Università
Ca'Foscari
Venezia

Dottorato di ricerca
in Informatica
Scuola di dottorato in Scienze e Tecnologie
Ciclo XXIV
(A.A. 2010 - 2011)

Extending Abstract Interpretation to New Applicative Scenarios

SETTORE SCIENTIFICO DISCIPLINARE DI AFFERENZA: INF/01
Tesi di dottorato di Raju Halder, matricola 955628

Coordinatore del Dottorato	**Tutore del dottorando**
Prof. Antonino Salibra	**Prof. Agostino Cortesi**

Extending Abstract Interpretation to New Applicative Scenarios

Raju Halder

Dipartimento di Scienze Ambientali, Informatica e Statistica

Università Ca' Foscari Venezia, Italy

Tutor: Prof. Agostino Cortesi

A thesis submitted for the degree of

Doctor of Philosophy (PhD)

2012 February

Abstract

The aim of this thesis is to extend the Abstract Interpretation framework to the broader context of Information Systems. In particular, we address issues related to security properties. We formalize a complete denotational semantics, both at concrete and abstract level, of data-intensive applications embedding data manipulation language operations such as SELECT, UPDATE, INSERT and DELETE. This theoretical work serves as a formal foundation of several interesting practical applications, including persistent watermarking, fine grained access control, SQL injection prevention, and cooperative query answering. We also address the issue of program slicing refinement, leading to an abstract program slicing algorithm that covers SQL data manipulation languages as well. A prototype of a tool implementing our abstract program slicing is also presented.

Dedication

The thesis is dedicated to my beloved parents **Mr. Subhash Chandra Halder** and **Mrs. Sandhya Halder**, who are a great source of motivation and inspiration of my life, and who sacrificed a lot for me to be what I am now. Without their loving care, prayers and support, it would have been very difficult for me to achieve my goals. I owe everything I have achieved or will achieve to them. I hope that by obtaining my PhD I can put smiles on their faces.

Acknowledgements

I am profoundly grateful to my advisor Prof. Agostino Cortesi who enlightens me through his wide knowledge on Program Analysis and Abstract Interpretation Theory. During the period, he helps me to see life and science in their full depth, and taught me how to appreciate the good scientific work that helps other researchers to build on it. Looking back, I am surprised and at the same time very grateful for all I have received from him that certainly shaped me as a person and has led me where I am now. All these years of PhD studies are full of such gifts.

I am heartily thankful to Dr. Francesco Logozzo and Prof. Letizia Tanca for giving their time and expertise to review the thesis and for their useful comments and suggestions.

My deep and sincere gratitude is devoted to Prof. Samar Sen Sarma, Prof. Partha Sarathi Dasgupta, Prof. Nabendu Chaki, Mr. Saptarshi Naskar and Mr. Krishnendu Basuli for their early inspiration and valuable guidance during my master degree that helps me to show a right potential and make a positive impact in the research works in subsequent periods of times.

I will always be indebted to all the people from "Technical Administrative Department", "Secretary office" and "International Relations Office" of our University. Special thanks goes to Laura Cappellesso, Giovanna Zamara, Fabrizio Romano, Gian-Luca Dei Rossi, Mery Sponchiado, Rossana Favaro and Sonia Barizza for their continuous support during the period. It was a pleasure to share doctoral studies and life with wonderful people like Matteo, Luca, Stefano, Andrea and others who are very close friends now.

Last but not least, I thank my parents and other family members for their unwavering love, support, and encouragement for my academic pursuits.

I finish with a final silence of gratitude for my life.

Contents

List of Figures

LIST OF FIGURES

List of Tables

Chapter 1

Introduction

Abstract Interpretation is a theory of sound approximation of mathematical structures, in particular those involved in the behavior of computer systems. It allows the systematic derivation of sound methods and algorithms for approximating undecidable or highly complex problems in various areas of computer science, for instance, in static program analysis, system verification, model checking, program transformation, process calculi, security, watermarking, type inference, theorem proving, constraint solving, parsing and comparative semantics, systems biology, etc. The era of Abstract Interpretation started about 30 years ago, in January 1977, with the pioneering work by Patrick Cousot and Radhia Cousot (47), and now it is widely applied to industrial software development tools (59). For a comprehensive overview of the main application scenarios of Abstract Interpretation, see (46).

