar{a}}\rangle$ and $\rho_{\bar{a}} = \langle x \mapsto [100,100]\rangle$ where $\bar{t}$ is depicted in Table 4(b), as defined in Example 9. Consider the following statements:

Q_{upd} =UPDATE t SET $sal = sal + x$ WHERE $sal \geqslant 1500$

Q_{ins} =INSERT INTO t (eid, sal, age, dno)VALUES$(5, 2700, 52, 20)$

Q_{del} =DELETE FROM t WHERE $age \geqslant 61$

Q_{sel} =SELECT age FROM t WHERE $age \leqslant 50$

The abstract syntax of the statements are:

$Q_{upd} = \langle \texttt{UPDATE}(\langle sal\rangle, \langle sal + x\rangle),\ sal \geqslant 1500\rangle$

$Q_{ins} = \langle \texttt{INSERT}(\langle eid,\ sal,\ age,\ dno\rangle,\ \langle 5, 2700, 52, 20\rangle),\ false\rangle$

$Q_{del} = \langle \texttt{DELETE}(\langle eid,\ sal,\ age,\ dno\rangle),\ age\ \geqslant\ 61\rangle$

$Q_{sel} = \langle \texttt{SELECT}(\langle age\rangle),\ age \leqslant 50\rangle$

Abstract semantics of Q_{upd} w.r.t. $\overline{\rho}$ is

$$\overline{\mathscr{T}}_{dep}[\![\Big\langle \texttt{UPDATE}(\langle sal\rangle,\ \ \langle sal\ +\ x\rangle),\ \ sal\ \geqslant\ 1500\Big\rangle]\!](\rho_{\bar{t}}\ ,\ \ \rho_{\bar{a}}) = \Big\langle \rho_{\overline{FM}}\ ,\ \rho_{\overline{TM}}\ ,\ \rho_{\overline{TM'}}\Big\rangle$$

where $\rho_{\overline{FM}}$, $\rho_{\overline{TM}}$ and $\rho_{\overline{TM'}}$ are shown in Table 6 of Example 9.

The abstract semantics of Q_{ins} w.r.t. $\overline{\rho}$ is

$$\overline{\mathscr{T}}_{dep}[\![\Big\langle \texttt{INSERT}(\langle eid, sal, age, dno\rangle, \langle 5, 2700, 52, 20\rangle), false\Big\rangle]\!](\rho_{\bar{t}}, \rho_{\bar{a}}) = \Big\langle \rho_{\bar{t}}\ ,\ \rho_{\perp}\ ,\ \rho_{\overline{new}}\Big\rangle \quad \text{where}$$

$$\rho_{\overline{new}} = \rho_{\perp}\Big[eid \leftarrow [5,\ 5],\ sal\ \leftarrow [2700,\ 2700],\ age\ \leftarrow [52,\ 52],\ dno \leftarrow [20,\ 20]\Big]$$

The abstract semantics of Q_{del} w.r.t. $\overline{\rho}$ is

$$\overline{\mathscr{T}}_{dep}[\![\Big\langle \texttt{DELETE}(\langle eid,\ \ sal,\ \ age,\ \ dno\rangle),\ \ age\ \geqslant\ 61\Big\rangle]\!](\rho_{\bar{t}}\ ,\ \rho_{\bar{a}}) = \Big\langle \rho_{\overline{FM}}\ ,\ \rho_{\overline{TM}}\ ,\ \rho_{\perp}\Big\rangle \quad \text{where}$$

$$\rho_{\overline{TM}} \ = \ \overline{\mathscr{T}_f}[\![\ age\ \geqslant\ 61\ ]\!](\rho_{\bar{t}}) \ = \ \rho_{\bar{t}}\Big[age\ \leftarrow\ [61,\ 62]\Big]$$

$$\rho_{\overline{FM}} \ = \ \overline{\mathscr{T}_f}[\![\neg(age\ \geqslant\ 61)\ ]\!](\rho_{\bar{t}}) \ = \ \rho_{\bar{t}}\Big[age\ \leftarrow\ [28,\ 60]\Big]$$

The abstract semantics of Q_{sel} w.r.t. $\overline{\rho}$ is

$$\overline{\mathscr{T}}_{dep}[\![\Big\langle \texttt{SELECT}(\langle age\rangle), age \leqslant\ 50\Big\rangle]\!](\rho_{\bar{t}}\ ,\ \rho_{\bar{a}}) = \Big\langle \rho_{\overline{FM}}\ ,\ \rho_{\overline{TM}}\ ,\ \rho_{\overline{TM}}\Big\rangle$$

where

$$\rho_{\overline{TM}} \ = \ \overline{\mathscr{T}_f}[\![\ age\ \leqslant\ 50\ ]\!](\rho_{\bar{t}}) \ = \ \rho_{\bar{t}}\Big[age\ \leftarrow\ [28,\ 50]\Big]$$

$$\rho_{\overline{FM}} \ = \ \overline{\mathscr{T}_f}[\![\neg(\ age\ \leqslant\ 50)\ ]\!](\rho_{\bar{t}}) \ = \ \rho_{\bar{t}}\Big[age\ \leftarrow\ [51,\ 62]\Big]$$

6.2.2 Relational Abstract Domain of Octagons

To yield more precise analysis as compared to the interval abstract domain, Antoine Miné [43] proposed a weekly relational abstract domain – the domain of octagons – which allows an analyzer to discover automatically common errors, such as division by zero, out-of-bound array access or deadlock, and more generally to prove safety properties of programs.

The octagon abstract domain encodes binary constraints between program variables in the form of $k_i x_i + k_j x_j \leqslant k$ where x_i, x_j are program variables, $k_i, k_j \in [-1, 0, 1]$ are coefficients and k is a constant in the numerical domain $\mathbb{R}$. Since coefficients can be either -1, 0 or 1, the number of inequalities between any two variables is bounded. The set of points satisfying the conjunction of such constraints forms an octagon.

Octagonal constraints representation in memory. The encoding of conjunctions of octagonal constraints makes use of Difference Bound Matrix (DBM) representation. Let us describe DBM first and then an extension to encode the set of octagonal constraints.

Difference Bound Matrices (DBM) [42]. Given a program $\mathcal{P}$ with a finite set of variables $\mathbb{V}_{\mathcal{P}} = \{x_1, \ldots, x_n\}$. A Difference Bound Matrix (DBM) m with size $n \times n$ represents a set of invariants each of the form $x_j - x_i \leqslant k$, where $k \in \mathbb{R}_\infty$ and $\mathbb{R}_\infty = \mathbb{R} \cup \{\infty\}$ such that:

$$\mathsf{m}_{ij} \triangleq \begin{cases} k & \text{if } (x_j - x_i \leqslant\ k)\ \text{ where } x_i, x_j \in \mathbb{V}_{\mathcal{P}} \text{ and } k \in \mathbb{R}_\infty, \\ \\ \infty & \text{otherwise.} \end{cases}$$

Example 12. Consider the constraints $\{x_1 - x_2 \leqslant 3, x_2 - x_3 \leqslant 4, x_3 - x_1 \leqslant 5, x_2 - x_4 \leqslant 4\}$. These constraints are represented by the DBM shown below:

	x_1	x_2	x_3	x_4
x_1	∞	∞	5	∞
x_2	3	∞	∞	∞
x_3	∞	4	∞	∞
x_4	∞	4	∞	∞

Extension to encode octagonal constraints [43]. The above DBM representation over program variables can represent only a subset of octagonal constraints of the form $x_i - x_j \leqslant k$. In order to allow more general form $\pm x_i \pm x_j \leqslant k$ of octagonal constraints, a DBM m of size $n \times n$ defined over $\mathbb{V}_{\mathcal{P}}$ is extended to another DBM m' of size $2n \times 2n$ over the set of enhanced variables $\mathbb{V}'_{\mathcal{P}} = \{x'_1, \ldots, x'_{2n}\}$ where each variable $x_i \in \mathbb{V}_{\mathcal{P}}$ comes in two forms: a positive form x'_{2i-1}, denoted x_i^+ and a negative form x'_{2i}, denoted x_i^-. This extended form of DBM m' is called coherent DBM (CDBM) representing octagon. This is illustrated in the following example.

Example 13. Consider the octagonal constraints $\{x_1 + x_2 \leqslant 3, x_1 - x_2 \leqslant 4, -x_1 - x_2 \leqslant 5, x_1 \leqslant 4\}$, its equivalent CDBM constraints are $\{x_1^+ - x_2^- \leqslant 3, x_2^+ - x_1^- \leqslant 3, x_1^+ - x_2^+ \leqslant 4, x_2^- - x_1^- \leqslant 4, x_1^- - x_2^+ \leqslant 5, x_2^- - x_1^+ \leqslant 5, x_1^+ - x_1^- \leqslant 8\}$. These constraints are represented in CDBM shown below:

	x_1^+	x_1^-	x_2^+	x_2^-
x_1^+	∞	∞	∞	5
x_1^-	8	∞	3	4
x_2^+	4	5	∞	∞
x_2^-	3	∞	∞	∞

Observe that any constraints of the form $(x_i \leqslant k)$ and $(x_i \geqslant k)$ can be represented as $(x_i^+ - x_i^- \leqslant 2k)$ and $(x_i^- - x_i^+ \leqslant -2k)$ respectively.

Closure. An octagon can be represented by more than one set of inequalities. For instance, the octagonal constraints $\{(x \leqslant 4) \wedge (y \leqslant 6)\}$ and $\{(x \leqslant 4) \wedge (y \leqslant 6) \wedge (x + y \leqslant 10)\}$ represent the same concrete values. Therefore, the use of closure operation ensures a unique representation of any octagonal constraints. The closure operation on CDBM follows Floyd – Warshall algorithm [45].

In the rest of the paper, we use the notation m to represent closed CDBM when the context is clear.

Galois Connections. Let $\mathsf{L}_c = \langle \wp(\mathbb{R}^n), \subseteq, \emptyset, \mathbb{R}^n, \cap, \cup \rangle$ be the concrete lattice. Let $\mathbb{M}$ be the set of all closed CDBMs representing the domain of octagons. Let $\mathbb{M}_\perp = \mathbb{M} \cup \{\mathsf{m}_\perp\}$ where $\mathsf{m}_\perp$ represents the bottom element that contains an unsatisfiable set of constraints. We define the abstract lattice $\mathsf{L}_a = \langle \mathbb{M}_\perp, \sqsubseteq, \mathsf{m}_\perp, \mathsf{m}_\top, \sqcap, \sqcup \rangle$ where $\mathsf{m}_\top$ represents the top element for which the bound for all constraints is ∞. The partial order, meet and join operations in L_a are defined as follows:

- $\forall \mathsf{m}, \mathsf{n} \in \mathbb{M}_\perp$: $\mathsf{m} \sqsubseteq \mathsf{n} \iff \forall i, j$: $\mathsf{m}_{ij} \leqslant \mathsf{n}_{ij}$.
- $\forall \mathsf{m}, \mathsf{n} \in \mathbb{M}_\perp$: $(\mathsf{m} \sqcap \mathsf{n}) = \mathsf{m}'$ where $\forall i, j$: $\mathsf{m}'_{ij} \triangleq \min(\mathsf{m}_{ij}, \mathsf{n}_{ij})$.
- $\forall \mathsf{m}, \mathsf{n} \in \mathbb{M}_\perp$: $(\mathsf{m} \sqcup \mathsf{n}) = \mathsf{m}'$ where $\forall i, j$: $\mathsf{m}'_{ij} \triangleq \max(\mathsf{m}_{ij}, \mathsf{n}_{ij})$.

Observe that since the union of two octagons is not always an octagon the result is approximated.

Let Σ be the set of all environments defined as $\Sigma : \mathbb{V} \mapsto \mathbb{R}$. An environment $\rho \in \Sigma$ maps each variable to its value. An environment will be understood as a point in $\mathbb{R}^n$ where $|\mathbb{V}| = n$. The Galois connection between L_c and L_a is formalized as $\langle \mathsf{L}_c, \alpha_{\mathbb{M}}, \gamma_{\mathbb{M}}, \mathsf{L}_a \rangle$ where $\alpha_{\mathbb{M}}$ and $\gamma_{\mathbb{M}}$ on $S \in \wp(\mathbb{R}^n)$ and $\mathsf{m} \in \mathbb{M}_\perp$ are defined below:

- if $S = \emptyset$: $\alpha_{\mathbb{M}}(S) \triangleq \mathsf{m}_\perp$
- if $S \neq \emptyset$: $\alpha_{\mathbb{M}}(S) = \mathsf{m}$ where $\mathsf{m}_{ij} \triangleq$

$$\begin{cases} \max\{\rho(x_l) - \rho(x_k) | \rho \in S\} & \text{when } i = 2k-1, j = 2l-1 \\ & \quad \text{or } i = 2l, j = 2k \\ \max\{\rho(x_l) + \rho(x_k) | \rho \in S\} & \text{when } i = 2k, j = 2l-1 \\ \max\{-\rho(x_l) - \rho(x_k) | \rho \in S\} & \text{when } i = 2k-1, j = 2l \end{cases}$$

$$\gamma_{\mathbb{M}}(\mathsf{m}) = \begin{cases} \emptyset & \text{if } \mathsf{m} = \mathsf{m}_\perp \\ \mathbb{R}^n & \text{if } \mathsf{m} = \mathsf{m}_\top \\ \Big\{(k_1, \ldots, k_n) \in \mathbb{R}^n \mid (k_1, -k_1 \ldots k_n, -k_n) & \\ \quad \in \mathsf{dom}(\mathsf{m}) \text{ and } \forall i, j : x_j - x_i \leqslant \mathsf{m}_{ij}\Big\} & \text{otherwise} \end{cases}$$

Sound operations in octagon domain. Let us recall from [43] some useful sound operations in octagon abstract domain defined in terms of CDBM:

- *Emptiness test*: Let m be a CDBM and G be a directed weighted graph of m. We say that the octagon is empty, i.e. $\gamma(\mathsf{m}) = \emptyset$, if and only if G has a simple cycle with a strictly negative total weight. The well-known Bellman-Ford [46] algorithm is used for such cycle detection.

- *Projection*: Let m be a CDBM representing a non empty octagon. We extract the values of the variable x_i from m in the form of interval as:

$$\{v \mid \exists (k_1 \ldots k_n) \in \gamma(\mathsf{m}) \text{ such that } k_i = v\}$$
$$= [-\mathsf{m}_{2i\ 2i+1}/2,\ \ \mathsf{m}_{2i+1\ 2i}/2]$$

Interested reader may refer to [43] [47] for more abstract operations (closure, widening, etc.) in octagon domain.

Abstract Semantics of Imperative Language in Octagon: A Quick Tour [43].

Given the set of boolean expressions $\mathbb{B}$ and commands $\mathbb{C}$. The concrete denotational semantics functions for state-filtering based on the boolean satisfiability is defined as $\mathscr{T}_f : \big(\mathbb{B} \mapsto \wp(\Sigma)\big) \mapsto \wp(\Sigma)$. The corresponding sound abstract function $\overline{\mathscr{T}_f}$ in the domain of octagons is defined as $\overline{\mathscr{T}_f} : (\mathbb{B} \mapsto \mathbb{M}_\perp) \mapsto \mathbb{M}_\perp$. Similarly, the concrete denotational semantic function for the effects of commands on states is defined as $\mathscr{T}_c : \big(\mathbb{C} \mapsto \wp(\Sigma)\big) \mapsto \wp(\Sigma)$ and its corresponding sound abstract function $\overline{\mathscr{T}_c}$ in octagon domain is defined as $\overline{\mathscr{T}_c} : (\mathbb{C} \mapsto \mathbb{M}_\perp) \mapsto \mathbb{M}_\perp$.

Test: Given a CDBM m representing abstract state at a program point and a boolean expression b. The state-filtering function $\overline{\mathscr{T}_f}$ finds m' applying b on m where $\gamma(\mathsf{m}')$ is $\{\rho \in \gamma(\mathsf{m}) \mid \rho \text{ satisfies } b\}$. However, as it is in general impossible to implement such a transition function, an upper approximation result is computed such that

$$\gamma(\mathsf{m}') \supseteq \{\rho \in \gamma(\mathsf{m}) \mid \rho \text{ satisfies } b\}$$

The tests that can be modeled in the octagon domain are: $x_h + x_l \leqslant k$, $x_h - x_l \leqslant k$, $-x_h - x_l \leqslant k$, $x_h + x_l = k$, $x_h \leqslant k$

and $x_h \geqslant k$. The state-filtering function $\overline{\mathscr{T}_f}$ for $x_h + x_l \leqslant k$ is defined as below:

$$\overline{\mathscr{T}_f}[\![x_h + x_l \leqslant k]\!]\mathsf{m} = \mathsf{m}' \text{ where}$$

$$\mathsf{m}'_{ij} \triangleq \begin{cases} \min(\mathsf{m}_{ij}, k) & \text{if } (i,j) \in \{(2h, 2l-1), (2l, 2h-1)\}, \\ \mathsf{m}_{ij} & \text{otherwise} \end{cases}$$

Observe that the entries in the CDBM m corresponding to the cells (x_h^-, x_l^+) and (x_l^-, x_h^+) are updated based on the value k, resulting into m' which satisfies $x_h + x_l \leqslant k$. Similarly $\overline{\mathscr{T}_f}$ for all others tests can also be defined.

Assignment: An assignment is to replace the value of a program variable x_i with the value of an expression e, formally $x_i = e$. Given an abstract state m representing octagonal constraints at a program point and an assignment $x_i = e$, the abstract semantics of the assignment on m results m' as an upper approximation such that

$$\gamma(\mathsf{m}') \supseteq \{\rho[x_i \leftarrow k] \mid \rho \in \gamma(\mathsf{m}) \wedge k = \mathscr{T}_e[\![e]\!]\rho\} \text{ where}$$

$\mathscr{T}_e$ is the semantic function of arithmetic expression and $\rho[x_i \leftarrow k]$ denote ρ with its i^{th} component changed into k.

The assignments that can be modeled in octagon domain are: $x_h = x_h + k$ and $x_h = x_l + k$ with $h \neq l$. In the first case $x_h = x_h + k$, we subtract k from inequalities having negative coefficient for x_h and we add k to inequalities having positive coefficient for x_h. On the other hand, for the second case $x_h = x_l + k$, the inequalities $x_h - x_l \leqslant k$ and $x_l - x_h \leqslant -k$ are added into the octagon. Let us illustrate them below:

1) If $x_h = x_h + k$:
$\overline{\mathscr{T}_c}[\![x_h = x_h + k]\!]\mathsf{m} = \mathsf{m}'$ where $\mathsf{m}'_{ij} \triangleq \mathsf{m}_{ij} + (\alpha_{ij} + \beta_{ij})k$
with
$$\alpha_{ij} \triangleq \begin{cases} +1 & \text{if } j = 2h, \\ \text{-1} & \text{if } j = 2h-1, \\ 0 & \text{otherwise} \end{cases}$$
and
$$\beta_{ij} \triangleq \begin{cases} \text{-1} & \text{if } i = 2h, \\ +1 & \text{if } i = 2h-1, \\ 0 & \text{otherwise} \end{cases}$$

2) If $x_h = x_l + k$ with $h \neq l$:
$\overline{\mathscr{T}_c}[\![x_h = x_l + k]\!]\mathsf{m} = \mathsf{m}'$ where
$$\mathsf{m}'_{ij} \triangleq \begin{cases} \text{k} & \text{if } (i,j) \in \{(2h, 2l); (2l-1, 2h-1)\}, \\ \text{-k} & \text{if } (i,j) \in \{(2l, 2h); (2h-1, 2l-1), \\ \mathsf{m}_{ij} & \text{if } i,j \notin \{2h, 2h-1\}, \\ +\infty & \text{otherwise} \end{cases}$$

Example 14. Consider the statement c ::= if $x \geqslant 5$ then $x = y + 1$ else $x = y - 1$. Let the initial abstract state be $\mathsf{m}_\top$ (which is the top element in the lattice of octagon abstract domain). The abstract semantics of c w.r.t. m is illustrated below, where $\stackrel{\text{rep}}{=}$ denotes an alternative representation in memory.

$$\overline{\mathscr{T}_f}[\![x \geqslant 5]\!]\mathsf{m}_\top = \{-x \leqslant -5\} \stackrel{\text{rep}}{=} \mathsf{m}_1$$
$$\overline{\mathscr{T}_f}[\![\neg(x \geqslant 5)]\!]\mathsf{m}_\top = \{x \leqslant 4\} \stackrel{\text{rep}}{=} \mathsf{m}_2$$
$$\overline{\mathscr{T}_c}[\![x = y + 1]\!]\mathsf{m}_1 = \{x - y \leqslant 1, y - x \leqslant -1, -x \leqslant -5\} \stackrel{\text{rep}}{=} \mathsf{m}_3$$
$$\overline{\mathscr{T}_c}[\![x = y - 1]\!]\mathsf{m}_1 = \{x - y \leqslant -1, y - x \leqslant 1, x \leqslant 4\} \stackrel{\text{rep}}{=} \mathsf{m}_4$$
$$\begin{aligned} \overline{\mathscr{T}_c}[\![\text{if } x \geqslant 5 \text{ then } x = y + 1 \text{ else } x = y - 1]\!]\mathsf{m}_\top &= (\mathsf{m}_3 \sqcup \mathsf{m}_4) \\ &\stackrel{\text{rep}}{=} \langle \{x - y \leqslant 1, y - x \leqslant -1, -x \leqslant -5\} \ \sqcup \\ &\qquad \{x - y \leqslant -1, y - x \leqslant 1, x \leqslant 4\} \rangle \\ &= \langle \{x - y \leqslant 1, y - x \leqslant 1\} \rangle = \mathsf{m}_5 \end{aligned}$$

After recalling the abstract semantics of imperative languages on octagon domains designed by [43], let us move to database languages.

Defining Abstract Semantics of Database Language in Octagon Domain.

In case of database applications, we consider two different environments: database environment $\rho_d \in \mathfrak{E}_{dbs}$ and application environment $\rho_a \in \mathfrak{E}_a$. To determine abstract semantics of database statements in the domain of octagons, we define the abstract state $\overline{\rho} \in \overline{\Sigma}_{dba}$ as

$$\overline{\rho} = \langle \mathsf{m}_d, \mathsf{m}_a \rangle$$

where m_d and m_a are CDBMs of octagonal constraints as abstraction of database values and application variables values respectively. Therefore, as defined in equation 8, the abstract semantic function for database statements $Q = \langle A, \phi \rangle$ is defined as: $\overline{\mathscr{T}}_{dba}[\![\langle A, \phi \rangle]\!](\mathsf{m}_d, \mathsf{m}_a) = \overline{\mathscr{T}}_{dba}[\![\langle A, \phi \rangle]\!](\mathsf{m}_t\ ,\ \mathsf{m}_a) = (\mathsf{m}_{t'}\ ,\ \mathsf{m}_a)$ where m_t is the octagonal representation of the concrete table t which acts as the target of Q and $\mathsf{m}_{t'}$ is the octagonal representation of the resultant table t'. Below is the abstract semantics for update statement.

$$\begin{aligned} &\overline{\mathscr{T}}_{dep}[\![\langle \mathtt{UPDATE}(\vec{v}_d, \vec{e}), \phi \rangle]\!](\mathsf{m}_t\ ,\ \mathsf{m}_a) \\ &= \overline{\mathscr{T}}_{dep}[\![\langle \mathtt{UPDATE}(\vec{v}_d, \vec{e}) \rangle]\!](\mathsf{m}_{TM}\ ,\ \mathsf{m}_a) \sqcup (\mathsf{m}_{FM}\ ,\ \mathsf{m}_a) \\ &= (\mathsf{m}_{TM'}\ ,\ \mathsf{m}_a) \sqcup (\mathsf{m}_{FM}\ ,\ \mathsf{m}_a) \\ &= (\mathsf{m}_{TM'} \sqcup \mathsf{m}_{FM}\ ,\ \mathsf{m}_a \sqcup \mathsf{m}_a) \\ &= (\mathsf{m}_{t'}\ ,\ \mathsf{m}_a) \end{aligned}$$

where

$$\overline{\mathscr{T}_f}[\![\neg\phi]\!](\mathsf{m}_t, \mathsf{m}_a) = (\mathsf{m}_{FM}, \mathsf{m}_a) \text{ and } \overline{\mathscr{T}_f}[\![\phi]\!](\mathsf{m}_t, \mathsf{m}_a) = (\mathsf{m}_{TM}, \mathsf{m}_a)$$

We can define similarly the abstract semantics for other database statements as well.

Example 15. Consider the concrete table t shown in Table 4(a). Consider the concrete application environment $\rho_a = \langle x \mapsto 100 \rangle$ where x is an application variable. The corresponding abstract representation of ρ_t and ρ_a in octagon domain are represented by CDBM m_t and m_a respectively as

$$\mathsf{m}_t \stackrel{\text{rep}}{=} \Big\{ -eid \leqslant -1,\ eid \leqslant 4,\ \ -sal \leqslant -800,\ sal \leqslant 3000,$$
$$-age \leqslant -28, age \leqslant 62, -dno \leqslant -10, dno \leqslant 20 \Big\}, \text{ and}$$
$$\mathsf{m}_a \stackrel{\text{rep}}{=} \Big\{ x \leqslant 100, -x \leqslant -100 \Big\}.$$

Consider the following `UPDATE` statement

$$Q_{upd}: \ \mathtt{UPDATE}\ t\ \mathtt{SET}\ sal = sal + x\ \mathtt{WHERE}\ age \geqslant 35$$

where $A = \mathtt{UPDATE}(\langle sal \rangle, \langle sal + x \rangle)$ and $\phi = age \geqslant 35$. The abstract semantics w.r.t. $\overline{\rho} = (\mathsf{m}_t, \mathsf{m}_a)$ is

$$\overline{\mathscr{T}}_{dba}[\![\langle A, \phi \rangle]\!](\mathsf{m}_t, \mathsf{m}_a)$$

$$
\begin{aligned}
&= \overline{\mathscr{T}}_{dba}\Big[\!\!\Big[\Big\langle \mathtt{UPDATE}(\langle sal\rangle, \langle sal + x\rangle),\ age \geqslant 35\Big\rangle\Big]\!\!\Big](\mathsf{m}_t, \mathsf{m}_a)\\
&= \overline{\mathscr{T}}_{dba}\Big[\!\!\Big[\Big\langle \mathtt{UPDATE}(\langle sal\rangle, \langle sal + x\rangle)\Big\rangle\Big]\!\!\Big](\mathsf{m}_{TM}, \mathsf{m}_a) \sqcup (\mathsf{m}_{FM}, \mathsf{m}_a)\\
&= (\mathsf{m}_{TM'}, \mathsf{m}_a)\ \sqcup\ (\mathsf{m}_{FM}, \mathsf{m}_a)\\
&= (\mathsf{m}_{TM'} \sqcup \mathsf{m}_{FM}\ ,\ \mathsf{m}_a \sqcup \mathsf{m}_a)\\
&= (\mathsf{m}_{t'}\ ,\ \mathsf{m}_a)
\end{aligned}
$$

where

$$
\mathsf{m}_{TM} \stackrel{\text{rep}}{=} \Big\{-eid \leqslant -1, eid \leqslant 4, -sal \leqslant -800, sal \leqslant 3000, -age \leqslant -35,\ age \leqslant 62,\ -dno \leqslant -10,\ dno \leqslant 20\Big\}
$$

$$
\mathsf{m}_{FM} \stackrel{\text{rep}}{=} \Big\{-eid \leqslant -1, eid \leqslant 4, -sal \leqslant -800, sal \leqslant 3000, -age \leqslant -28,\ age \leqslant 34,\ -dno \leqslant -10, dno \leqslant 20\Big\}
$$

$$
\mathsf{m}_{TM'} \stackrel{\text{rep}}{=} \Big\{-eid \leqslant -1, eid \leqslant 4, -sal \leqslant -900, sal \leqslant 3100, -age \leqslant -35,\ age \leqslant 62,\ -dno \leqslant -10,\ dno \leqslant 20\Big\}
$$

Observation. We can follow an alternative equivalent way of abstract state representation by combining both CDBM of m_d and m_a for the sake of simplicity. Let p and q denote the numbers of database variables and application variables respectively. Given m_d and m_a as CDBM representations of database values and variables values, these can be combined into equivalent CDBM m defined in $(p+q)$ – dimension space by merging m_d and m_a. In the subsequent sections we define abstract semantics w.r.t. abstract state $\overline{\rho} = \mathsf{m}$.

Abstract Semantics towards Independency Computation

Like for the interval domain, the following transition relation is defined, according to equation 9, to compute semantics-based DD-independency in the domain of octagons:

$$
\overline{\mathscr{T}}_{dep} : \mathbb{C} \times \mathbb{M}_\perp \mapsto (\mathbb{M}_\perp \times \mathbb{M}_\perp \times \mathbb{M}_\perp)
$$

Below is the definition of $\overline{\mathscr{T}}_{dep}$ for various database statements in octagon domain.

1. UPDATE:

$$
\overline{\mathscr{T}}_{dep}[\![\langle \mathtt{UPDATE}(\vec{v}_d,\ \vec{e}),\ \phi\rangle]\!]\mathsf{m} = \begin{cases} \Big\langle \mathsf{m}_F,\ \mathsf{m}_T,\ \mathsf{m}_{T'}\Big\rangle & \text{if } \phi \in \Big\{k_i x_i + k_j x_j \leqslant k\Big\} \text{ where } x_i, x_j \in \mathbb{V} \text{ and } k_i, k_j \in [-1, 0, 1] \text{ and } k \in \mathbb{R}\\ \langle \mathsf{m}, \mathsf{m}, \mathsf{m}'\rangle & \text{otherwise} \end{cases}
$$

where $\overline{\mathscr{T}_f}[\![\neg\phi]\!]\mathsf{m} = \mathsf{m}_F$ and $\overline{\mathscr{T}_f}[\![\phi]\!]\mathsf{m} = \mathsf{m}_T$

$$
\overline{\mathscr{T}_c}[\![\mathtt{UPDATE}(\vec{v}_d,\ \vec{e})]\!]\mathsf{m}_T = \mathsf{m}_T\Big[\vec{v}_d \leftarrow \overline{\mathscr{T}_e}[\![\vec{e}]\!]\mathsf{m}_T\Big] = \mathsf{m}_{T'}
$$

$$
\overline{\mathscr{T}_c}[\![\mathtt{UPDATE}(\vec{v}_d, \vec{e})]\!]\mathsf{m} = \mathsf{m}'
$$

2. INSERT: $\overline{\mathscr{T}}_{dep}[\![\Big\langle \mathtt{INSERT}(\vec{v}_d, \vec{e}), false\Big\rangle]\!]\mathsf{m} = \Big\langle \mathsf{m},\ \mathsf{m}_\perp,\ \mathsf{m}_{new}\Big\rangle$

where $\overline{\mathscr{T}_f}[\![\neg false]\!]\mathsf{m} = \mathsf{m}$ and $\overline{\mathscr{T}_f}[\![false]\!]\mathsf{m} = \mathsf{m}_\perp$ and

$$
\overline{\mathscr{T}_c}[\![\mathtt{INSERT}(\vec{v}_d, \vec{e})]\!]\mathsf{m}_\perp = \mathsf{m}_\perp[\vec{v}_d \leftarrow \overline{\mathscr{T}_e}[\![\vec{e}]\!]\mathsf{m}_\perp] = \mathsf{m}_{new}
$$

where $\mathsf{m}_\perp$ represents bottom element that contains an unsatisfiable set of constraints.

3. DELETE:

$$
\overline{\mathscr{T}}_{dep}[\![\langle \mathtt{DELETE}(\vec{v}_d),\ \phi\rangle]\!]\mathsf{m} = \begin{cases} \Big\langle \mathsf{m}_F,\ \mathsf{m}_T,\ \mathsf{m}_\perp\Big\rangle & \text{if } \phi \in \Big\{k_i x_i + k_j x_j \leqslant k\Big\} \text{ where } x_i, x_j \in \mathbb{V} \text{ and } k_i, k_j \in [-1, 0, 1] \text{ and } k \in \mathbb{R}\\ \langle \mathsf{m}, \mathsf{m}, \mathsf{m}_\perp\rangle & \text{otherwise} \end{cases}
$$

4. SELECT:

$$
\overline{\mathscr{T}}_{dep}[\![\Big\langle \mathtt{SELECT}\Big(f(\vec{e'}), r(\vec{h}(\vec{x})), \phi_2, g(\vec{e})\Big), \phi_1\Big\rangle]\!]\mathsf{m} = \begin{cases} \Big\langle \mathsf{m}_F,\ \mathsf{m}_T,\ \mathsf{m}_T\Big\rangle & \text{if } \phi \in \Big\{k_i x_i + k_j x_j \leqslant k\Big\} \text{ where } x_i, x_j \in \mathbb{V} \text{ and } k_i, k_j \in [-1, 0, 1] \text{ and } k \in \mathbb{R}\\ \langle \mathsf{m}, \mathsf{m}, \mathsf{m}\rangle & \text{otherwise} \end{cases}
$$

Observe that $\phi \in \{k_i x_i + k_j x_j \leqslant k\}$ checks whether the condition in `WHERE` clause of database statement respects the form of octagonal constraints.

Example 16. Consider the concrete database table t shown in Table 4(a), and its corresponding abstract representation in the form of CDBM m_t in the domain of octagons as

$$
\mathsf{m}_t \stackrel{\text{rep}}{=} \Big\{-eid \leqslant -1, eid \leqslant 4, -sal \leqslant -800, sal \leqslant 3000, -age \leqslant -28, age \leqslant 62, -dno \leqslant -10, dno \leqslant 20\Big\}.
$$

Consider the following statements:

$$
\begin{aligned}
Q_{upd} &= \mathtt{UPDATE}\ t\ \mathtt{SET}\ sal = sal + x\ \mathtt{WHERE}\ age \geqslant 35\\
Q_{ins} &= \mathtt{INSERT\ INTO}\ t(eid, sal, age, dno)\mathtt{VALUES}(5, 2700, 52, 20)\\
Q_{del} &= \mathtt{DELETE\ \ FROM}\ t\ \mathtt{WHERE}\ age \geqslant 61\\
Q_{sel} &= \mathtt{SELECT}\ sal\ \mathtt{FROM}\ t\ \mathtt{WHERE}\ age \leqslant 50
\end{aligned}
$$

The abstract syntax are

$$
\begin{aligned}
Q_{upd} &= \langle \mathtt{UPDATE}(\langle sal\rangle,\ \langle sal\ +\ 100\rangle),\ age\ \geqslant\ 35\rangle\\
Q_{ins} &= \langle \mathtt{INSERT}(\langle eid,\ sal,\ age,\ dno\rangle,\ \langle 5, 2700, 52, 20\rangle),\ false\rangle\\
Q_{del} &= \langle \mathtt{DELETE}(\langle eid,\ sal,\ age,\ dno\rangle),\ age\ \geqslant\ 61\rangle\\
Q_{sel} &= \langle \mathtt{SELECT}(\langle sal\rangle),\ age \leqslant 50\rangle
\end{aligned}
$$

The abstract semantics of the Q_{upd} with respect to m_t is

$$
\overline{\mathscr{T}}_{dep}[\![\Big\langle \mathtt{UPDATE}(\langle sal\rangle, \langle sal + 100\rangle), age \geqslant 35\Big\rangle]\!]\mathsf{m}_t = \Big\langle \mathsf{m}_F, \mathsf{m}_T, \mathsf{m}_{T'}\Big\rangle
$$

where m_T, m_F and $\mathsf{m}_{T'}$ are depicted in Example 15.

The abstract semantics of the Q_{ins} w.r.t. m_t is

$$
\overline{\mathscr{T}}_{dep}[\![\Big\langle \mathtt{INSERT}(\langle eid, sal, age, dno\rangle, \langle 5, 2700, 52, 20\rangle),\ false\Big\rangle]\!]\mathsf{m}_t = \Big\langle \mathsf{m}_t,\ \mathsf{m}_\perp,\ \mathsf{m}_{new}\Big\rangle \quad \text{where}
$$

$$
\mathsf{m}_{new} \stackrel{\text{rep}}{=} \Big\{-eid \leqslant -5, eid \leqslant 5, -sal \leqslant -2700, sal \leqslant 2700, -age \leqslant -52,\ age \leqslant 52, -dno \leqslant -20,\ dno \leqslant 20\Big\}
$$

The abstract semantics of the Q_{del} w.r.t. m_t is

$$
\overline{\mathscr{T}}_{dep}[\![\Big\langle \mathtt{DELETE}(\langle eid,\ \ sal,\ \ age,\ \ dno\rangle),\ \ age\ \geqslant\ 61\Big\rangle]\!]\mathsf{m}_t = \Big\langle \mathsf{m}_F,\ \mathsf{m}_T,\ \mathsf{m}_\perp\Big\rangle \quad \text{where}
$$

$$
\mathsf{m}_T \stackrel{\text{rep}}{=} \Big\{-eid \leqslant -1, eid \leqslant 4, -sal \leqslant -800, sal \leqslant 3000,\ age \leqslant 62,
$$

$$-age \leqslant -61, -dno \leqslant -10, dno \leqslant 20\Big\}$$

$$\mathsf{m}_F \overset{\text{rep}}{=} \Big\{ -eid \leqslant -1, eid \leqslant 4, -sal \leqslant -800, sal \leqslant 3000,\ age \leqslant 60,$$
$$-age \leqslant -28, -dno \leqslant -10, dno \leqslant 20\Big\}$$

The abstract semantics of the Q_{sel} w.r.t. m_t is

$$\overline{\mathscr{T}}_{dep}[\![\Big\langle \mathtt{SELECT}(\langle age\rangle),\ \ age\ \leqslant\ 50\Big\rangle]\!]\mathsf{m}_t\ =\ \Big\langle \mathsf{m}_F,\ \ \mathsf{m}_T,\ \ \mathsf{m}_T\Big\rangle$$

where

$$\mathsf{m}_T \overset{\text{rep}}{=} \Big\{ -eid \leqslant -1, eid \leqslant 4, -sal \leqslant -800, sal \leqslant 3000,\ age \leqslant 50,$$
$$-age \leqslant -28, -dno \leqslant -10, dno \leqslant 20\Big\}$$

$$\mathsf{m}_F \overset{\text{rep}}{=} \Big\{ -eid \leqslant -1, eid \leqslant 4, -sal \leqslant -800, sal \leqslant 3000,\ age \leqslant 62,$$
$$-age \leqslant -51, -dno \leqslant -10, dno \leqslant 20\Big\}$$

Limitations. The octagon abstract domain is a weakly relational domain that allows a limited number of relations between program variables. Due to this bottleneck, analyses in this abstract domain may fail to produce precise results. For example, consider the following statement: `UPDATE` t `SET` $a = a + 1$ `WHERE` $a + b + c \geqslant 35$. Given an abstract state m in the domain of octagons, the abstract semantics function $\overline{\mathscr{T}}_{dep}$ fails to capture m_{TM} and m_{FM} precisely as the constraint $a+b+c \geqslant 35$ involves more than two variables and hence can not be represented in octagonal constraint form.

6.2.3 *Relational Abstract Domain of Polyhedra*

The preciseness of the analysis in relational abstract domain improves significantly if more number of relations among variables or attributes are in consideration when analyzing the programs. Thus, the analysis in the polyhedra abstract domain, although computationally costly, improves the precision significantly compared to the octagon abstract domain. P. Cousot and N. Halbwachs in their seminal work [41] first introduced the polyhedra abstract domain for static determination of linear equality and inequality relations among program variables, and over the past several decades this has been widely used in several engineering problems such as static analysis of gated Data Dependence Graphs (gated DDGs) [48], Information flow analysis to detect possible information leakages combining symbolic propositional formulas domain and numerical polyhedra domain [49], Hybrid systems verification tool SpaceEx [50], etc.