Due to incessant growth of the amount of data, the information systems are facing serious challenges while managing, processing, analyzing, or understanding large volume of data in restricted environments. As a result of this, the performance of the systems in terms of optimization issues are really under big threat. The gap between the advances in information technology and the amount of data with which systems are dealing is a major concern for scientists now-a-days. To cope with this situation, in this thesis, we extend and integrate the well-established mathematical framework "Abstract Interpretation" to the context of Information Systems in order to achieve a sound approximation of the system. Interestingly, it supports a wide range of applications areas, in particular the security properties of information.

Although the Abstract Interpretation is routinely used to cope with infinite state systems, this thesis shows a suitable application of it to Information System scenarios that usually deal with finite systems. To the best of our knowledge, this thesis is the first work to create a bridge between Abstract Interpretation and Information Systems.

In particular, we formalize concrete and abstract semantics of database applications accessing and manipulating databases, by introducing the notion of *Abstract Databases*. Despite of the finiteness of the databases, the unpredictability in measuring their dimension in advance and the dynamism of the database states mandate the use of formal methods to work on safe approximations instead, in order to accomplish efficiency and effectiveness of the systems. The Abstract Interpretation formulation of database systems serves as a formal foundation of many interesting real-life applications, for instance, (*i*) to address the security properties, like watermarking and access control; (*ii*) to provide a novel cooperative query answering schema; (*iii*) to perform abstract slicing of applications accessing or manipulating databases, etc.

The main contributions of the thesis can be summarized as follows:

1. Abstract Interpretation of Database Query Languages.

   In the context of Relational Database Management Systems (RDBMS), there exist relational algebra and relational calculus as semantic descriptions for SQL. However, these semantic descriptions cover only a subset of SQL that are used for pure retrieval with no side effects on the database states (28, 30, 154). In particular, problems arise when dealing with UPDATE, INSERT or DELETE statements since operators originally proposed in relational algebra do not support them. Moreover, they do not cover the aggregate operators since these are still within the range of pure retrieval statements. This motivates our theoretical work aiming at defining a complete denotational semantics of applications embedding SQL statements, both at the concrete and at the abstract level, as a basis to develop an Abstract Interpretation for sound approximation of the applications. In this setting, we represent all the syntactic elements in SQL statements (for example, GROUP BY, ORDER BY, DISTINCT clauses, etc) as functions and the semantics is described as a partial functions on the states which specify how expressions are evaluated and instructions are executed. In order to define abstract states,

we introduce the notion of *Abstract Databases* obtained by abstracting concrete database information by suitable properties of interest in an abstract domain. We provide suitable examples of real life applications where abstraction of database systems plays important roles. As far as we know, the impact of abstract interpretation for sound approximation of database query languages has not yet been investigated. This is the aim of this work.

This content is explained in Chapter 3 and has already been published in (43, 79, 87).

2. Persistent Watermarking of Relational Databases.

   Digital watermarking is the process of embedding a piece of information into digital content which may be used later to verify its authenticity or the identity of its owners. The embedding of watermarks in the existing watermarking techniques (91) for relational databases are based on the database content itself. Benign Updates or any other authorized processing of this content may damage or distort the existing watermarks, leading the detection phase almost infeasible. In this work, we address the notion of persistent watermarking of relational databases that serves as a way to recognize the integrity and ownership proof of the database while allowing the evaluation of its content by a set of SQL statements Q associated with it. We propose a novel *fragile* and *robust persistent* watermarking scheme that embeds both *private* and *public* watermarks where the former allows the owner to prove his ownership, while the latter allows any end-user to verify the correctness and originality of the data in the database without loss of strength and security. The persistency of the watermark is preserved by exploiting (*i*) the concrete data belonging to the static part of the database states *w.r.t.* Q, (*ii*) the abstract representation of the database information in non-static part of the database states *w.r.t.* Q, and (*iii*) the invariants of the database information represented by its semantics-based properties.