Let us briefly recall some basics. The regions in n-dimensional space $\mathbb{R}^n$ bounded by finite sets of hyperplanes are called polyhedra. Let $\mathbb{V}_{\mathcal{P}} = \{x_1,\ x_2,\ \dots\ x_n\}$ be the set of variables in program $\mathcal{P}$. We represent by $\vec{v} = \langle v_1,\ v_2,\ \dots\ v_n\rangle \in \mathbb{R}^n$, an n-tuple (vector) of real numbers. By $\beta = \vec{v}.\vec{x} \geqslant k$ where $\vec{v} \neq \vec{0}$, $\vec{x} = \langle x_1, x_2, \dots, x_n\rangle$, $k \in \mathbb{R}$, we represent a linear inequality over $\mathbb{R}^n$. A linear inequality defines an affine half-space of $\mathbb{R}^n$. If P is expressed as the intersection of a finite number of affine half-spaces of $\mathbb{R}^n$, then $\mathsf{P} \in \mathbb{R}^n$ is a convex polyhedron. Formally, a convex polyhedron $\mathsf{P} = (\Theta,\ n)$ is a set of linear inequalities $\Theta = \{\beta_1, \beta_2 \dots \beta_m\}$ on $\mathbb{R}^n$. Equivalently, P can be represented by frame representation which is a collection of generators i.e. *vertices* and *rays* [51]. On the other hand, given a set of linear inequalities Θ on $\mathbb{R}^n$, a set of solutions or points defines a polyhedron $\mathsf{P} = (\Theta, n)$.

Concretization Function. Let $\mathsf{L}_c = \langle \wp(\mathbb{R}^n), \subseteq, \emptyset, \mathbb{R}^n, \cap, \cup\rangle$ be the concrete lattice defined over the concrete domain. The set of polyhedra $\mathbb{P}$ with partial order $\sqsubseteq$ forms an abstract lattice $\mathsf{L}_a = \langle \mathbb{P}, \sqsubseteq, \mathsf{P}_\bot, \mathsf{P}_\top, \sqcap, \sqcup\rangle$. Given $\mathsf{P}_1,\ \mathsf{P}_2 \in \mathbb{P}$, the partial order, meet and join operations are defined below:

- $\mathsf{P}_1 \sqsubseteq \mathsf{P}_2$ if and only if $\gamma(\mathsf{P}_1) \subseteq \gamma(\mathsf{P}_2)$, where $\gamma(\mathsf{P})$ represents the set of solutions or points in P as concrete values.
- $\mathsf{P}_1 \sqcap \mathsf{P}_2$ is the convex polyhedron containing exactly the set of points $\gamma(\mathsf{P}_1) \cap \gamma(\mathsf{P}_2)$.
- $\mathsf{P}_1 \sqcup \mathsf{P}_2$ is not necessarily a convex-polyhedron. Therefore, the least polyhedron enclosing this union is computed in terms of convex hull.

An environment $\rho \in \Sigma \triangleq \mathbb{V} \mapsto \mathbb{R}$ map each variable to its value in $\mathbb{R}$. Given $\mathsf{P} \in \mathbb{P}$, $\gamma_{\mathbb{P}}$ is defined below:

$$\gamma_{\mathbb{P}}(\mathsf{P}) = \begin{cases} \emptyset & \text{if } \mathsf{P} = \mathsf{P}_\bot \\ \mathbb{R}^n & \text{if } \mathsf{P} = \mathsf{P}_\top. \\ \Big\{\rho \in \Sigma \mid \forall(\vec{v}.\vec{x} \geqslant k) : \vec{v}.\rho(\vec{x}) \geqslant k\Big\} & \text{otherwise} \end{cases}$$

Note that there is no abstraction function in polyhedra abstract domain because some vector sets do not have a best over-approximation as a convex closed polyhedron [41]. Therefore, in this case we denote by $\alpha_{\mathbb{P}}(\mathsf{S})$ a (possibly minimal) polyhedron in $\mathbb{P}$ such that $\gamma_{\mathbb{P}}(\alpha_{\mathbb{P}}(\mathsf{S})) \supseteq \mathsf{S}$.

Sound operations in polyhedra domain. Let us recall from [41], [52], [53] some useful operations in the abstract domain of polyhedra:

- *Emptiness test*: Program analyzers during their analysis may encounter constraints present in program statements. Addition of a constraint to a non-empty polyhedron may lead to an empty polyhedron. A polyhedron is empty if and only if its constraint set is infeasible. The Linear Programming (LP) solver [54] is used for checking feasibility of such constraint system. For example, adding a new constraint $\vec{v}.\vec{x} \geqslant k$ to a non empty polyhedra P, we can solve the LP problem $\mu = \min\ \vec{v}.\vec{x}$ subject to P. If $k > \mu$, then new polyhedron is empty. Alternatively, in generator representation a polyhedron is empty if and only if its set of *vertices* and *rays* are empty.
- *Projection*: Let P be a non empty polyhedron. The projection operation removes all constraints information from P corresponding to a variable x_i without affecting the relational information between other variables, defined as:

  $$\Pi_{x_i}(\mathsf{P}) = \{\rho[v/x_i]\ \mid\ \rho \in \gamma(\mathsf{P}),\ v \in \mathbb{R}\}$$

  This is computed by eliminating all occurrences of x_i in the constraints of P by using the Fourier-Motzkin algorithm [55] as below:

  $$\begin{aligned} F(\mathsf{P}, x_i) \triangleq\ & \{(\Sigma_i v_i x_i \geqslant k) \in \Theta^i \mid v_i = 0\} \cup \\ & \{(-v_i^-)\beta^+ + v_i^+\beta^- \mid \beta^+ = (\Sigma_i v_i^+ x_i \geqslant k^+) \in \Theta^+,\ v_i^+ > 0, \\ & \qquad \beta^- = (\Sigma_i v_i^- x_i \geqslant k^-) \in \Theta^-,\ v_i^- < 0\} \end{aligned}$$

  where v_i^+ and v_i^- represent positive and negative coefficients for x_i respectively. The algorithm partitions the set of liner inequalities $\Theta = \{\beta_1, \beta_2 \dots \beta_m\}$ into Θ^+, Θ^i and Θ^-, corresponding to inequalities that have positive, zero and negative coefficients for x_i. For each pair (β^+, β^-) of inequalities drawn from $\Theta^+ \times \Theta^-$, the algorithm multiplies β^+ by the absolute value of x_i-*th* coefficient ($|v_i^-|$) in β^- and similarly multiplies β^- by x_i-*th* coefficient v_i^+ in β^+. The combination of these two results finally removes x_i as the resultant coefficient becomes zero.

- *Inclusion test*: Let P_1 and P_2 be non empty polyhedra. The inclusion test (denoted $\mathsf{P}_2 \sqsubseteq \mathsf{P}_1$) reduces to the problem of checking whether each inequality in P_2 is entailed by P_1, which can be implemented using LP. For example, we can compute $\mu = \min \vec{v}.\vec{x}$ subject to P_1 for each $\vec{v}.\vec{x} \geqslant k$. If $\mu < k$ then inclusion does not hold.

Abstract Semantics of Imperative Language in Polyhedra – A Quick Tour [41].

Given the concrete denotational semantics functions $\mathscr{T}_f : \big(\mathbb{B} \mapsto \wp(\Sigma)\big) \mapsto \wp(\Sigma)$, the corresponding sound abstract function $\overline{\mathscr{T}_f}$ in the domain of polyhedra is defined as $\overline{\mathscr{T}_f} : (\mathbb{B} \mapsto \mathbb{P}) \mapsto \mathbb{P}$. Similarly, given the concrete denotational semantic function $\mathscr{T}_c : \big(\mathbb{C} \mapsto \wp(\Sigma)\big) \mapsto \wp(\Sigma)$, its corresponding sound abstract function $\overline{\mathscr{T}_c}$ in polyhedra domain is defined as $\overline{\mathscr{T}_c} : (\mathbb{C} \mapsto \mathbb{P}) \mapsto \mathbb{P}$.

Test: Let b be a boolean expression in the form of linear inequalities $\vec{v}.\vec{x} \geqslant k$ and the abstract state in the form of polyhedron P. The state-filtering function $\overline{\mathscr{T}_f}$ finds P' applying b on P define as

$$\overline{\mathscr{T}_f}[\![\vec{v}.\vec{x} \geqslant k]\!]\mathsf{P} = \mathsf{P}'$$

where $\mathsf{P}' = \mathsf{P} \sqcap b$.

Example 17. Given P=($\{x \geqslant 8,\ y \geqslant 6\}$, 2). The equivalent generators representation (*vertices* and *rays*) of P is V={(8, 6)} and R={(1, 0), (0, 1)}. The abstract semantics of boolean expression $x \geqslant 20$ is defined as: $\overline{\mathscr{T}_f}[\![x \geqslant 20]\!]\mathsf{P} = \mathsf{P}'$ where $\mathsf{P}' = (\{x \geqslant 20, y \geqslant 6\}, 2)$ and its equivalent generators representation is V'={(20, 6)} and R'={(1, 0), (0, 1)}.

Assignment statement: $\overline{\mathscr{T}_c}[\![x_j = e]\!](\mathsf{P}) = \mathsf{P}'$ where P' is obtained as follows: (*i*) **Case**-1: If e is non-linear expression or the assignment is non-invertible, then we simply project-out the corresponding variable from the linear inequalities in P, resulting into a new polyhedron P'; (*ii*) **Case**-2: otherwise, we introduce a fresh variable $x_j{}'$ to hold the value of e, then we project out x_j and finally we reuse $x_j{}'$ in place of x_j which results into P'.

Example 18. Given P=($\{x \geqslant 3,\ y \geqslant 2\}$, 2). The equivalent generators representation (*vertices* and *rays*) of P is V={(3, 2)} and R={(1, 0), (0, 1)}. The $\overline{\mathscr{T}_c}$ of assignment $x = x + y$ is define as

$$\overline{\mathscr{T}_c}[\![x = x + y]\!](\{x \geqslant 3, y \geqslant 2\}, 2) = \mathsf{P}' \qquad \text{where}$$

$\mathsf{P}' = (\{x - y \geqslant 3, y \geqslant 2\}, 2)$ and its equivalent generators representation is V'={(5, 2)} and R'={(1, 0), (-1, -1)}.

Defining Abstract Semantics of Database Language in Polyhedra Domain.

Let us define the abstract semantics for four database operations in the domain of polyhedra. Like octagon domain, given $\overline{\rho} = \langle \mathsf{P}_d, \mathsf{P}_a \rangle \in \overline{\Sigma}_{dba}$ where P_d and P_a are polyhedra representation of database values and application variables values respectively. According to equation 8, the abstract semantic function for database statements $Q = \langle A, \phi \rangle$ is defined as: $\overline{\mathscr{T}}_{dba}[\![\langle A, \phi \rangle]\!](\mathsf{P}_d, \mathsf{P}_a) = \overline{\mathscr{T}}_{dba}[\![\langle A, \phi \rangle]\!](\mathsf{P}_t\ ,\ \mathsf{P}_a) = (\mathsf{P}_{t'}\ ,\ \mathsf{P}_a)$ where P_t is the polyhedron representation of the concrete table t which acts as the target of Q, and $\mathsf{P}_{t'}$ is the polyhedron representation of the resultant table t'.

Alternatively, like octagon abstract domain, for the sake of simplicity, we may combine both P_d and P_a into a single polyhedra P as an abstract program state. In the subsequent sections we define abstract semantics suitable for independency computations w.r.t. an abstract state $\overline{\rho}$= P in the domain of polyhedra.

Abstract Semantics towards Independency Computation.

Let us define the transition relation $\overline{\mathscr{T}}_{dep} : \mathbb{C} \times \mathbb{P} \mapsto (\mathbb{P} \times \mathbb{P} \times \mathbb{P})$ to compute semantics-based DD-independency in the domain of polyhedra for database statements:

1. UPDATE: $\overline{\mathscr{T}}_{dep}[\![\langle \mathsf{UPDATE}(\vec{v}_d, \vec{e}),\ \phi \rangle]\!]\mathsf{P} = \Big\langle \mathsf{P}_F, \mathsf{P}_T, \mathsf{P}_{T'} \Big\rangle$ where

$$\overline{\mathscr{T}_f}[\![\neg\phi]\!]\mathsf{P} = \mathsf{P}_F.$$
$$\overline{\mathscr{T}_f}[\![\phi]\!]\mathsf{P} = \mathsf{P}_T$$
$$\overline{\mathscr{T}_c}[\![\mathsf{UPDATE}(\vec{v}_d, \vec{e})]\!]\mathsf{P}_T = \overline{\mathscr{T}_c}[\![\vec{v}_d = \vec{e}]\!]\mathsf{P}_T = \mathsf{P}_{T'}$$

We denote by the notation $\vec{v}_d$= $\vec{e}$ a series of assignments $\langle v_1$= e_1, v_2= e_2, ..., v_n= $e_n\rangle$ where $\vec{v}_d = \langle v_1, v_2, \dots, v_n\rangle$ and $\vec{e}$=$\langle e_1, e_2, \dots, e_n\rangle$, which follow the transition semantic definition for the assignment statement.

2. INSERT: $\overline{\mathscr{T}}_{dep}[\![\langle \mathsf{INSERT}(\vec{v}_d, \vec{e}), false \rangle]\!]\mathsf{P} = \Big\langle \mathsf{P}, \mathsf{P}_\perp, \mathsf{P}_{new} \Big\rangle$ where

$$\overline{\mathscr{T}_f}[\![\neg false]\!]\mathsf{P} = \mathsf{P}$$
$$\overline{\mathscr{T}_f}[\![false]\!]\mathsf{P} = \mathsf{P}_\perp$$
$$\overline{\mathscr{T}_c}[\![\mathsf{INSERT}(\vec{v}_d, \vec{e})]\!]\mathsf{P}_\perp = \mathsf{P}_\perp\Big[\vec{v}_d \leftarrow \overline{\mathscr{T}_e}[\![\vec{e}]\!]\mathsf{P}_\perp\Big] = \mathsf{P}_{new}$$

P_{new} represents a polyhedron corresponding to the new inserted tuple values.

3. DELETE: $\overline{\mathscr{T}}_{dep}[\![\langle \mathsf{DELETE}(\vec{v}_d),\ \phi \rangle]\!]\mathsf{P} = \Big\langle \mathsf{P}_F,\ \mathsf{P}_T,\ \mathsf{P}_\perp \Big\rangle$

4. SELECT:

$$\overline{\mathscr{T}}_{dep}[\![\Big\langle \mathsf{SELECT}\Big(f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2,\ g(\vec{e})\Big),\ \phi_1 \Big\rangle]\!]\mathsf{P} = \Big\langle \mathsf{P}_F, \mathsf{P}_T, \mathsf{P}_T \Big\rangle$$

Example 19. Consider the database table t in Table 4(a) and its corresponding abstract representation $\mathsf{P}_t = (\Theta, n)$ in the form of polyhedron, where

$$\mathsf{P}_t = \Big\{eid \geqslant 1, -eid \geqslant -4, sal \geqslant 800, -sal \geqslant -3000,$$
$$age \geqslant 28, -age \geqslant -62, dno \geqslant 10, -dno \geqslant -20\Big\}$$

Consider the following statements:

$$Q_{upd} = \texttt{UPDATE}\ t\ \texttt{SET}\ sal = sal + sal \times 0.2\ \texttt{WHERE}\ dno + age \geqslant 60$$
$$Q_{ins} = \texttt{INSERT INTO}\ t\ (eid, sal, age, dno)\texttt{VALUES}(5, 2700, 52, 20)$$
$$Q_{del} = \texttt{DELETE FROM}\ t\ \texttt{WHERE}\ age \geqslant 61$$
$$Q_{sel} = \texttt{SELECT}\ age\ \texttt{FROM}\ t\ \texttt{WHERE}\ age + dno \leqslant 60$$

The equivalent abstract syntax are:

$$Q_{upd} = \langle \texttt{UPDATE}(\langle sal\rangle,\ \langle sal + sal\ \times\ 0.2\rangle),\ \ dno + age\ \geqslant\ 60\rangle$$
$$Q_{ins} = \langle \texttt{INSERT}(\langle eid,\ sal,\ age,\ dno\rangle, \langle 5, 2700, 52, 20\rangle),\ \ false\rangle$$
$$Q_{del} = \langle \texttt{DELETE}(\langle eid,\ sal,\ age,\ dno\rangle),\ age\ \geqslant\ 61\rangle$$
$$Q_{sel} = \langle \texttt{SELECT}(\langle age\rangle),\ age + dno\ \leqslant\ 60\rangle$$

The abstract semantics of Q_{upd} w.r.t. P_t is

$$\overline{\mathscr{T}}_{dep}[\![\Big\langle \texttt{UPDATE}(\langle sal\rangle,\ \langle sal + sal \times 0.2\rangle),\ \ dno + age \geqslant 60\Big\rangle]\!]\mathsf{P}_t = \Big\langle \mathsf{P}_F,\ \mathsf{P}_T,\ \mathsf{P}_{T'}\Big\rangle$$

where

$$\mathsf{P}_T = \Big\{eid \geqslant 1, -eid \geqslant -4, sal \geqslant 800, -sal \geqslant -3000, age \geqslant 40, -age \geqslant -62, dno \geqslant 10, -dno \geqslant -20, dno + age \geqslant 60\Big\}$$

$$\mathsf{P}_F = \Big\{eid \geqslant 1, -eid \geqslant -4, sal \geqslant 800, -sal \geqslant -3000, age \geqslant 28, -age \geqslant -49, dno \geqslant 10, -dno \geqslant -20, -dno - age \geqslant -59\Big\}$$

$$\mathsf{P}_{T'} = \Big\{eid \geqslant 1, -eid \geqslant -4, sal \geqslant 960, -sal \geqslant -3600, age \geqslant 40, -age \geqslant -62, dno \geqslant 10, -dno \geqslant -20, dno + age \geqslant 60\Big\}$$

The abstract semantics of Q_{ins} w.r.t. P_t is

$$\overline{\mathscr{T}}_{dep}[\![\Big\langle \texttt{INSERT}(\langle eid, sal, age, dno\rangle, \langle 5, 2700, 52, 20\rangle),\ \ false\Big\rangle]\!]\mathsf{P}_t = \Big\langle \mathsf{P}_t,\ \mathsf{P}_\perp,\ \mathsf{P}_{new}\Big\rangle$$

where

$$\mathsf{P}_{new} = \Big\{eid \geqslant 5, -eid \geqslant -5, sal \geqslant 2700, -sal \geqslant -2700, age \geqslant 52, -age \geqslant -52, dno \geqslant 20, -dno \geqslant -20\Big\}$$

The abstract semantics of Q_{del} w.r.t. P_t is

$$\overline{\mathscr{T}}_{dep}[\![\Big\langle \texttt{DELETE}(\langle eid,\ sal,\ age,\ dno\rangle),\ age\ \geqslant\ 61\Big\rangle]\!]\mathsf{P}_t = \Big\langle \mathsf{P}_F,\ \mathsf{P}_T,\ \mathsf{P}_\perp\Big\rangle$$

where

$$\mathsf{P}_T = \Big\{eid \geqslant 1, -eid \geqslant -4, sal \geqslant 800, -sal \geqslant -3000, age \geqslant 61, -age \geqslant -62, dno \geqslant 10, -dno \geqslant -20\Big\}$$

$$\mathsf{P}_F = \Big\{eid \geqslant 1, -eid \geqslant -4, sal \geqslant 800, -sal \geqslant -3000, age \geqslant 28, -age \geqslant -60, dno \geqslant 10, -dno \geqslant -20\Big\}$$

The abstract semantics of Q_{sel} w.r.t. P_t is

$$\overline{\mathscr{T}}_{dep}[\![\Big\langle \texttt{SELECT}(\langle age\rangle),\ age + dno \leqslant 60\Big\rangle]\!]\mathsf{P}_t = \Big\langle \mathsf{P}_F,\ \mathsf{P}_T,\ \mathsf{P}_T\Big\rangle$$

where

$$\mathsf{P}_T = \Big\{eid \geqslant 1, -eid \geqslant -4, sal \geqslant 800, -sal \geqslant -3000, age \geqslant 28, -age \geqslant -50, dno \geqslant 10, -dno \geqslant -20, -dno - age \geqslant -60\Big\}$$

$$\mathsf{P}_F = \Big\{eid \geqslant 1, -eid \geqslant -4, sal \geqslant 960, -sal \geqslant -3600, age \geqslant 51, -age \geqslant -62, dno \geqslant 10, -dno \geqslant -20, dno + age \geqslant 61\Big\}$$

6.2.4 *Powerset Abstract Domain*

The finite powerset construction of an abstract domain yields a new abstract domain which improves the precision of the analysis as compared to the original one [56]. For example, application of condition-part in many cases may result in multiple abstract values for an attribute. In such cases the powerset representation of abstract state is more suitable in terms of precision. Due to the scattered nature of data in the database, the semantics-based dependency analysis of database applications in the above-mentioned abstract domains may often be highly over-approximated. Thus powerset abstract domains, on top of the existing relational- and non-relational abstract domains, may capture the database values as a way of refined approximation, improving the analysis results significantly.

Let $\mathsf{L}_c = \langle \mathbb{D}, \leqslant, \bot_c, \top_c, \cap_c, \cup_c\rangle$ be a concrete lattice and $\mathsf{L}_a = \langle \overline{\mathbb{D}}, \sqsubseteq, \bot_a, \top_a, \sqcap_a, \sqcup_a\rangle$ an abstract lattice over an abstract domain $\mathbb{A}$. The L_c and L_a are related by the Galois connection $(\mathsf{L}_c, \alpha, \gamma, \mathsf{L}_a)$. Considering the powerset abstract domain, the powerset of $\overline{\mathbb{D}}$ denoted by $\wp(\overline{\mathbb{D}})$ with the order relations $\preceq$ forms an abstract lattice $\mathsf{L}_p = \langle \wp(\overline{\mathbb{D}}), \preceq, \emptyset, \overline{\mathbb{D}}, \wedge, \vee\rangle$. The partial order, meet and join operations in this abstract domain are defined as follows:

- $\forall S_1, S_2 \in \wp(\overline{\mathbb{D}}) : S_1 \preceq S_2 \Leftrightarrow \forall \overline{v}_i \in S_1 : \exists \overline{v}_j \in S_2.\ \overline{v}_i \sqsubseteq \overline{v}_j$.
- $\forall S_1, S_2 \in \wp(\overline{\mathbb{D}}) : S_1 \wedge S_2 = \{\overline{v}_i \sqcap \overline{v}_j \mid \forall \overline{v}_i \in S_1, \forall \overline{v}_j \in S_2\}$.
- $\forall S_1, S_2 \in \wp(\overline{\mathbb{D}}) : S_1 \vee S_2 = S_1 \cup S_2$.

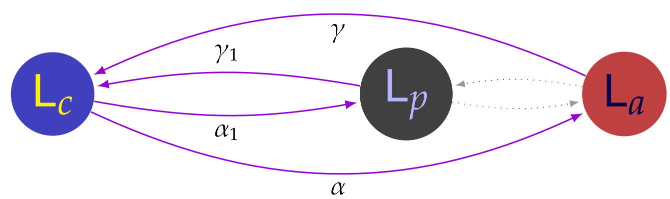
Observe that in powerset abstract domain the meet operation $S_1 \wedge S_2$ is defined by the pairwise meet of the elements from S_1 and S_2, whereas the join operation $S_1 \vee S_2$ reduces to a set union.

The L_c and L_p are related by a Galois connections $\langle \mathsf{L}_c, \alpha_1, \gamma_1, \mathsf{L}_p\rangle$ where α_1 and γ_1 on $\forall X \in \mathbb{D}$ and $\forall Y \in \wp(\overline{\mathbb{D}})$ are defined below:

$$\alpha_1(X) = \begin{cases} \emptyset & \text{if } X = \bot_c \\ \overline{\mathbb{D}} & \text{if } X = \top_c \\ \big\{\alpha(X)\big\} & \text{otherwise} \end{cases}$$

$$\gamma_1(Y) = \begin{cases} \bot_c & \text{if } Y = \emptyset \\ \top_c & \text{if } Y = \overline{\mathbb{D}}. \\ \bigcup\big\{\gamma(\overline{v}) \mid \overline{v} \in Y\big\} & \text{otherwise} \end{cases}$$

The pictorial representation of the Galois Connection among the concrete domain (L_c), the abstract domain (L_a), and the powerset of the abstract domain (L_p) is shown below:



Let us explain the powerset construction over the interval abstract domain. Consider the interval abstract domain $\overline{\mathbb{I}} = \{[l,\ h] \mid l \in \mathbb{Z} \cup \{-\infty\}, h \in \mathbb{Z} \cup \{+\infty\}, l \leq h\} \cup \bot$ forming an abstract lattice $L_a = \langle \overline{\mathbb{I}}, \sqsubseteq, \perp, [-\infty, +\infty], \sqcap, \sqcup \rangle$. The powerset of the intervals denoted by $\wp(\overline{\mathbb{I}})$ forms the abstract lattice $L_p = \langle \wp(\overline{\mathbb{I}}), \preceq, \emptyset, \overline{\mathbb{I}}, \curlywedge, \curlyvee \rangle$.

The correspondence between L_a and L_p is formalized as the Galois connections $\langle L_a, \alpha_1, \gamma_1, L_p \rangle$. The partial order, meet and join operations in the powerset domain of intervals can be defined accordingly.

Given a powerset abstract domain of intervals $\wp(\overline{\mathbb{I}})$, the set of abstract states $\overline{\Sigma} : \mathbb{V} \mapsto \wp(\overline{\mathbb{I}})$ respects the Galois Connection, i.e. $\forall \rho \in \Sigma, \forall \overline{\rho} \in \overline{\Sigma}$: $\alpha_1(\rho) \sqsubseteq \overline{\rho} \iff \rho \subseteq \gamma_1(\overline{\rho})$.

Given $S_1, S_2 \in \wp(\overline{\mathbb{I}})$, the sound abstract arithmetic operations $\overline{\oplus}$ in the powerset abstract domain of intervals are defined as:

$$\forall S_1, S_2 \in \wp(\overline{\mathbb{I}}) : S_1 \overline{\oplus} S_2 = \{\overline{v}_i \mathbin{\overline{\oplus}} \overline{v}_j \mid \forall \overline{v}_i \in S_1, \forall \overline{v}_j \in S_2\}$$

The corresponding sound abstract semantics function $\overline{\mathscr{T}_e} : (\mathbb{E} \cup \mathbb{B}) \mapsto (\overline{\Sigma} \mapsto \wp(\overline{\mathbb{I}}) \cup \{true, false, \top_B\})$ for expression evaluation where $\top_B$ denotes " *may be true or may be false*", the abstract semantics functions $\overline{\mathscr{T}_f} : \mathbb{B} \mapsto (\overline{\Sigma} \mapsto \overline{\Sigma})$ for abstract state-filtering and $\overline{\mathscr{T}_c} : \mathbb{C} \mapsto (\overline{\Sigma} \mapsto \overline{\Sigma})$ for commands are defined accordingly in the powerset domain of intervals. Example 20 illustrates this.

Example 20. Consider the statement $c ::=$ if $x \geqslant 5$ then $x = x + y$ else $x = x - y$. Consider an abstract state in the powerset domain of interval $\overline{\rho} = \langle x \mapsto \{[2,6],[8,10]\}, y \mapsto \{[1,1]\} \rangle$. The abstract semantics of c w.r.t. $\overline{\rho}$ is illustrated below:

$$\begin{aligned}
\overline{\mathscr{T}_f}[\![x \geqslant 5]\!](\overline{\rho}) &= \overline{\rho}[x \leftarrow \{[5,6],[8,10]\}] = \overline{\rho}_1 \\
\overline{\mathscr{T}_f}[\![\neg(x \geqslant 5)]\!](\overline{\rho}) &= \overline{\rho}[x \leftarrow \{[2,4]\}] = \overline{\rho}_2 \\
\overline{\mathscr{T}_e}[\![x + y]\!](\overline{\rho}_1) &= \overline{\rho}_1(x) \mathbin{\overline{+}} \overline{\rho}_1(y) = \{[6,7],[9,11]\} \\
\overline{\mathscr{T}_e}[\![x - y]\!](\overline{\rho}_2) &= \overline{\rho}_2(x) \mathbin{\overline{-}} \overline{\rho}_2(y) = \{[1,3]\} \\
\overline{\mathscr{T}_c}[\![x = x + y]\!](\overline{\rho}_1) &= \overline{\rho}_1[x \leftarrow \{[6,7],[9,11]\}] = \overline{\rho}_3 \\
\overline{\mathscr{T}_c}[\![x = x - y]\!](\overline{\rho}_2) &= \overline{\rho}_2[x \leftarrow \{[1,3]\}] = \overline{\rho}_4 \\
\overline{\mathscr{T}_c}[\![\text{if } x \geqslant 5 \text{ then } x = x + y \text{ else } x = x - y]\!](\overline{\rho}) & \\
= (\overline{\rho}_3 \curlyvee \overline{\rho}_4) &= \langle \{[6,7],[9,11]\} \curlyvee \{[1,3]\},\ \{[1,1]\} \curlyvee \{[1,1]\} \rangle \\
&= \langle \{[1,3],[6,7],[9,11]\},\ \{[1,1]\} \rangle = \overline{\rho}_5
\end{aligned}$$

Defining Abstract Semantics of Database Language in the Powerset of Interval

Given the semantic domain $\wp(\overline{\mathbb{I}})$, the abstract state is defined as $\overline{\rho} = (\rho_{\overline{t}}, \rho_{\overline{a}})$ where $\rho_{\overline{t}} : \mathtt{attr}(\overline{t}) \to \wp(\overline{\mathbb{I}})$ and $\rho_{\overline{a}} : \mathbb{V}_a \to \wp(\overline{\mathbb{I}})$.

Like other domains, according to equation 9, the abstract semantics in the powerset abstract domain for database statements are similarly defined below:

`UPDATE`:

$$\overline{\mathscr{T}}_{dep}[\![\left\langle \mathtt{UPDATE}(\vec{v}_d, \vec{e}),\ \phi \right\rangle]\!](\rho_{\overline{t}}, \rho_{\overline{a}}) = \left\langle \rho_{\overline{FM}}, \rho_{\overline{TM}}, \rho_{\overline{TM'}} \right\rangle$$

$$\begin{aligned}
\text{where } & \overline{\mathscr{T}_f}[\![\neg\phi]\!](\rho_{\overline{t}}, \rho_{\overline{a}}) = (\rho_{\overline{FM}}, \rho_{\overline{a}}) \\
& \overline{\mathscr{T}_f}[\![\phi]\!](\rho_{\overline{t}}, \rho_{\overline{a}}) = (\rho_{\overline{TM}}, \rho_{\overline{a}}) \\
& \overline{\mathscr{T}_c}[\![\mathtt{UPDATE}(\vec{v}_d, \vec{e})]\!](\rho_{\overline{TM}}, \rho_{\overline{a}}) \\
& = (\rho_{\overline{TM}}[\vec{v}_d \leftarrow \overline{\mathscr{T}_e}[\![\vec{e}]\!](\rho_{\overline{TM}}, \rho_{\overline{a}})], \rho_{\overline{a}}) = (\rho_{\overline{TM'}}, \rho_{\overline{a}})
\end{aligned}$$

`INSERT`: $\overline{\mathscr{T}}_{dep}[\![\left\langle \mathtt{INSERT}(\vec{v}_d, \vec{e}),\ false \right\rangle]\!](\rho_{\overline{t}}, \rho_{\overline{a}}) = \left\langle \rho_{\overline{t}}, \rho_{\perp}, \rho_{\overline{new}} \right\rangle$

$$\begin{aligned}
\text{where } & \overline{\mathscr{T}_f}[\![\neg false]\!](\rho_{\overline{t}}, \rho_{\overline{a}}) = (\rho_{\overline{t}}, \rho_{\overline{a}}) \\
& \overline{\mathscr{T}_f}[\![false]\!](\rho_{\overline{t}}, \rho_{\overline{a}}) = (\rho_{\perp}, \rho_{\overline{a}}) \text{ where } \rho_{\perp} : \mathtt{attr}(t) \to \emptyset \\
& \overline{\mathscr{T}_c}[\![\mathtt{INSERT}(\vec{v}_d, \vec{e})]\!](\rho_{\perp}, \rho_{\overline{a}}) \\
& = (\rho_{\perp}[\vec{v}_d \leftarrow \overline{\mathscr{T}_e}[\![\vec{e}]\!](\rho_{\perp}, \rho_{\overline{a}})], \rho_{\overline{a}}) = (\rho_{\overline{new}}, \rho_{\overline{a}})
\end{aligned}$$

`DELETE`: $\overline{\mathscr{T}}_{dep}[\![\left\langle \mathtt{DELETE}(\vec{v}_d),\ \phi \right\rangle]\!](\rho_{\overline{t}}, \rho_{\overline{a}}) = \left\langle \rho_{\overline{FM}}, \rho_{\overline{TM}}, \rho_{\perp} \right\rangle$

SELECT:

$$\begin{aligned}
& \overline{\mathscr{T}}_{dep}[\![\left\langle \mathtt{SELECT}\left(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2,\ g(\vec{e})\right),\ \phi_1 \right\rangle]\!](\rho_{\overline{t}}, \rho_{\overline{a}}) \\
& = \left\langle \rho_{\overline{FM}}, \rho_{\overline{TM}}, \rho_{\overline{TM}} \right\rangle
\end{aligned}$$

6.3 A Summary on Abstract Domains

Let us summarize the strengths and limitations of various relational and non-relational abstract domains which we have used above to define abstract semantics of database applications. As abstraction in the interval domain does not capture any relation among variables or attributes, this yields a highly approximated analysis-results. On the other hand, although abstract semantics in both octagon and polyhedra domains capture relationships among variables or attributes, the octagon domain allows a weak form of constraints compared to that in polyhedra domain. Due to this reason, analysis in octagon domain is less precise than that in the polyhedra domain. Intuitively, preciseness of the analysis in relational abstract domain improves significantly when more number of relations among variables or attributes is present in the program itself, e.g. in the `WHERE` clause or in the conditional or iterative statements. In terms of algorithmic efficiency, octagon domain always lies between interval and polyhedra. Analyses (involving all common operations e.g. emptiness test, inclusion, etc.) in polyhedra abstract domain experience an exponential ($O(2^n)$) worst-case time complexity [54], whereas in octagonal domain the graph-based analysis algorithms for all common operations experience $O(n^3)$ worst-case time complexity, where n is the number of variables in the program [43]. Powerset operator, on the other hand, can generate very expressive interpretations. In fact, the powerset abstract domain gains the capability of expressing the logical disjunction of the properties represented by the original domain. A summary on the strength and weakness of domains is reported in Table 8.

6.4 Algorithm to Compute Abstract Semantics of Database Applications

We now design the algorithm **CompAbsSem**, depicted in Algorithm 1, which makes use of the semantics function $\overline{\mathscr{T}}_{dba}$ and computes abstract states w.r.t. abstract domain $\overline{D}$ at each program point of the database program. The algorithm is based on the data flow analysis considering various control-flow nodes: *start, DB-connect, assignment, test, update, delete, insert, select, join, end*. We denote by $pred(c_i)$ and $AS(c_i)$ the set of predecessor of c_i and the abstract state at c_i respectively. The algorithm starts in step 2 with undefined abstract state at each program point and then applies in step 3 all the data-flow equations (defined in steps 4-25)

Domain	Invariants	Time cost	Memory cost	Precision
Interval	$x \in \{[l,\ h] \mid l,\ h \in \mathbb{R}, l \leq h, x \in \mathbb{V}\}$	$O(n)$	$O(n)$	*low*
Octagon	$\pm x_i \pm x_j \leqslant k$, $x_i, x_j \in \mathbb{V} \wedge k \in \mathbb{R} \cup \{\infty\}$	$O(n^3)$	$O(n^2)$	*medium*
Polyhedra	$\Sigma_{i=1}^{n} a_i x_i \geqslant k$, $x_i \in \mathbb{V} \wedge a_i, k \in \mathbb{R}^n$	$O(2^n)$	$O(2^n)$	*high*
Powerset	$\wp(\overline{\mathbb{D}})$	*Depends on* $\overline{\mathbb{D}}$	*Depends on* $\overline{\mathbb{D}}$	*Improves w.r.t.* $\overline{\mathbb{D}}$

TABLE 8: A summary on various abstract domains

until least fixed point solution is reached. After obtaining the abstract state at each program point in the form of collecting semantics, step 26 applies $\overline{\mathscr{T}}_{dep}$ in order to get state-representation in the form of three-tuples $\langle \rho_{\overline{o}}, \rho_{\overline{\square}}, \rho_{\overline{\blacksquare}} \rangle$ (as defined in equation 9). This abstract semantics is used to compute *used*- and *defined*-parts and hence the semantics-based independencies (described next). Observe that if the initial database is unknown then the domain range of each attribute and other integrity constraints are considered to represent the initial abstraction of database as an overapproximation of all possible initial database states, as defined in steps 1 and 9.

6.5 Approximating *used*- and *defined* Database Parts in Various Abstract Domains

Given a database statement Q, let $\overline{\rho} = \langle \rho_{\overline{o}}, \rho_{\overline{\square}}, \rho_{\overline{\blacksquare}} \rangle$ be an abstract state at Q obtained by following Algorithm 1. In order to determine abstract DD-dependency between two database statements, we need to identify abstract database-parts to be *defined* or *used* by Q. To this aim, let us define sound abstract functions $\overline{\mathbf{A}}_{def}$ and $\overline{\mathbf{A}}_{use}$ w.r.t. their concrete counterparts already defined in equations 6 and 7 respectively. Suppose D^{Q} and U^{Q} denote the *defined* and the *used* abstract database-parts by Q respectively. Therefore,

$$\mathsf{D}^{Q} = \overline{\mathbf{A}}_{def}(Q, \overline{\rho}) = \langle \rho_{\overline{\square}}, \rho_{\overline{\blacksquare}} \rangle \tag{12}$$

$$\mathsf{U}^{Q} = \overline{\mathbf{A}}_{use}(Q, \overline{\rho}) = \langle \rho_{\overline{\square}} \rangle \tag{13}$$

Observe that $\overline{\mathbf{A}}_{use}$ maps a query Q to the abstract database-part used by it, whereas $\overline{\mathbf{A}}_{def}$ defines the changes occurred in the abstract database states after performing the action in Q. We represent D^{Q} in the form of two-tuple where $\rho_{\overline{\square}}$ and $\rho_{\overline{\blacksquare}}$ respectively represent the true-part before and the updated-part after executing Q on the abstract database. Note that although the *defined*-part can be computed by following the abstract difference operation $\overline{\Delta}$ (corresponding to Δ defined in equation 6), however to avoid computational complexity in dependency computation, we keep both of these separated. Table 9 depicts *defined* and *used* parts by different database statements in various abstract domains.

6.6 Dependency Computations

We are now in a position to compute DD-independencies among database statements based on the information on *used*- and *defined*-parts as obtained in the previous section.