   This content is explained in Chapter 4 and has already been published in (82, 83, 91).

3. Observation-based Fine Grained Access Control (OFGAC).

The granularity of the traditional access control mechanism is coarse-grained and can be applied only at database/table level for RDBMS or file/document level for XML. The use of Fine Grained Access Control (FGAC) mechanisms (54, 192), on the other hand, provide the safety of the database information even at lower level such as individual tuple/cell level for RDBMS or element/attribute level for XML without changing their original structure. However, in case of traditional FGAC, the notion of sensitivity of the information is too restrictive (either public or private) and impractical in some real systems where intensional leakage of the information to some extent is allowed with the assumption that the observational power of the external observers is bounded. For instance, suppose, according to the disclosure policy, that the employees of the customer-care section in a bank are able to see the last four digits of the credit card numbers, keeping the other digits completely hidden. The traditional FGAC policy is unable to implement this type of security framework without changing the database structure. To cope with this situation, we introduce the notion of Observation-based Fine Grained Access Control (OFGAC) mechanism for RDBMS and XML documents based on the Abstract Interpretation framework. In this setting, data are made accessible at various levels of abstractions based on their sensitivity levels. Unauthorized users are not able to infer the exact content of an attribute containing partial sensitive information, while they are allowed to get a partial or relaxed view of it, according to their access rights, represented by specific property.

This content is explained in Chapter 5 and has already been published in (81, 85, 86).

4. SQL Injection Attacks (SQLIA).

SQL injection is an attack in which malicious code is inserted into SQL statement and pass through a web application for execution by the backend database. If not prevented properly, web applications may result in SQL Injection attacks that allow hackers to view information from the database and/or even wipe it out. We address and propose a SQL injection prevention technique where we perform an obfuscation based analysis to prevent it. The proposed scheme has three phases, the first one is performed statically, while the latter two are performed dynamically: (*i*) Obfuscating the legitimate SQL statement Q into Q' at each hotspot of

the application, (*ii*) After merging the user input into the obfuscated statement at run-time, the dynamic verifier checks it at atomic formula level in order to detect the presence of possible SQLIA, and (*iii*) Reconstruction into the original statement Q from the obfuscated one Q' before submitting it to the database, if no possible SQLIA was detected. The proposal has many advantages: it reduces false-positiveness in case of structural inputs, it is application-independent, it reduces run-time over-head by distinguishing secure and vulnerable atomic formulas where verification is performed on vulnerable atomic formulas only, there is no need to use any secret key, etc.

This content is explained in Chapter 6 and has already been published in (80).

5. Cooperative Query Answering.

   The traditional query processing system requires the users to have precise information about the problem domain, database schema, and database content. It always provides limited and exact answers, or even no information at all when the exact information is not available in the database. To remedy such shortcomings, the notion of cooperative query answering has been explored as an effective mechanism that provides users an intelligent database interface to issue approximate queries independent to the underlying database structure and its content, and supplies additional useful information as well as the exact answers. We propose a cooperative query answering system based on the Abstract Interpretation framework. In this context, we address three key issues: soundness, relevancy and optimality of the cooperative answers, which can be used as a milestone to compare different cooperative schemes in the literature.

   This content is explained in Chapter 7 and has already been published in (84).

6. Refinement of Abstract Program Slicing techniques.

   Program slicing is a well-known decomposition technique that extracts from programs the statements which are relevant to a given behavior. It is a fundamental operation for addressing many software-engineering problems, including program understanding, debugging, maintenance, testing, parallelization, integration, software measurement etc. Many slicing techniques for imperative languages have been proposed based on the traditional Program Dependence