Algorithm 1: CompAbsSem

Input: Database program $\mathcal{P}$ containing n statements, Initial database dB, and Abstract domain $\overline{D}$.
Output: Abstract state at each program point of $\mathcal{P}$

1 Compute $\overline{\rho} = \langle \rho_{\overline{dB}}, \rho_{\overline{a}} \rangle$ using the abstraction function α following the Galois connection $\Big\langle (\wp(D), \subseteq), \alpha, \gamma, (\overline{D}, \sqsubseteq) \Big\rangle$ as formalized in Definition 7.
2 $\forall i \in [1, \ldots, n]$, $AS(c_i) = \perp$. // Initializing $AS(c_i)$ as initial abstract collecting semantics.
3 Apply data flow equations defined in steps 4 - 25 until least fix-point is reached.
4 **for** i =1 *to* n **do**
  // Defining data flow equation for CFG-node corresponding to a statement c_i in $\mathcal{P}$.
5   **switch** (c_i)
6   **case** *start:*
7       $AS(c_i)$=$\perp$
8   **case** *DB-connect:*
9       $AS(c_i)$=$\langle \rho_{\overline{dB}}, \rho_{\overline{a}} \rangle$
10   **case** *assignment:*
11       $AS(c_i) = \bigsqcup_{c_j \in pred(c_i)} \Big\{ \overline{\mathscr{T}}_{dba} [\![x = e]\!](\overline{\rho}) \mid \overline{\rho} \in AS(c_j) \Big\}$
12   **case** *test:*
13       $AS(c_i) = \bigsqcup_{c_j \in pred(c_i)} \Big\{ \overline{\mathscr{T}}_{dba} [\![b]\!](\overline{\rho}) \mid \overline{\rho} \in AS(c_j) \Big\}$
14   **case** *update:*
15       $AS(c_i) = \bigsqcup_{c_j \in pred(c_i)} \Big\{ \overline{\mathscr{T}}_{dba} [\![\mathtt{UPDATE}(\vec{v}_d,\ \vec{e}),\ \phi \rangle]\!](\overline{\rho}) \mid \overline{\rho} \in AS(c_j) \Big\}$
16   **case** *delete:*
17       $AS(c_i) = \bigsqcup_{c_j \in pred(c_i)} \Big\{ \overline{\mathscr{T}}_{dba} [\![\mathtt{DELETE}(\vec{v}_d),\ \phi \rangle]\!](\overline{\rho}) \mid \overline{\rho} \in AS(c_j) \Big\}$
18   **case** *insert:*
19       $AS(c_i) = \bigsqcup_{c_j \in pred(c_i)} \Big\{ \overline{\mathscr{T}}_{dba} [\![\mathtt{INSERT}(\vec{v}_d,\ \vec{e}),\ false \rangle]\!](\overline{\rho}) \mid \overline{\rho} \in AS(c_j) \Big\}$
20   **case** *select:*
21       $AS(c_i) = \bigsqcup_{c_j \in pred(c_i)} \Big\{ \overline{\mathscr{T}}_{dba} [\![\langle \mathtt{SELECT}\big(f(\vec{e'}), r(\vec{h}(\vec{x})), \phi_2, g(\vec{e})\big), \phi_1 \rangle]\!](\overline{\rho}) \mid \overline{\rho} \in AS(c_j) \Big\}$
22   **case** *join:*
23       $AS(c_i)$= $\bigsqcup_{c_j \in pred(c_i)} AS(c_j)$
24   **case** *end:*
25       $AS(c_i)$=$\bigsqcup_{c_j \in pred(c_i)} AS(c_j)$
26 Apply the abstract transition relation $\overline{\mathscr{T}}_{dep}$ on the abstract state $AS(c_j)$ obtained at each program point.
27 **End**

Let $\overline{\rho}^{Q_1} = \langle \rho^{Q_1}_{\overline{o}}, \rho^{Q_1}_{\overline{\square}}, \rho^{Q_1}_{\overline{\blacksquare}} \rangle$ and $\overline{\rho}^{Q_2} = \langle \rho^{Q_2}_{\overline{o}}, \rho^{Q_2}_{\overline{\square}}, \rho^{Q_2}_{\overline{\blacksquare}} \rangle$ be the abstract states at Q_1 and Q_2 respectively. The *defined*-part by Q_1 and the *used*-part by Q_2 are :

$$\mathsf{D}^{Q_1} = \overline{\mathbf{A}}_{def}(Q_1, \overline{\rho}^{Q_1}) = \langle \rho^{Q_1}_{\overline{\square}}, \rho^{Q_1}_{\overline{\blacksquare}} \rangle$$

SQL	Abstract Domain of Intervals $\overline{\rho}$		Abstract Domain of Octagons $\overline{\rho}$		Abstract Domain of Polyhedra $\overline{\rho}$		Powerset of Interval Domain $\hat{\rho}_{\mathbb{I}}$	
	Abstract state	*defined-/ used*-part	Abstract state	*defined-/ used*-part	Abstract state	*defined-/ used*-part	Abstract state	*defined-/ used*-part
Update Q_{upd}	$\langle \rho_{\overline{FM}}, \rho_{\overline{TM}}, \rho_{\overline{TM'}} \rangle$	$\mathbf{A}_{def}(Q_{upd}, \overline{\rho}) = \langle \rho_{\overline{TM}}, \rho_{\overline{TM'}} \rangle$ $\mathbf{A}_{use}(Q_{upd}, \overline{\rho}) = \langle \rho_{\overline{TM}} \rangle$	$\langle \mathsf{m}_F, \mathsf{m}_T, \mathsf{m}'_T \rangle$	$\mathbf{A}_{def}(Q_{upd}, \overline{\rho}) = \langle \mathsf{m}_T, \mathsf{m}'_T \rangle$ $\mathbf{A}_{use}(Q_{upd}, \overline{\rho}) = \langle \mathsf{m}_T \rangle$	$\langle \mathsf{P}_F, \mathsf{P}_T, \mathsf{P}'_T \rangle$	$\mathbf{A}_{def}(Q_{upd}, \overline{\rho}) = \langle \mathsf{P}_T, \mathsf{P}'_T \rangle$ $\mathbf{A}_{use}(Q_{upd}, \overline{\rho}) = \langle \mathsf{P}_T \rangle$	$\langle \rho_{\overline{FM}}, \rho_{\overline{TM}}, \rho_{\overline{TM'}} \rangle$	$\mathbf{A}_{def}(Q_{upd}, \overline{\rho}) = \langle \rho_{\overline{TM}}, \rho_{\overline{TM'}} \rangle$ $\mathbf{A}_{use}(Q_{upd}, \overline{\rho}) = \langle \rho_{\overline{TM}} \rangle$
Delete Q_{del}	$\langle \rho_{\overline{FM}}, \rho_{\overline{TM}}, \rho_{\perp} \rangle$	$\mathbf{A}_{def}(Q_{del}, \overline{\rho}) = \langle \rho_{\overline{TM}}, \emptyset \rangle$ $\mathbf{A}_{use}(Q_{del}, \overline{\rho}) = \langle \rho_{\overline{TM}} \rangle$	$\langle \mathsf{m}_F, \mathsf{m}_T, \mathsf{m}_{\perp} \rangle$	$\mathbf{A}_{def}(Q_{del}, \overline{\rho}) = \langle \mathsf{m}_T, \emptyset \rangle$ $\mathbf{A}_{use}(Q_{del}, \overline{\rho}) = \langle \mathsf{m}_T \rangle$	$\langle \mathsf{P}_F, \mathsf{P}_T, \mathsf{P}_{\perp} \rangle$	$\mathbf{A}_{def}(Q_{del}, \overline{\rho}) = \langle \mathsf{P}_T, \emptyset \rangle$ $\mathbf{A}_{use}(Q_{del}, \overline{\rho}) = \langle \mathsf{P}_T \rangle$	$\langle \rho_{\overline{FM}}, \rho_{\overline{TM}}, \rho_{\perp} \rangle$	$\mathbf{A}_{def}(Q_{del}, \overline{\rho}) = \langle \rho_{\overline{TM}}, \emptyset \rangle$ $\mathbf{A}_{use}(Q_{del}, \overline{\rho}) = \langle \rho_{\overline{TM}} \rangle$
Insert Q_{ins}	$\langle \rho_{\overline{t}}, \rho_{\perp}, \rho_{\overline{new}} \rangle$	$\mathbf{A}_{def}(Q_{ins}, \overline{\rho}) = \langle \emptyset, \rho_{\overline{new}} \rangle$ $\mathbf{A}_{use}(Q_{ins}, \overline{\rho}) = \langle \emptyset \rangle$	$\langle \mathsf{m}_t, \mathsf{m}_{\perp}, \mathsf{m}_{new} \rangle$	$\mathbf{A}_{def}(Q_{ins}, \overline{\rho}) = \langle \emptyset, \mathsf{m}_{new} \rangle$ $\mathbf{A}_{use}(Q_{ins}, \overline{\rho}) = \langle \emptyset \rangle$	$\langle \mathsf{P}_t, \mathsf{P}_{\perp}, \mathsf{P}_{new} \rangle$	$\mathbf{A}_{def}(Q_{ins}, \overline{\rho}) = \langle \emptyset, \mathsf{P}_{new} \rangle$ $\mathbf{A}_{use}(Q_{ins}, \overline{\rho}) = \langle \emptyset \rangle$	$\langle \rho_{\overline{t}}, \rho_{\perp}, \rho_{\overline{new}} \rangle$	$\mathbf{A}_{def}(Q_{ins}, \overline{\rho}) = \langle \emptyset, \rho_{\overline{new}} \rangle$ $\mathbf{A}_{use}(Q_{ins}, \overline{\rho}) = \langle \emptyset \rangle$
Select Q_{sel}	$\langle \rho_{\overline{FM}}, \rho_{\overline{TM}}, \rho_{\overline{TM}} \rangle$	$\mathbf{A}_{def}(Q_{sel}, \overline{\rho}) = \langle \emptyset, \emptyset \rangle$ $\mathbf{A}_{use}(Q_{sel}, \overline{\rho}) = \langle \rho_{\overline{TM}} \rangle$	$\langle \mathsf{m}_F, \mathsf{m}_T, \mathsf{m}_T \rangle$	$\mathbf{A}_{def}(Q_{sel}, \overline{\rho}) = \langle \emptyset, \emptyset \rangle$ $\mathbf{A}_{use}(Q_{sel}, \overline{\rho}) = \langle \mathsf{m}_T \rangle$	$\langle \mathsf{P}_F, \mathsf{P}_T, \mathsf{P}_T \rangle$	$\mathbf{A}_{def}(Q_{sel}, \overline{\rho}) = \langle \emptyset, \emptyset \rangle$ $\mathbf{A}_{use}(Q_{sel}, \overline{\rho}) = \langle \mathsf{P}_T \rangle$	$\langle \rho_{\overline{FM}}, \rho_{\overline{TM}}, \rho_{\overline{TM}} \rangle$	$\mathbf{A}_{def}(Q_{sel}, \overline{\rho}) = \langle \emptyset, \emptyset \rangle$ $\mathbf{A}_{use}(Q_{sel}, \overline{\rho}) = \langle \rho_{\overline{TM}} \rangle$

TABLE 9: Abstract *defined*- and *used*-part of database by SQL statements in various abstract domains

$$\mathsf{U}^{Q_2} = \overline{\mathbf{A}}_{use}(Q_2, \overline{\rho}^{Q_2}) = \langle \rho^{Q_2}_{\square} \rangle$$

The semantic dependency and independency of Q_2 on Q_1 are determined based on the following four cases:

Case − 1. $\rho^{Q_1}_{\square} \sqcap \rho^{Q_2}_{\square} \neq \emptyset \wedge \rho^{Q_1}_{\blacksquare} \sqcap \rho^{Q_2}_{\square} = \emptyset$

Case − 2. $\rho^{Q_1}_{\square} \sqcap \rho^{Q_2}_{\square} = \emptyset \wedge \rho^{Q_1}_{\blacksquare} \sqcap \rho^{Q_2}_{\square} \neq \emptyset$

Case − 3. $\rho^{Q_1}_{\square} \sqcap \rho^{Q_2}_{\square} \neq \emptyset \wedge \rho^{Q_1}_{\blacksquare} \sqcap \rho^{Q_2}_{\square} \neq \emptyset$

Case − 4. $\rho^{Q_1}_{\square} \sqcap \rho^{Q_2}_{\square} = \emptyset \wedge \rho^{Q_1}_{\blacksquare} \sqcap \rho^{Q_2}_{\square} = \emptyset$

The pictorial representation of the above cases are depicted in Figure 12. Observe that only case 4 indicates a semantic independency between Q_1 and Q_2 whereas all other cases indicate a semantic dependency between them. Therefore, Q_2 is DD-independent on Q_1 iff $\mathsf{D}^{Q_1} \sqcap \mathsf{U}^{Q_2} = \emptyset$; that is

$$\rho^{Q_1}_{\square} \sqcap \rho^{Q_2}_{\square} = \emptyset \wedge \rho^{Q_1}_{\blacksquare} \sqcap \rho^{Q_2}_{\square} = \emptyset \tag{14}$$

Theorem 1 states that, given an abstract domain, equation 14 is necessary and sufficient condition for abstract DD-independency. Observe that this theorem does not establish anything about its soundness w.r.t. its concrete counterpart.

Theorem 1. Given an abstract domain, the necessary and sufficient condition for a SQL statement Q_2 to be abstract DD-independent on another statement Q_1 is $\rho^{Q_1}_{\square} \sqcap \rho^{Q_2}_{\square} = \emptyset \wedge \rho^{Q_1}_{\blacksquare} \sqcap \rho^{Q_2}_{\square} = \emptyset$.

Proof 1. Consider two database statements $Q_1 = \langle A_1, \phi_1 \rangle$ and $Q_2 = \langle A_2, \phi_2 \rangle$. Given the abstract states $\overline{\rho}$ and $\overline{\rho}'$ at Q_1 and Q_2 respectively which are obtained in step 25 of Algorithm 1. Let the abstract semantics applying $\overline{\mathscr{T}}_{dep}$ in step 26 be $\overline{\mathscr{T}}_{dep}[\![Q_1]\!]\overline{\rho} = \langle \rho^{Q_1}_{\bar{o}}, \rho^{Q_1}_{\square}, \rho^{Q_1}_{\blacksquare} \rangle$ and $\overline{\mathscr{T}}_{dep}[\![Q_2]\!]\overline{\rho}' = \langle \rho^{Q_2}_{\bar{o}}, \rho^{Q_2}_{\square}, \rho^{Q_2}_{\blacksquare} \rangle$. Intuitively, we can say that Q_2 is abstract DD-dependent on Q_1 when any modification on the abstract database by Q_1 affects the abstract database-part to be accessed by Q_2. The following three kinds of affects may happen on Q_2 due to Q_1:

1) *Inclusion of new information*: Because of the modification by Q_1 some new data may be accessed by Q_2 satisfying ϕ_2. This is captured in **Case-2**.

2) *Removal of existing information*: As a result of the modification done by Q_1 some information (which was previously accessed by Q_2) now can not be accessed by Q_2 due to the unsatisfiability of ϕ_2. This is captured in **Case-1**.

3) *Access of modified information*: Q_2 can access now modified values, instead of their original values, of some attributes due to the application of Q_1. This is captured in **Case-3**.

Therefore, we can say Q_2 is semantically abstract DD-independent on Q_1 when the above three affects do not take place. In other words, the abstract database-part $\rho^{Q_2}_{\square}$ accessed by Q_2 overlaps with the parts $\rho^{Q_1}_{\square}$ and $\rho^{Q_1}_{\blacksquare}$ referred by Q_1 operations. This is captured in **Case-4**.

Algorithm to Compute Semantics-based DD-dependencies. The algorithm **semDOPDG** in Algorithm 2 takes a list of *used*- and *defined*-parts at each program point of the database program $\mathcal{P}$ and computes semantic-based DD-dependency among database statements. The algorithm, in step 2, first identifies all database statements present in the program. Step 5 inside the loops checks whether the *defined*-part by Q_i overlaps with the *used*-part by Q_j, and accordingly DD-dependency edge is created between them in step 6 and the flag is set to *true* in step 7. If dependency exists between Q_i and Q_j and *flag* is *true*, then in the next step 11 the algorithm checks the condition $\mathsf{D}^{Q_i} \sqsubseteq \mathsf{D}^{Q_j}$ in order to verify whether *defined*-part at program point Q_i is fully covered by the *defined*-part at program point Q_j. If yes, the execution immediately breaks the inner loop and does not check for dependency of the subsequent database statements (after Q_j) on Q_i, and hence disregards the false dependencies which may occur due to redefinition of attributes values by Q_j.

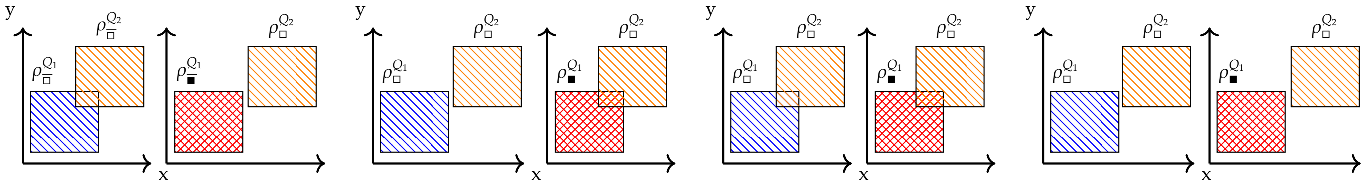
7 Illustration on the Running Example

Now we illustrate our approach on the running example "`Prog`" in section 2. The semantic-based data independencies are computed applying the following steps in different abstract domains:

- Compute abstract semantics using Algorithm 1 at each program point of "`Prog`".
- Compute *defined*- and *used*-parts based on the abstract semantics.
- Refinement of syntactic dependencies in "`Prog`" based on the semantics-based independencies using Algorithm 2.

A comparative result of the analysis in various abstract domains is depicted in Table 10. Let us explain briefly few scenarios by illustrating our approach.

For the sake of simplicity, since statements 5 and 6 involve only the attribute '*purchase_amt*' and the applications variables '*x*' and '*y*', let us consider the abstract initial state $\overline{\rho}$ taking those variables into account with an assumption that *purchase_amt* is typed with unsigned smallint. Therefore, $\overline{\rho} =$



(a) case 1 (b) case 2 (c) case 3 (d) case 4 (Independency exist)

Fig. 12: Representations of independencies and dependencies

Algorithm 2: semDOPDG

Input: *used*- and *defined*-parts at each program point in the database program $\mathcal{P}$.
Output: Semantic-based DD-dependency

Set *flag=true*
Identify database statements present in $\mathcal{P}$. Let m be the number of database statements.
for *i =1 to m-1* **do**
  for *j=i+1 to m* **do**
    if $D^{Q_i} \sqcap U^{Q_j} \neq \emptyset$ **then**
      Add edges from i^{th} statement to j^{th} statement $(i \rightarrow j)$
      Set *flag = true*;
    else
      Set *flag = false*;
    if *flag==true* **then**
      if $D^{Q_i} \sqsubseteq D^{Q_j}$ **then**
        break;
End

$(\rho_{\overline{dB}}\ ,\ \rho_{\overline{a}})$ and $\overline{\mathscr{T}}_{dba}[\![4]\!](\overline{\rho}) = \overline{\rho}^4$ where $\rho^{4}_{\overline{dB}} = \langle purchase\_amt \mapsto [0,\ 65000]\rangle$. and $\rho_{\overline{a}} = \langle x \mapsto [0.1,\ 0.1],\ y \mapsto [0.05,\ 0.05]\rangle$.

The abstract semantics of statement 5 is

$$\overline{\mathscr{T}}_{dba}[\![5]\!](\overline{\rho}^4) = \overline{\rho}^5 \quad \text{and} \quad \overline{\mathscr{T}}_{dep}[\![5]\!](\overline{\rho}^4) = \left\langle \rho^{5}_{\overline{FM}}\ ,\ \rho^{5}_{\overline{TM}}\ ,\ \rho^{5}_{\overline{TM'}} \right\rangle$$

where $\rho^{5}_{\overline{FM}}$ is obtained from $\rho^{4}_{\overline{dB}}$ where the condition is not satisfied. This creates two intervals ($purchase\_amt \leftarrow [0,\ 999]$) and ($purchase\_amt \leftarrow [3001,\ 65000]$). Therefore, $\rho^{5}_{\overline{FM}}$ is represented using two abstract tuples l_1 and l_2 such that $\rho^{5}_{\overline{FM}}=\rho^{4}_{\overline{dB}}\Big[\ l_1(purchase\_amt \leftarrow [0,\ 999]),\ l_2(purchase\_amt \leftarrow [3001,\ 65000])\ \Big]$. The part for which the condition evaluates to true is $\rho^{5}_{\overline{TM}} = \rho^{4}_{\overline{dB}}\Big[\ purchase\_amt \leftarrow [1000,\ 3000]\Big]$ and therefore $\rho^{5}_{\overline{TM'}} = \rho_{\overline{TM}}\Big[purchase\_amt \leftarrow [950,\ 2850]\Big]$.

Similarly, abstract semantics of the statement 6 is

$$\overline{\mathscr{T}}_{dba}[\![6]\!](\overline{\rho}^5) = \overline{\rho}^6 \quad \text{and} \quad \overline{\mathscr{T}}_{dep}[\![6]\!](\overline{\rho}^5) = \left\langle \rho^{6}_{\overline{FM}}\ ,\ \rho^{6}_{\overline{TM}}\ ,\ \rho^{6}_{\overline{TM'}} \right\rangle$$

where $\rho^{6}_{\overline{FM}} = \rho^{5}_{\overline{dB}}\Big[\ purchase\_amt \leftarrow [0, 3000]\ \Big]$, $\rho^{6}_{\overline{TM}} = \rho^{5}_{\overline{dB}}\Big[\ purchase\_amt \leftarrow [3001, 65000]\ \Big]$ and $\rho^{6}_{\overline{TM'}} = \rho_{\overline{TM}}\Big[\ purchase\_amt \leftarrow [2701, 58500]\ \Big]$.

The *defined*-part by statement 5 and the *used*-part by statement 6 are defined as follows:

$$\mathsf{D}^5 = \overline{\mathbf{A}}_{def}(\overline{\rho}^5, 5) = \langle \rho^{5}_{\overline{TM}}\ ,\ \rho^{5}_{\overline{TM'}}\rangle \text{ and } \mathsf{U}^6 = \overline{\mathbf{A}}_{use}(\overline{\rho}^6, 6) = \langle \rho^{6}_{\overline{TM}}\rangle$$

Therefore, the dependency $5 \dashrightarrow 6$ does not exist semantically as

$$\mathsf{D}^5 \sqcap \mathsf{U}^6 = \emptyset \Longrightarrow \rho^{5}_{\overline{TM}} \sqcap \rho^{6}_{\overline{TM}} = \emptyset \wedge \rho^{5}_{\overline{TM'}} \sqcap \rho^{6}_{\overline{TM}} = \emptyset$$

This way one can easily capture semantics independencies. Note that interval analysis is not yet an optimal setting to capture all such independencies in "`Prog`", for instance $15 \dashrightarrow 16$.

On the other hand, consider the domain of polyhedra. Consider the statements 15 and 16 which involve attributes '*purchase_amt*', '*wallet_bal*' and '*point*'. Let us consider the abstract initial database state in the form of polyhedron P_{dB} based on the assumption that *purchase_amt* is typed with unsigned smallint and the integrity constraints $0 \leqslant point \leqslant 100$ and $100 \leqslant wallet\_bal \leqslant 90000$ are defined on '*point*' and '*wallet_bal*'. Therefore, the abstract state at program point 4 is:

$$\mathsf{P}^{11}_{dB} = \Big\{purchase\_amt \geqslant 0, -purchase\_amt \geqslant -65000, point \geqslant 0, -point \geqslant -100, wallet\_bal \geqslant 100, -wallet\_bal \geqslant -90000\Big\}$$

The abstract semantics of statement 15 is defined as:

$$\overline{\mathscr{T}}_{dba}[\![15]\!](\mathsf{P}^{11}_{dB}) = \mathsf{P}^{15}_{dB} \text{ and } \overline{\mathscr{T}}_{dep}[\![15]\!](\mathsf{P}^{11}_{dB}) = \Big\langle \mathsf{P}^{15}_{F}, \mathsf{P}^{15}_{T}, \mathsf{P}^{15}_{T'} \Big\rangle \text{ where}$$

$$\mathsf{P}^{15}_{F} = \Big\{purchase\_amt \geqslant 0, -purchase\_amt \geqslant -65000, point \geqslant 0, -point \geqslant -100, wallet\_bal \geqslant 100, -wallet\_bal \geqslant -90000\Big\}$$

$$\mathsf{P}^{15}_{T} = \Big\{purchase\_amt \geqslant 0, -purchase\_amt \geqslant -9899, point \geqslant 0, -point \geqslant -100, wallet\_bal \geqslant 100, -wallet\_bal \geqslant -9999, 5000 \leqslant purchase\_amt + wallet\_bal \leqslant 9999\Big\}$$

$$\mathsf{P}^{15}_{T'} = \Big\{purchase\_amt \geqslant 0, -purchase\_amt \geqslant -9899, point \geqslant 2, -point \geqslant -102, wallet\_bal \geqslant 100, -wallet\_bal \geqslant -9999, 5000 \leqslant purchase\_amt + wallet\_bal \leqslant 9999\Big\}$$

Similarly, abstract semantics of statement 16 is:

$$\overline{\mathscr{T}}_{dba}[\![16]\!](\mathsf{P}^{15}_{dB}) = \mathsf{P}^{16}_{dB} \text{ and } \overline{\mathscr{T}}_{dep}[\![16]\!](\mathsf{P}^{15}_{dB}) = \Big\langle \mathsf{P}^{16}_{F}, \mathsf{P}^{16}_{T}, \mathsf{P}^{16}_{T'} \Big\rangle \text{ where}$$

$$\mathsf{P}^{16}_{F} = \Big\{purchase\_amt \geqslant 0, -purchase\_amt \geqslant -9899, point \geqslant 0, -point \geqslant -100, wallet\_bal \geqslant 100, -wallet\_bal \geqslant -9999\ purchase\_amt + wallet\_bal \leqslant 9999\Big\}$$

data dependency	pure syntax-based	Improved syntax-based	Condition-Action rule-based	Interval domain	Octagon domain	Polyhedra domain
DD-dependency	$4 \dashrightarrow \{5,6,7,11,15,16\}$ $5 \dashrightarrow \{6,7,11,15,16\}$ $6 \dashrightarrow \{7,11,15,16\}$ $7 \dashrightarrow \{11,15,16\}$ $15 \dashrightarrow 16$	$4 \dashrightarrow \{5,6,7,11,15,16\}$ $5 \dashrightarrow \{6,7\}$ $6 \dashrightarrow 7$ $7 \dashrightarrow \{11,15,16\}$ $15 \dashrightarrow 16$	$4 \dashrightarrow \{5,6,7,11,15,16\}$ $5 \dashrightarrow \{7,11,15,16\}$ $6 \dashrightarrow \{7,11,15,16\}$ $7 \dashrightarrow \{11,15,16\}$	$4 \dashrightarrow \{5,6,7,11,15,16\}$ $5 \dashrightarrow 7$ $6 \dashrightarrow 7$ $7 \dashrightarrow \{11,15,16\}$ $15 \dashrightarrow 16$	$4 \dashrightarrow \{5,6,7,11,15,16\}$ $5 \dashrightarrow 7$ $6 \dashrightarrow 7$ $7 \dashrightarrow \{11,15,16\}$	$4 \dashrightarrow \{5,6,7,11,15,16\}$ $5 \dashrightarrow 7$ $6 \dashrightarrow 7$ $7 \dashrightarrow \{11,15,16\}$
PD-dependency	$2 \dashrightarrow 6$, $3 \dashrightarrow 5$	$2 \dashrightarrow 6$, $3 \dashrightarrow 5$	$2 \dashrightarrow 6$, $3 \dashrightarrow 5$	$2 \dashrightarrow 6$, $3 \dashrightarrow 5$	$2 \dashrightarrow 6$, $3 \dashrightarrow 5$	$2 \dashrightarrow 6$, $3 \dashrightarrow 5$

TABLE 10: Representation of dependency results on "Prog" in various approaches

$$\begin{aligned}
\mathsf{P}^{16}_{T} =& \Big\{purchase_amt \geqslant 0, -purchase_amt \geqslant -65000, point \geqslant 0,\\
&-point \geqslant -100, wallet_bal \geqslant 100, -wallet_bal \geqslant -90000\\
&purchase_amt + wallet_bal \geqslant 10000\Big\}\\
\mathsf{P}^{16}_{T'} =& \Big\{purchase_amt \geqslant 0, -purchase_amt \geqslant -65000, point \geqslant 4,\\
&-point \geqslant -104, wallet_bal \geqslant 100, -wallet_bal \geqslant -90000\\
&purchase_amt + wallet_bal \geqslant 10000\Big\}
\end{aligned}$$

The *defined*-part by statement 15 and the *used*-part by statement 16 are computed as follows:

$$\mathsf{D}^{15} = \overline{\mathbf{A}}_{def}(\mathsf{P}^{15}, 15) = \langle \mathsf{P}^{15}_{T}\ ,\ \mathsf{P}^{15}_{T'}\rangle \text{ and } \mathsf{U}^{16} = \overline{\mathbf{A}}_{use}(\mathsf{P}^{6}, 16) = \langle \mathsf{P}^{16}_{T}\rangle$$

Therefore the dependency $15 \dashrightarrow 16$ does not exist semantically, as

$$\mathsf{D}^{15} \sqcap \mathsf{U}^{16} = \emptyset \Longrightarrow \mathsf{P}^{15}_{T} \sqcap \mathsf{P}^{16}_{T} = \emptyset \wedge \mathsf{P}^{15}_{T'} \sqcap \mathsf{P}^{16}_{T} = \emptyset$$

This way other data independencies can also be captured under polyhedral analysis.

8 Soundness of the Analysis

Lemma 1. Given an abstract state $\overline{\rho} = (\rho_{\overline{t}}\ ,\ \rho_{\overline{a}})$, the abstract semantics function $\overline{\mathscr{T}}_{dba}$ is sound w.r.t. the concretization function γ if $\forall Q \in \mathbb{Q}, \forall \rho_t \in \gamma(\rho_{\overline{t}}),\ \forall \rho_a \in \gamma(\rho_{\overline{a}})$: $\mathscr{T}_{dba}[\![Q]\!](\rho_t, \rho_a) \subseteq \gamma(\overline{\mathscr{T}}_{dba}[\![Q]\!](\rho_{\overline{t}}, \rho_{\overline{a}}))$.

Proof 2. Given an abstract state $\overline{\rho}$, the abstract semantics of $Q = \langle A, \phi\rangle \in \mathbb{Q}$ w.r.t. $\overline{\rho}$ is defined as $\overline{\mathscr{T}}_{dba}[\![Q]\!]\overline{\rho} = \overline{\mathscr{T}}_{dba}[\![\langle A, \phi\rangle]\!]\overline{\rho} = \overline{\mathscr{T}}_{dba}[\![\langle A\rangle]\!]\rho_{\overline{TM}} \sqcup \rho_{\overline{FM}} = \rho_{\overline{TM'}} \sqcup \rho_{\overline{FM}} = \overline{\rho}'$, where $\rho_{\overline{TM}}$ represents abstract database state which satisfies ϕ and $\rho_{\overline{FM}}$ represents abstract database state which does not satisfy ϕ. Abstract state which, due to abstraction, may satisfy ϕ is included in both $\rho_{\overline{TM}}$ and $\rho_{\overline{FM}}$. The state $\rho_{\overline{TM'}}$ is obtained by performing A on $\rho_{\overline{TM}}$. Now let us consider a concrete state $\rho \in \gamma(\overline{\rho})$. The concrete semantics of $Q = \langle A, \phi\rangle$ w.r.t. ρ is $\mathscr{T}_{dba}[\![Q]\!]\rho = \mathscr{T}_{dba}[\![\langle A, \phi\rangle]\!]\rho = \mathscr{T}_{dba}[\![\langle A\rangle]\!]\rho_T \cup \rho_F = \rho_{T'} \cup \rho_F = \rho'$, where ρ_T and ρ_F represent concrete database states based on the satisfaction and dissatisfaction of ϕ respectively. As ϕ in the abstract domain considers three valued logic due to the imprecision introduced in the abstraction, and since both $\rho_{\overline{TM}}$ and $\rho_{\overline{FM}}$ include the database state for which ϕ evaluates to "*may be true or false*", assuming local correctness of the functions and relations involved in ϕ we get $\rho_T \in \gamma(\rho_{\overline{TM}})$ and $\rho_F \in \gamma(\rho_{\overline{FM}})$. Similarly, the local correctness of the operations involved in A guarantees $\rho_{T'} \in \gamma(\rho_{\overline{TM'}})$ [39]. Considering the Galois connection between concrete and abstract database and application domains, we therefore get $(\rho_{T'} \cup \rho_F) \in \gamma(\rho_{\overline{TM'}} \sqcup \rho_{\overline{FM}})$ and so $\rho' \in \gamma(\overline{\rho}')$. This is depicted below:

$$\begin{array}{ccc}
\rho & \xrightarrow{\mathscr{T}_{dba}[\![Q]\!]} & \rho' \subseteq \gamma(\overline{\rho}')) \\
\uparrow \gamma & & \uparrow \gamma \\
\overline{\rho} & \xrightarrow[\overline{\mathscr{T}}_{dba}[\![Q]\!]]{} & \overline{\rho}'
\end{array}$$

Lemma 2. Let $\overline{\rho}$ be an abstract state. The abstract semantic function $\overline{\mathscr{T}}_{dep}$ is sound w.r.t. γ if $\forall Q \in \mathbb{Q}$, $\forall \rho \in \gamma(\overline{\rho})$: $\mathscr{T}_{dep}[\![Q]\!]\rho \subseteq \gamma(\overline{\mathscr{T}}_{dep}[\![Q]\!]\overline{\rho})$.

Proof 3. Given an abstract state $\overline{\rho}$ and a database statement $Q = \langle A,\ \phi\rangle \in \mathbb{Q}$, the abstract semantic function $\overline{\mathscr{T}}_{dep}$ on $\overline{\rho}$ computes abstract database state in the form of three-tuple as follows:

$$\overline{\mathscr{T}}_{dep}[\![Q]\!]\overline{\rho} = \overline{\mathscr{T}}_{dep}[\![\langle A,\ \phi\rangle]\!]\overline{\rho} = \langle \rho_{\overline{o}}, \rho_{\overline{\square}}, \rho_{\overline{\blacksquare}}\rangle$$

where $\rho_{\overline{o}}$ represents abstract database state which must (or may) not satisfy ϕ, whereas $\rho_{\overline{\square}}$ represents abstract database state which must (or may) satisfy ϕ. $\rho_{\overline{\blacksquare}}$ is obtained after performing an action A on $\rho_{\overline{\square}}$. Now let ρ be a concrete state such that $\rho \in \gamma(\overline{\rho})$, the concrete semantics similarly is defined as

$$\mathscr{T}_{dep}[\![Q]\!]\rho = \mathscr{T}_{dep}[\![\langle A,\ \phi\rangle]\!]\rho = \langle \rho_o, \rho_\square, \rho_\blacksquare\rangle$$

where ρ_o, $\rho_\square$ represent concrete database state based on the satisfaction and dissatisfaction of ϕ respectively, and $\rho_\blacksquare$ is obtained after performing A on $\rho_\square$. Like in lemma 1, because of three-valued logic of ϕ due to the imprecision introduced in the abstract domain and the local correctness of the operations in A, we get $\rho_o \in \gamma(\rho_{\overline{o}})$, $\rho_\square \subseteq \gamma(\rho_{\overline{\square}})$ and $\rho_\blacksquare \subseteq \gamma(\rho_{\overline{\blacksquare}})$, which implies that $\mathscr{T}_{dep}[\![Q]\!]\rho \subseteq \gamma(\overline{\mathscr{T}}_{dep}[\![Q]\!]\overline{\rho}$.

Lemma 3. Let $\overline{\rho}$ be an abstract state. The abstract function $\overline{\mathbf{A}}_{def}$ is sound w.r.t. γ if $\forall \rho \in \gamma(\overline{\rho})$, $\forall Q \in \mathbb{Q}$: $\gamma(\overline{\mathbf{A}}_{def}(Q,\ \overline{\rho})) \supseteq \mathbf{A}_{def}(Q,\ \rho)$

Proof 4. Given a database statement $Q = \langle A,\ \phi\rangle \in \mathbb{Q}$ and an abstract state $\overline{\rho}$, the abstract semantics (based on equation 9) is defined as $\overline{\mathscr{T}}_{dep}[\![Q]\!]\overline{\rho} = \overline{\mathscr{T}}_{dep}[\![\langle A,\ \phi\rangle]\!]\overline{\rho} = \langle \rho_{\overline{o}}, \rho_{\overline{\square}}, \rho_{\overline{\blacksquare}}\rangle$. As per the equation 12, the abstract *defined*-part is

$$\overline{\mathbf{A}}_{def}(Q,\ \overline{\rho}) = \langle \rho_{\overline{\square}}, \rho_{\overline{\blacksquare}}\rangle$$

Now given a concrete state $\rho = (\rho_t\ ,\ \rho_a) \in \gamma(\overline{\rho})$, we get the concrete semantics of Q, according to equation 5, as $\mathscr{T}_{dba}[\![Q]\!]\rho = \mathscr{T}_{dba}[\![\langle A,\ \phi\rangle]\!](\rho_t\ ,\ \rho_a) = (\rho_{t'}\ ,\ \rho_a)$.

Alternatively, $\mathscr{T}_{dep}$ on ρ computes concrete semantics of Q as $\mathscr{T}_{dep}[\![Q]\!]\rho = \mathscr{T}_{dep}[\![\langle A,\ \phi\rangle]\!]\rho = \langle \rho_o, \rho_\square, \rho_\blacksquare \rangle$. As per the equation 6, we can define the *defined*-part in the concrete domain by defining Δ, which computes the difference between database states before and after applying Q, in the form below:

$$\mathbf{A}_{def}(Q,\ \rho_t) = \Delta(\rho_{t'}\ ,\ \rho_t) = \langle \rho_\square, \rho_\blacksquare \rangle$$

Assuming the local correctness of ϕ and A, we get $\rho_\square \subseteq \gamma(\rho_{\overline{\square}})$ and $\rho_\blacksquare \subseteq \gamma(\rho_{\overline{\blacksquare}})$ respectively. Therefore, $\gamma(\overline{\mathbf{A}}_{def}(Q,\ \overline{\rho})) \supseteq \mathbf{A}_{def}(Q,\ \rho)$.

Lemma 4. Let $\overline{\rho}$ be an abstract state. The abstract function $\overline{\mathbf{A}}_{use}$ is sound w.r.t. γ if $\forall \rho \in \gamma(\overline{\rho})$, $\forall Q \in \mathbb{Q}$: $\gamma(\overline{\mathbf{A}}_{use}(Q,\ \overline{\rho})) \supseteq \mathbf{A}_{use}(Q,\ \rho)$

Proof 5. Proof is same as lemma 3.

Soundness. The semantics-based independency computation is sound if and only if an absence of dependency in the abstract domain guarantees that no dependency is present in the concrete domain.

Theorem 2 (Soundness of semantic independencies). Given two database statements Q_1 and Q_2, let $\overline{\rho}$ and $\overline{\rho}'$ be the abstract states at Q_1 and Q_2 respectively. The computation of semantic independency is sound if

$$\forall X \in \gamma(\overline{\mathbf{A}}_{def}(Q_1, \overline{\rho})), \forall Y \in \gamma(\overline{\mathbf{A}}_{use}(Q_2, \overline{\rho}')) :$$
$$X \cap Y \subseteq \gamma(\overline{\mathbf{A}}_{def}(Q_1, \overline{\rho}) \sqcap \overline{\mathbf{A}}_{use}(Q_2, \overline{\rho}'))$$

which implies $\overline{\mathbf{A}}_{def}(Q_1, \overline{\rho}) \sqcap \overline{\mathbf{A}}_{use}(Q_2, \overline{\rho}') = \emptyset \Rightarrow X \cap Y = \emptyset$.

Proof 6. Consider two database statements Q_1 and Q_2. Let $\overline{\rho} = \langle \rho_{\overline{o}}, \rho_{\overline{\square}}, \rho_{\overline{\blacksquare}} \rangle$ and $\overline{\rho}' = \langle \rho'_{\overline{o}}, \rho'_{\overline{\square}}, \rho'_{\overline{\blacksquare}} \rangle$ be the abstract states at Q_1 and Q_2 respectively, which are obtained by applying Algorithm 1. According to equations 12 and 13, we get the *defined*-part by Q_1 and the *used*-part by Q_2 as $\overline{\mathbf{A}}_{def}(Q_1, \overline{\rho}) = \langle \rho_{\overline{\square}}, \rho_{\overline{\blacksquare}} \rangle$ and $\overline{\mathbf{A}}_{use}(Q_2, \overline{\rho}') = \langle \rho'_{\overline{\square}} \rangle$ respectively. Now, the semantics independency in abstract domain can be defined as $\rho_{\overline{\square}} \sqcap \rho'_{\overline{\square}} = \emptyset \wedge \rho_{\overline{\blacksquare}} \sqcap \rho'_{\overline{\square}} = \emptyset$. Given the concrete states $\rho = \langle \rho_o, \rho_\square, \rho_\blacksquare \rangle$ and $\rho' = \langle \rho'_o, \rho'_\square, \rho'_\blacksquare \rangle$ where $\rho \in \gamma(\overline{\rho})$ and $\rho' \in \gamma(\overline{\rho}')$, the semantics independency between Q_1 and Q_2 in the concrete domain is defined as $\rho_\square \cap \rho'_\square = \emptyset \wedge \rho_\blacksquare \cap \rho'_\square = \emptyset$. From lemma 3 and 4, we get $\gamma(\overline{\mathbf{A}}_{def}(Q_1,\ \overline{\rho})) \supseteq \mathbf{A}_{def}(Q_1,\ \rho)$ and $\gamma(\overline{\mathbf{A}}_{use}(Q_2,\ \overline{\rho}')) \supseteq \mathbf{A}_{use}(Q_2,\ \rho')$ respectively. This implies that $\gamma(\overline{\mathbf{A}}_{def}(Q_1, \overline{\rho}) \sqcap \overline{\mathbf{A}}_{use}(Q_2, \overline{\rho}')) \supseteq \mathbf{A}_{def}(Q_1,\ \rho) \cap \mathbf{A}_{use}(Q_2,\ \rho')$. Therefore, $\overline{\mathbf{A}}_{def}(Q_1, \overline{\rho}) \sqcap \overline{\mathbf{A}}_{use}(Q_2, \overline{\rho}') = \emptyset \Rightarrow X \cap Y = \emptyset$ where $X = \mathbf{A}_{def}(Q_1,\ \rho) \in \gamma(\overline{\mathbf{A}}_{def}(Q_1, \overline{\rho}))$ and $Y = \mathbf{A}_{use}(Q_2,\ \rho') \in \gamma(\overline{\mathbf{A}}_{use}(Q_2, \overline{\rho}'))$.

9 Implementation and Experimental Evaluation

We have implemented a prototype tool `SemDDA`[6] – Semantics-based Database Dependency Analyzer – following the Algorithms 1 and 2, to perform experimental evaluation on a set of open-source database-driven JSP web applications as part of the GotoCode project [37][7, 8].

6. The source code is available on github: https://github.com/angshumanjana/SemDDA.

7. The original website ‘‘http://www.gotocode.com’’ does no longer exist at this moment. We have archived the benchmark codes at ‘‘https://github.com/angshumanjana/GotoCode’’.

8. These benchmark codes are used by many authors in their experiments, such as [57], [58], [59], [60].

9.1 The SemDDA Tool

The aim of designing `SemDDA` is to provide a user-friendly interface for the users to compute both syntax and semantic-based DD-dependency in various abstract domains of interest. The current implementation is in its preliminary stage which accepts only database-driven JSP codes. We provide a modular-based design and implementation of our tool, facilitating an easy expansion in future. The tool consists of two major components: (i) Syntax-based module, and (ii) Semantic-based module. Figure 13 depicts the overall architecture of the tool, where database program and underlying database are provided as input and a set of syntax-based dependencies and its refinement based on the abstract semantics are generated as output. The code is implemented in Java version 1.7. We used Eclipse version 4.2 as the development platform and Java applet technology for designing User Interfaces of semSSA.

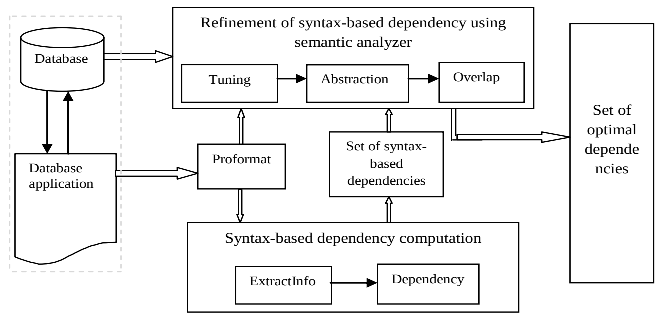


Fig. 13: Architecture of `SemDDA`

(i) **Proformat:** The module “Proformat” accepts database code written in JSP embedding SQL, and preprocesses it to add line numbers (starting from zero) to all statements, ignoring comments. Assuming input programs syntactically correct, the module separates program’s statements based on the predefined delimiters and right braces. During this process, it also computes Non-Comment Lines of Code (NCLOC) and the number of SQL statements present in the program. In particular, the presence of Data Manipulation Language (DML) statements is identified based on the presence of keywords such as `SELECT`, `UPDATE`, `DELETE` and `INSERT` in the statements.

(ii) **ExtractInfo:** This module extracts detail information about input programs, i.e. control statements, defined variables, used variables, etc. for all statements in the program.

Modules “Proformat” and “ExtractInfo” currently support only JSP embedded database code. The extension of these modules to support other programming languages does not require major design efforts, and it is currently in the to-do list for the next version of our analyser.

(iii) **Dependency:** The “Dependency” module computes syntax-based dependencies among program statements using the information computed by “ExtractInfo” module.

(iv) **Tuning:** At this preliminary stage of implementation, this module supports three abstract domains (Interval, Octagon, and Polyhedra). The module automatically picks the best domain based on the attribute relationships present in SQL statements. If none of the statements contains any relationship among attributes, then "Tuning" module automatically picks interval domain. On the other hand, either octagon or polyhedra abstract domain is chosen if at least one SQL statement contains respectively octagonal or polyhedral form of constraint. Moreover, users can also select one of the abstract domains of her choice based on the importance of computational cost and analysis-precision.

(v) **Abstraction:** The module "Abstraction" computes abstract semantics in the chosen abstract domain based on the data-flow analysis. Currently the module supports intervals, octagons, polyhedra, and powerset of intervals abstract domains.

(vi) **Overlap:** Finally this module identifies false dependency (if any) based on the semantics-based approximation of *used* and *defined* parts and their overlapping.

9.2 Experimental Results

We have used semDDA to perform experiments on a set of benchmark programs which are open-source database-driven web applications in JSP as part of the GotoCode project [37]. A brief description of these benchmark codes are mentioned in Table 11. The experiment is performed on a system configured with Intel i3 processor, 1.80GHz clock speed, Windows 7 Professional 64-bit Operating System with 8GB RAM.

In the following sections, we provide experimental results in various approaches on a set of benchmark codes under consideration.

9.2.1 DD-dependency results in pure syntax-based approach

The DD-dependency results on the benchmark codes in pure syntax-based approach is depicted in the 5^{th} column of Table 12. It is worthwhile to mention that, for the given benchmark codes, the improved syntax-based approach generates same results as that by pure syntax-based approach.

9.2.2 DD-dependency results in Condition-Action rules-based approach

We implemented Condition-Action rules using Satisfiability Modulo Theories (SMT). In particular, we used Z3 [61], a high-performance SMT Solver implemented in C++ and developed by Microsoft Research. For this purpose, we performed the following steps: (i) Selection of database statements in pairs according to their order of occurrences in the program, (ii) Conversion of these database statements into Static Single Assignment (SSA) form, (iii) Generation of Verification Condition (VC) from each pair by extracting predicates from the action- and condition-parts of the first statement and the condition-part of the second statement in the pair, and finally (iv) Dependency verification based on the satisfiability of VCs using Z3 tool. We used the online version of the Z3 tool available at "https://rise4fun.com/z3". We encoded VCs by following Z3 language syntax (which is an extension of the one defined by the SMT-LIB 2.0 standard). After compilation and execution by Z3, the output "UNSAT" for a pair indicates that the second database statement is not dependent on the first one in the pair. Let us explain this with the following simple example.

Example 21. Consider the following pair of database statements:

Q_1 : $\texttt{UPDATE}\ emp\ \texttt{SET}\ hra = hra + 100\ \texttt{WHERE}\ da + hra \geqslant 1000$
Q_2 : $\texttt{SELECT}\ hra\ \texttt{FROM}\ emp\ \texttt{WHERE}\ da + hra \leqslant 5000$

The equivalent SSA form of these statements are:

Q_1 : $\texttt{UPDATE}\ emp\ \texttt{SET}\ hra2 = hra1 + 100$
$\qquad\texttt{WHERE}\ da1 + hra1 \geqslant 1000$
Q_2 : $\texttt{SELECT}\ hra2\ \texttt{FROM}\ emp\ \texttt{WHERE}\ da1 + hra2 \leqslant 5000$

The VC of this pair of statements is:

$$V_c = (hra2 == hra1 + 100) \wedge (da1 + hra1 \geqslant 1000) \wedge (da1 + hra2 \leqslant 5000)$$

The encoded version of V_c in Z3 is:

1. $(declare - const\ hra1\ Int)$
2. $(declare - const\ hra2\ Int)$
3. $(declare - const\ da1\ Int)$
4. $(push)$
5. $(assert\ (=\ (+\ hra1\ 100)\ hra2))$
6. $(assert\ (>=\ (+\ hra1\ da1)\ 1000))$
7. $(assert\ (<=\ (+\ hra2\ da1)\ 5000))$
8. $(check - sat)$

As the Z3 reports this formula as satisfiable (Z3 output is "SAT"), this indicates that Q_2 depend on Q_1.

The DD-dependency results on the benchmark codes using this approach is depicted in the 6^{th} column of Table 12. This shows an improvement in the precision over the syntax-based results. In fact, on the given benchmark codes, an average of 12% improvement is observed as compared to the syntax-based approach.

9.2.3 Results based on the Abstract Semantics

Columns 7^{th}, 8^{th}, 9^{th} and 10^{th} of Table 12 depict DD-dependency results in the domains of intervals, octagons, polyhedra and powerset of intervals respectively. It is worthwhile to note that the analysis-results for five benchmark codes ('EditOfficer', 'EditMember', 'BookMaint', 'EditorialsRecord', 'BugRecord') in the interval domain improves w.r.t. their syntax-based results. On the other hand, analysis in the domain of octagons for 'EmployeeMaint', 'ProjectMaint', 'EventNew' and 'EmployeeMaint' results in more precise dependency information compared to that in the interval domain, due to the presence of restricted attributes relationship (which involves at most two attributes) in SQL statements. Similarly, polyhedra domain analysis captures more precise DD-dependency results, shown in the case of 'LedgerRecord' and 'EmpsRecord', compared to their interval and octagon counterparts, as they allow unrestricted relationship among attributes. We obtain an

Applications Names	Number of Files Tested	Descriptions
Events	1	It is a basic online event management system. It includes many features like event information (event name, year, presenter, etc.), users administration, etc.
Ledger	1	It is an example implementation of a web-based ledger which allows a user to track bank deposits, withdrawals, commission and view current balance.
Portal	2	It is a fully functional online web-based Portal which is useful for small organizations, clubs, user groups, and schools. It provides several functionalities like user registration, news section, list of club officers and etc. The considering files mainly work on the administration of club officers and members.
EmplDir	2	It is a basic employee directory that may use as an online system for small companies. It serves deferent searching facilities (e.g. by name, email) to the user. The selected files are dealing to store the employee and departmental information.
Bookstore	2	It is an online store system that keeps various books information, articles and other items. It has many features like user registrations, shopping cart, administration of credit card types and etc. It utilizes VeriSign's payflow link system to verify and charge credit cards.
BugTrack	3	It is a basic fully functional web-based bug tracking system which may useful for small teams working on software projects. It keeps projects information and its associated employee's detail (consider files work for this purpose), also provides many searching options.

TABLE 11: Description of the benchmark programs [37]

Applications (File Names)	NCLOC	Number of SQL Stmts	Number of Attributes	Number of DD-dependencies					
				Pure Syntax-based	Condition-Action Rule-based	Interval Domain	Octagon Domain	Polyhedra Domain	Powerset of Intervals
Events(EventNew.jsp)	334	6	5	8	6	8	6	6	8
Ledger(LedgerRecord.jsp)	436	9	8	22	18	22	22	16	22
Portal(EditOfficer.jsp)	300	7	4	21	21	19	19	19	19
Portal(EditMembers.jsp)	362	10	5	16	15	14	14	14	14
EmplDir(DepsRecord.jsp)	285	4	3	9	8	9	8	8	9
EmplDir(EmpsRecord.jsp)	435	9	7	23	21	23	22	14	23
Bookstore(EditorialsRecord.jsp)	294	6	3	5	4	4	4	4	4
Bookstore(BookMaint.jsp)	357	6	5	10	7	7	7	7	6
BugTrack(ProjectMaint.jsp)	307	7	4	15	13	15	13	13	15
BugTrack(EmployeeMaint.jsp)	316	6	5	12	11	12	10	10	12
BugTrack(BugRecord.jsp)	336	6	4	9	8	8	8	8	7

TABLE 12: DD-dependency results in various approaches (NCLOC denotes Non-Comment Lines of Code)

improvement in the precision for two benchmark codes 'BugRecord' and 'BookMaint' w.r.t. the analysis-results in other domains when we consider an abstract representation of initial databases in the powerset of intervals domain. Overall, we achieved an improvement in the precision on an average of 6% in the interval domain, 11% in the octagon, 21% in the polyhedra domain and 7% in the powerset of intervals domain, as compared to the syntax-based approach for the chosen set of benchmark codes. Figure 14 compares all DD-dependency results.

Table 13 reports the execution time (in milliseconds) of the analysis in the interval, octagon, polyhedra and powerset of intervals abstract domains. This is to mention that we do not observe any notable variation in the execution time across multiple trials. The variation of execution time for various benchmarks is depicted in Figure 15. Observe that we represent along y-axis the execution time (in milliseconds) in $\log_{10}$ scale, as the data range over several orders of magnitude. The reason behind the massive growth of execution time in polyhedra domain for 'EmpsRecord' and 'LedgerRecord' is exponential time complexity of the analysis w.r.t. the number of attributes (as reported in Table 12).

To finally conclude, our observation on the experimental evaluation results indicates that proper tuning of abstract domains from coarse to fine level in precision, perhaps compromising the computational costs, plays a crucial role to meet the analysis-objectives.

10 Case Study

Program slicing [13] is an effective static analysis technique which extracts from programs a subset of statements relevant to a given behavior. This allows engineers to address several software-related problems, including program understanding, debugging, maintenance, testing, parallelization, integration, software measurement, etc. Since the pioneer work of Mark Weiser [23] who introduces the notion

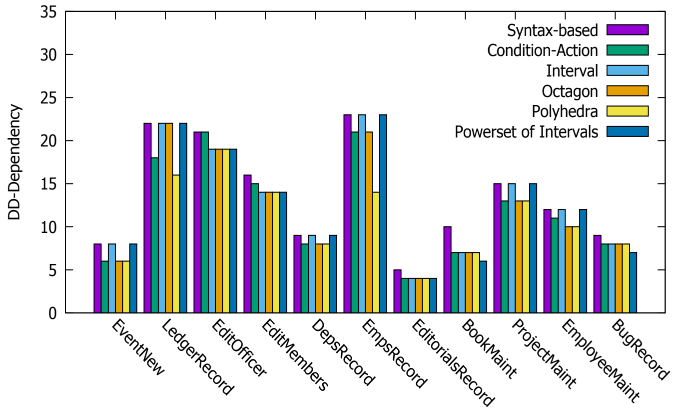


Fig. 14: Comparative analysis of DD-dependency results in various approaches.

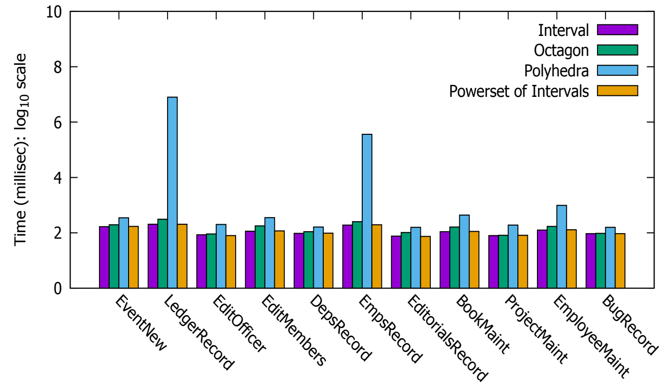


Fig. 15: Analysis Time (in $\log_{10}$ scale) in various Abstract Domains

File Names (.jsp)	Abstract Domains			
	Interval	Octagon	Polyhedra	Powerset of Intervals
EventNew	167	194	345	171
LedgerRecord	204	312	7943737	211
EditOfficer	86	91	201	87
EditMembers	116	178	354	119
DepsRecord	96	110	163	98
EmpsRecord	192	254	366314	197
EditorialsRecord	76	103	160	77
BookMaint	110	162	432	113
ProjectMaint	80	81	189	81
EmployeeMaint	126	169	986	129
BugRecord	94	97	157	97

TABLE 13: Execution time (in milliseconds) in various Abstract Domains.

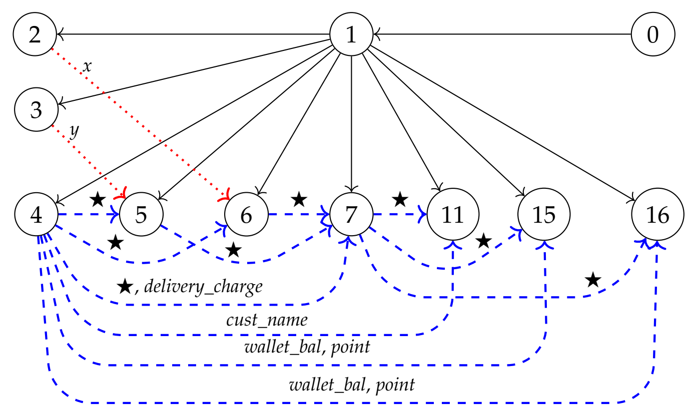
of static program slicing using program dependency graph, different algorithms to compute slice and different slicing variants (e.g. dynamic, conditioned, amorphous) are proposed by tuning them towards specific program analysis aim [26], [27], [62], [63], [64].

Let us apply our proposed dependency refinement in computing a slice of our running example program "`Prog`".

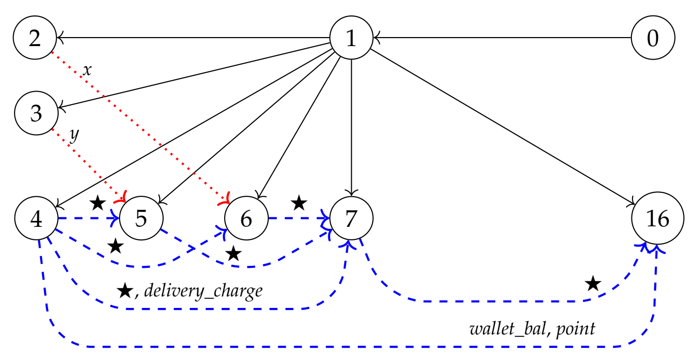
Consider the slicing criterion ψ= ⟨16, *point*⟩ where *point* is an attribute of interest at program point 16. The objective of the backward slicing is to extract only semantically relevant statements affecting the values of *point* at 16. Let us perform the below steps:

- Construction of syntax-based DOPDG of "`Prog`".
- Refinement based on our proposed approach.
- Finally, computation of backward slicing w.r.t. ψ on the refined DOPDG.

According to the first step, the syntax-based DOPDG of "`Prog`" is depicted in Figure 3 (Section 3). Following our semantics-based refinement analysis in polyhedra abstract domain, we identify the following false dependencies: $5 \dashrightarrow 6$, $4 \dashrightarrow 11$, $5 \dashrightarrow 11$, $6 \dashrightarrow 11$, $4 \dashrightarrow 15$, $5 \dashrightarrow 15$, $6 \dashrightarrow 15$, $4 \dashrightarrow 16$, $5 \dashrightarrow 16$, $6 \dashrightarrow 16$ and $15 \dashrightarrow 16$. The refined DOPDG discarding all these false dependencies is depicted in Figure 16(a). Finally, according to the third step, we traverse the DOPDG in a backward direction starting from node 16 considering *point* as the variable of interest which produces a sub-DOPDG shown in Figure 16(b). The corresponding slice code is shown in Figure 17. Observe that the resultant slice is more precise as it is able to capture statement 15 as semantically irrelevant compared to other syntactic approaches.



(a) Refined DOPDG of the syntax-based DOPDG in Figure 3



(b) Sub-DOPDG by traversing backward w.r.t. ⟨16, *point*⟩

Fig. 16: DOPDG and sub-DOPDG of "`Prog`" (★ denotes *purchase_amt*)

```
0. public class saleOffer {
1.   public static void main(String[] args) throws SQLException {
2.     float x = 0.1;
3.     float y = 0.05;
4.     try { Statement con = DriverManager.getConnection("jdbc mysql: . . . ", "scott", "tiger").createStatement();

5.         con.executeQuery("UPDATE Sales SET purchase_amt = purchase_amt − y ∗ purchase_amt WHERE purchase_amt BETWEEN 1000 AND 3000 ");

6.         con.executeQuery("UPDATE Sales SET purchase_amt = purchase_amt − x ∗ purchase_amt WHERE purchase_amt > 3000 ");

7.         con.executeQuery("UPDATE Sales SET purchase_amt = purchase_amt − delivery_charge ");

16.        con.executeUpdate("UPDATE Sales SET point = point + 4 WHERE (purchase_amt + wallet_bal) ⩾ 10000 "); } catch (Exception e) { . . . } }}
```

Fig. 17: Program "Prog"

11 Related Works

Ferrante et al. [14] first introduced the notion of Program Dependency Graph (PDG) aiming program optimization. Since then, PDG is playing crucial roles in a wide range of software-engineering activities, e.g. program slicing [13], code-reuse [16], language-based information flow security analysis [11], [12], [29], code-understanding [17]. Over the time, various forms of dependency graphs for various programming languages are proposed in order to address several language-specific problems. In [65], Zhao proposed a static dependency analysis algorithm for concurrent Java programs based on Multi-thread Dependency Graph (MDG). An MDG consists of a collection of TDGs (Thread Dependency Graphs) each of which represents a single thread. Cheng [66] proposed a PDG for parallel and distributed programs. In [67], the authors introduced the notion of Concurrency Program Dependency Graph (CPDG) to represent concurrent programs written using Unix primitives. It represents various aspects of concurrent programs in a hierarchical fashion. Horwitz et al. [20] introduced System Dependency Graph (SDG) in case of inter-procedural programs. Class Dependency Graph (ClDG) is introduced for Object Oriented Programming (OOP) languages in [21]. Willmor et.al. [22] introduced a variant of program dependence graph, known as Database-Oriented Program Dependence Graph (DOPDG), by considering two additional types of data dependences: Program-Database and Database-Database dependencies. The authors observed that, although the generation of used and defined sets of variables is straightforward, but the identification of overlap of database parts by different statements is more challenging. To this aim, they refer to the Condition-Action rules introduced by Baralis and Widom in [35]. The propagation algorithm based on Condition-Action rules predicts how the action of one rule can affect the condition of another. In other words, the analysis checks whether the condition sees any data inserted or deleted or modified due to the action.

Mastroeni and Zanardini [34] first introduced the notion of semantic data independencies following the Abstract Interpretation framework at expression-level. This leads to generate more precise semantics-based PDGs by removing false data dependencies w.r.t. the traditional syntactic PDGs. Our previous attempts to refine dependencies and hence more precise code analysis for database programs are reported in [27], [36], [38]. [38] applied predicate transformer (weakest precondition) to apply on dependency tree among a series of attribute-defining statements, whereas [27], [36] formalized the semantics for dependency refinement in a simple setting following the Abstract Interpretation as an initial attempt.

The authors in [68] and [69] addressed a closely related problem, known as query containment problem, which checks whether, for every database, the result of one query is a subset of the result of another query. For instance, a query Q_1 is contained in a query Q_2 if and only if the result of applying Q_1 to any database D is contained in the result of applying Q_2 to the same database D. Formally, a query Q_1 is said to be contained in a query Q_2, denoted $Q_1 \sqsubseteq Q_2 \iff \forall D\ Q_1(D) \subseteq Q_2(D)$ and $Q_1 \equiv Q_2 \iff Q_1 \sqsubseteq Q_2 \wedge Q_2 \sqsubseteq Q_1$, where $Q(D)$ represents the result of query Q on database D. The computational complexity of conjunctive query containment is NP-complete [68]. Query containment is useful for the various purposes of query optimization, detecting independency of queries from database updates, rewriting queries using views, etc. As dependency computations of database applications consider DML commands (`INSERT`, `UPDATE`, `DELETE`), the solutions proposed in [68], [69] for only conjunctive queries is, therefore, unable to provide a complete solution in our case which involves both *write-write* and *write-read* operations.

The authors in [70] addressed an undecidable problem which aims to identify all possible values that may occur as results of string expressions. Few interesting applications of the solution, among many others, include static analysis of validity of dynamically generated XML documents in the JWIG extension of Java, static syntax checking of dynamically generated queries in database programs. The authors proposed a static analysis technique for extracting context-free grammar from a given program and applied a variant of the Mohri-Nederhof approximation algorithm to approximate the possible values of string expressions in Java programs. A static analysis framework is proposed in [71] to automatically identify possible SQL injection attacks, SQL query performance optimization and data integrity violations in database programs. For this purpose, the framework adapts data and control flow analysis of traditional optimizing compilers techniques by leveraging understanding of data access APIs. [72] proposed a sound static analysis technique for verifying the correctness of dynamically generated SQL query strings in database applications. The technique is based on a combination of automata-theoretic techniques and a variant of the context-free language reachability algorithm. A new framework [73] is proposed for context-sensitive program analysis. The concept of deductive database technology is used here to create a higher abstraction for this cloning-based approach to context sensitivity. The framework allows users to express whole-program analysis succinctly with a small number of Datalog rules that operate on a cloned call graph. In [74], the authors proposed the constraint coverage criteria and the column coverage criteria for testing the specification of integrity constraints in a relational database schema. They expressed integrity constraints as predicates with constraint coverage, whereas they generated test requirements with the column coverage for checking integrity constraints.

12 Conclusion and Future Works

Dependency analysis of database programs plays a crucial role in different fields of software engineering. Some applications among many others include Program Slicing, Language-based Information Flow Security Analysis, Data Provenance, Concurrent System Modeling, Materialization View Creation. Although syntax-based dependency computation is straightforward, its semantics-based refinement is quite challenging when considering attributes' values in possible database instances. This paper proposes a novel approach to compute semantics-based independencies among database statements, based on the Abstract Interpretation framework. This steers construction of semantic-based DOPDGs with more precise set of dependencies. Most importantly, this serves as a powerful basis to give solution even in case of undecidable scenario when no initial database state is provided. The comparative study among various approaches and various abstract domains in terms of precision and efficiency clearly indicates that a trade-off in choosing appropriate abstract domains or their combination is very crucial to meet the objectives. There are many application areas where false dependence information could lead to huge financial loss while proving crucial properties of software products. Information flow security analysis of critical softwares is one such example. In such case, precision dominates over the analysis cost and a choice of stronger abstract domain, e.g. polyhedra domain, may be a good choice. On the other hand, when development speed is an important factor, choice of weakly relational or even non-relational abstract domain may be a wise decision. Experimental evaluation on the benchmarks set reports a precision improvement in the range of 6% - 21% under various levels of abstractions. This proves that the approach may impact significantly when to deal with large-scale complex software systems involving huge variables set and millions of lines of codes.

Some interesting future research scopes identified in this direction are designing suitable reduced products on multiple abstract domains, designing new ad-hoc abstract domains in the context of database states, extending the analysis to a distributed scenario with multiple transactions and heterogeneous database systems. As our future work, we shall extend our tool `SemDDA` to support some popular languages, such as C, Python, Java, in its next release. Besides, we shall also consider string data-type [75] along with numerical attributes.

Acknowledgement

This work is partially supported by the research grant (SB/FTP/ETA-315/2013) from the Science & Engineering Research Board (SERB), Department of science and Technology, Government of India, and by CINI Cybersecurity National Laboratory within the project FilieraSicura, Italy.

References

[1] F. Nielson, H. R. Nielson, and C. Hankin, *Principles of Program Analysis*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1999.

[2] W. Landi, "Undecidability of static analysis," *ACM Letters on Programming Languages and Systems (LOPLAS)*, vol. 1, no. 4, pp. 323–337, 1992.

[3] R. N. Taylor and L. J. Osterweil, "Anomaly detection in concurrent software by static data flow analysis," *IEEE Transactions on Software Engineering*, no. 3, pp. 265–278, 1980.

[4] T. Reps, S. Horwitz, and M. Sagiv, "Precise interprocedural dataflow analysis via graph reachability," in *Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM, 1995, pp. 49–61.

[5] F. E. Allen, "Control Flow Analysis," in *Proceedings of a Symposium on Compiler Optimization*. New York, NY, USA: ACM, 1970, pp. 1–19.

[6] J. Palsberg, "Type-based Analysis and Applications," in *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE '01)*. Snowbird, Utah, USA: ACM, 2001, pp. 20–27.

[7] B. Nordström, K. Petersson, and J. M. Smith, *Programming in Martin-Löf's type theory*. Oxford University Press Oxford, 1990, vol. 200.

[8] S. Thompson, *Type theory and functional programming*. Addison Wesley, 1991.

[9] P. Cousot and R. Cousot, "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *Proc. of the POPL'77*, 1977, pp. 238–252.

[10] P. Cousot and R. Cousot, "Systematic design of program analysis frameworks," in *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. San Antonio, Texas: ACM Press, 1979, pp. 269–282.

[11] C. Hammer, "Experiences with PDG-Based IFC," in *Proc. of the Engineering Secure Software and Systems*. Pisa, Italy: Springer-Verlag, 2010, pp. 44–60.

[12] J. Krinke, "Information flow control and taint analysis with dependence graphs," in *3rd International Workshop on Code Based Security Assessments (CoBaSSA 2007)*, 2007, pp. 6–9.

[13] F. Tip, "A Survey of Program Slicing Techniques." Tech. Rep., 1994.

[14] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Trans. on Programming Lang. and Sys.*, vol. 9, no. 3, pp. 319–349, 1987.

[15] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan, "PLUTO: A practical and fully automatic polyhedral program optimization system," in *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation (PLDI 08), Tucson, AZ (June 2008)*, 2008.

[16] L. Jiang, "Scalable Detection of Similar Code: Techniques and Applications," Ph.D. dissertation, Davis, CA, USA, 2009.

[17] A. Podgurski and L. A. Clarke, "A Formal Model of Program Dependences and its Implications for Software Testing, Debugging, and Maintenance," *IEEE Trans. on Software Engineering*, vol. 16, no. 9, pp. 965–979, 1990.

[18] K. J. Ottenstein and L. M. Ottenstein, "The program dependence graph in a software development environment," *ACM SIGPLAN Notices*, vol. 19, no. 5, pp. 177–184, 1984.

[19] S. Horwitz and T. Reps, "The Use of Program Dependence Graphs in Software Engineering," in *Proc. of the 14th ICSE*. ACM Press, 1992, pp. 392–411.

[20] S. Horwitz, T. Reps, and D. Binkley, "Interprocedural slicing using dependence graphs," *ACM Transactions on PLS*, vol. 12, no. 1, pp. 26–60, 1990.

[21] L. Larsen and M. J. Harrold, "Slicing object-oriented software," in *Proceedings of the 18th ICSE*. Berlin, Germany: IEEE CS, 1996, pp. 495–505.

[22] D. Willmor, S. M. Embury, and J. Shao, "Program Slicing in the Presence of a Database State," in *Proceedings of the 20th IEEE ICSM*, ser. ICSM '04, 2004, pp. 448–452.

[23] M. Weiser, "Program slicing," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 4, pp. 352–357, 1984.

[24] F. Lanubile and G. Visaggio, "Extracting Reusable Functions by Flow Graph-Based Program Slicing," *IEEE Transactions on Software Engineering*, vol. 23, no. 4, pp. 246–259, 1997.

[25] B. Korel and J. Rilling, "Program Slicing in Understanding of Large Programs," in *Proceedings of the 6th International Workshop on Program Comprehension (IWPC '98)*. Ischia, Italy: IEEE Computer Society, June 1998, pp. 145–152.

[26] A. Jana, R. Halder, N. Chaki, and A. Cortesi, "Policy-Based Slicing of Hibernate Query Language," in *Computer Information Systems and Industrial Management, CISIM, LNCS, September 24-26, 2015*. Springer, 2015, pp. 267–281.

[27] R. Halder and A. Cortesi, "Abstract Program Slicing of Database Query Languages," in *Proceedings of the the 28th Symposium On Applied Computing - Special Track on Database Theory, Technology, and Applications*. Coimbra, Portugal: ACM Press, 2013, pp. 838–845.

[28] A. Sabelfeld and A. C. Myers, "Language-Based Information-Flow Security," *IEEE Journal on SAC*, vol. 21, p. 2003, 2003.

[29] R. Halder, M. Zanioli, and A. Cortesi, "Information Leakage Analysis of Database Query Languages," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, ser. SAC '14. New York, NY, USA: ACM, 2014, pp. 813–820.

[30] R. Halder, A. Jana, and A. Cortesi, "Data Leakage Analysis of the Hibernate Query Language on a Propositional Formulae Domain," *Trans. Large-Scale Data- and Knowledge-Centered Systems*, vol. 23, pp. 23–44, 2016.

[31] J. Cheney, A. Ahmed, and U. A. Acar, "Provenance As Dependency Analysis," in *Proceedings of the 11th ICDPL*, ser. DBPL'07, 2007, pp. 138–152.

[32] S. Sen, A. Dutta, A. Cortesi, and N. Chaki, "A New Scale for Attribute Dependency in Large Database Systems," in *CISIM*, ser. LNCS, 2012, vol. 7564, pp. 266–277.

[33] S. Itzhaky, T. Kotek, N. Rinetzky, M. Sagiv, O. Tamir, H. Veith, and F. Zuleger, "On the automated verification of web applications with embedded SQL," *arXiv preprint arXiv:1610.02101*, 2016.

[34] I. Mastroeni and D. Zanardini, "Data dependencies and program slicing: from syntax to abstract semantics," in *Proc. of the ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, 2008, pp. 125–134.

[35] E. Baralis and J. Widom, "An Algebraic Approach to Rule Analysis in Expert Database Systems," in *Proceedings of the 20th International Conference on Very Large Data Bases*, ser. VLDB '94. Morgan Kaufmann Publishers Inc., 1994, pp. 475–486.

[36] A. Jana and R. Halder, "Defining Abstract Semantics for Static Dependence Analysis of Relational Database Applications," in *Information Systems Security - 12th International Conference, ICISS 2016, Jaipur, India, December 16-20, 2016, Proceedings*, 2016, pp. 151–171.

[37] "Gotocode," http://www.gotocode.com, [Online; accessed 20-Dec-2015], (now archived at: https://github.com/angshumanjana/GotoCode).

[38] M. I. Alam and R. Halder, "Refining Dependencies for Information Flow Analysis of Database Applications," in *International Journal of Trust Management in Computing and Communications*. Inderscience, 2016.

[39] R. Halder and A. Cortesi, "Abstract Interpretation of Database Query Languages," *Computer Languages, Systems & Structures*, vol. 38, pp. 123–157, 2012.

[40] A. N. S. INSTITUTE, "Information technology-database languages-SQL-part 2: Foundation (SQL/foundation), 2003."

[41] P. Cousot and N. Halbwachs, "Automatic Discovery of Linear Restraints Among Variables of a Program," in *Proceedings of the POPL '78*, 1978, pp. 84–96.

[42] A. Miné, "A New Numerical Abstract Domain Based on Difference-Bound Matrices," in *Programs as Data Objects, Second Symposium, PADO*, 2001, pp. 155–172.

[43] A. Minè, "The Octagon Abstract Domain," *Higher Order Symbol. Comput.*, vol. 19, no. 1, pp. 31–100, 2006. http://www.astree.ens.fr/.

[44] P. Cousot, "Constructive design of a hierarchy of semantics of a transition system by abstract interpretation," *Theoretical Computer Science*, vol. 277, no. 1-2, pp. 47–103, 2002.

[45] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, 1990.

[46] R. Bellman, "On a Routing Problem," *Quarterly of Applied Mathematics*, vol. 16, pp. 87–90, 1958.

[47] A. Miné, "The Octagon Abstract Domain," *CoRR*, vol. abs/cs/0703084, 2007.

[48] C. Hymans and E. Upton, "Static Analysis of Gated Data Dependence Graphs," in *Static Analysis, 11th International Symposium, SAS 2004, Verona, Italy, August 26-28, 2004, Proceedings*, 2004, pp. 197–211.

[49] M. Zanioli and A. Cortesi, "Information Leakage Analysis by Abstract Interpretation," in *SOFSEM 2011: Theory and Practice of Computer Science - 37th Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 22-28, 2011. Proceedings*, 2011, pp. 545–557.

[50] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable Verification of Hybrid Systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV)*, ser. LNCS. Springer, 2011.

[51] N. Chernikoba, "Algorithm for discovering the set of all the solutions of a linear programming problem," *USSR Computational Mathematics and Mathematical Physics*, vol. 8, no. 6, pp. 282–293, 1968.

[52] L. Chen, A. Minè, and P. Cousot, "A Sound Floating-Point Polyhedra Abstract Domain," in *Proc. of the 6th Asian Symposium on PLS*, 2008, pp. 3–18.

[53] R. Bagnara, P. M. Hill, and E. Zaffanella, "The PPL: Toward a Complete Set of Numerical Abstractions for the Analysis and Verification of Hardware and Software Systems," Dipartimento di Matematica, Università di Parma, Italy, Tech. Rep., 2006. http://www.cs.unipr.it/ppl/.

[54] J. A. Kelner and D. A. Spielman, "A Randomized Polynomial-time Simplex Algorithm for Linear Programming," in *Proc. of the 38th Annual ACM Symposium on Theory of Computing*. ACM, 2006, pp. 51–60.

[55] J.-L. Imbert, "Fourier's Elimination: Which to Choose?" in *PPCP*, 1993, pp. 117–129.

[56] R. Bagnara, P. M. Hill, and E. Zaffanella, "Widening operators for powerset domains," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 9, no. 3, pp. 413–414, 2007.

[57] W. G. Halfond and A. Orso, "AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. ACM, 2005, pp. 174–183.

[58] W. G. Halfond, A. Orso, and P. Manolios, "Using positive tainting and syntax-aware evaluation to counter SQL injection attacks," in *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 2006, pp. 175–185.

[59] P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan, "CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 2, pp. 14:1–14:39, 2010.

[60] M. E. Ruse, "Model checking techniques for vulnerability analysis of Web applications," Ph.D. dissertation, 2013. https://lib.dr.iastate.edu/etd/13211.

[61] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340, 2008.

[62] B. Korel and J. Laski, "Dynamic program slicing," *Information Processing Letters*, vol. 29, no. 3, pp. 155–163, 1988.

[63] D. Binkley, S. Danicic, T. Gyimóthy, M. Harman, A. Kiss, and B. Korel, "A formalisation of the relationship between forms of program slicing," *Science of Computer Programming*, vol. 62, no. 3, pp. 228–252, 2006.

[64] M. Harman, D. Binkley, and S. Danicic, "Amorphous program slicing," *Journal of Systems and Software*, vol. 68, no. 1, pp. 45–64, 2003.

[65] J. Zhao, "Multithreaded dependence graphs for concurrent java program," in *pdse*. IEEE, 1999, p. 13.

[66] J. Cheng, "Dependence analysis of parallel and distributed programs and its applications," in *Advances in Parallel and Distributed Computing, 1997. Proceedings*. IEEE, 1997, pp. 370–377.

[67] D. Goswami, R. Mall, and P. Chatterjee, "Static slicing in Unix process environment," *Softw., Pract. Exper.*, vol. 30, no. 1, pp. 17–36, 2000.

[68] T. Millstein, A. Levy, and M. Friedman, "Query Containment for Data Integration Systems," in *Proc. of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on PDS*. ACM, 2000, pp. 67–75.

[69] A. Y. Levy and Y. Sagiv, "Queries Independent of Updates," in *Proceedings of the 19th International Conf. on VLDB*. Morgan Kaufmann Publishers Inc., 1993, pp. 171–181.

[70] A. S. Christensen, A. Møller, and M. I. Schwartzbach, "Precise Analysis of String Expressions," in *Proceedings of the 10th International Conference on Static Analysis*, ser. SAS'03. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 1–18.

[71] A. Dasgupta, V. Narasayya, and M. Syamala, "A static analysis framework for database applications," in *IEEE International Conference on Data Engineering*. IEEE, 2009, pp. 1403–1414.

[72] G. Wassermann, C. Gould, Z. Su, and P. Devanbu, "Static Checking of Dynamically Generated Queries in Database Applications," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 4, 2007.

[73] M. S. Lam, J. Whaley, V. B. Livshits, M. C. Martin, D. Avots, M. Carbin, and C. Unkel, "Context-sensitive Program Analysis As Database Queries," in *Proceedings of the Twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '05. ACM, 2005, pp. 1–12.

[74] P. Mcminn, C. J. Wright, and G. M. Kapfhammer, "The Effectiveness of Test Coverage Criteria for Relational Database Schema Integrity Constraints," *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 1, pp. 8:1–8:49, 2015.

[75] G. Costantini, P. Ferrara, and A. Cortesi, "A Suite of Abstract Domains for Static Analysis of String Values," *Softw. Pract. Exper.*, vol. 45, no. 2, pp. 245–287, 2015.