Graph (PDG) representation (2, 68, 96, 160). In traditional PDGs, the notion of dependences between statements is based on syntactic presence of a variable in the definition of another variable or in a conditional expression. Therefore, the semantic requirement of the slicing creates a gap between the slicing and the dependences. Mastroeni and Zanardini (140) first introduced the notion of semantic data dependences, both in concrete and abstract domains, that helps in converting traditional syntactic PDGs into more refined semantics-based (abstract) PDGs by disregarding some false dependences from them. As a result, the slicing techniques based on these semantics-based (abstract) PDGs result into more precise slices. In this work, we strictly improve this approach by (*i*) introducing the notion of semantic relevancy of statements, and (*ii*) combining it with conditional dependences (182). This allows us to transform syntactic PDGs into semantics-based (abstract) Dependence Condition Graphs (DCGs) that enable to identify the conditions for dependences between program points. These two contributions in combination with semantic data dependences lead to a refined abstract program slicing algorithm. Finally, we extend this approach to the slicing of data-intensive applications that interact with relational databases by exploiting the notion of database abstraction as discussed before.

This content is explained in Chapter 8 and has already been published in (44, 88).

7. **Tukra**: A Semantics-based Abstract Program Slicing Tool.

   We introduce **Tukra**, a semantics-based abstract program slicing tool, based on our proposal. This tool is implemented in Java. Given a program and an abstract slicing criterion from the user-end as input, **Tukra** is able to perform both syntax- and semantics-based abstract program slicing of a program *w.r.t.* the criterion. We provide its architecture, some snapshots to show how it works and some preliminary experimental results giving evidence towards its accuracy with respect to the literature.

   This content is explained in Chapter 9.

All the scenarios above draw an initial encouraging direction towards the application of Abstract Interpretation theory to the context of Information Systems, giving rise to a formal support to sound management of abstraction of data. We show that this

approach is effective on a suite of non-trivial issues, like the ones concerning security properties.

Chapter 2

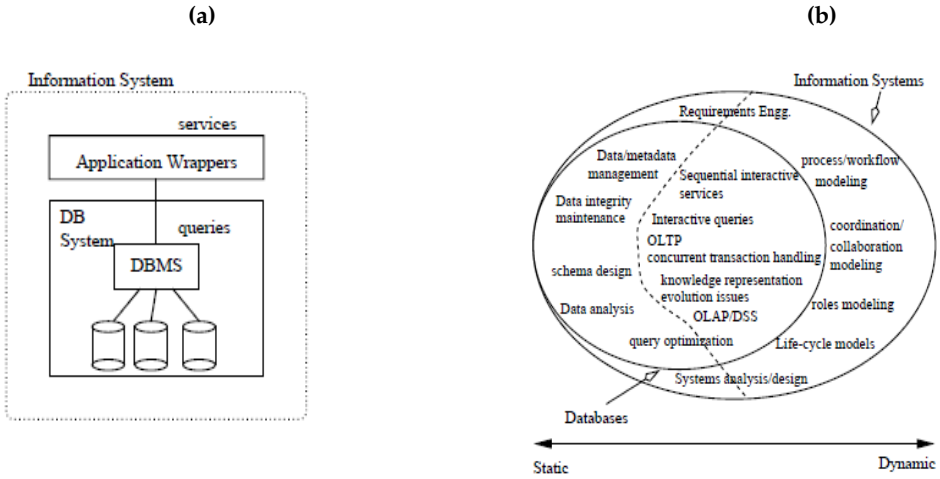
Background: Information Systems, Databases, Abstract Interpretation

In this chapter, we recall from (47, 48, 49, 55, 56, 64, 65, 66, 176) some basic background on Information Systems, Databases, and Abstract Interpretation theory that will be useful in the rest of the thesis.

Information Systems (IS) (65, 66, 176) is a generic term referring to software and hardware systems that support data-intensive applications and provide information-based services. In the past, organizations recognized the importance of managing resources such as labor, capital, and raw materials. Today, it is widely accepted that managing the information resource is very often equally important. The notion of information systems has been addressed from various perspectives like managerial, technical, organizational etc.

Information systems for most organizations are currently supported by databases. As a consequence, information system design addresses a number of database issues like conceptual modeling of data elements, metadata management, etc. On the other hand, any useful database system is actually an information system providing additional services beyond simply storing data and running queries and updates. As a result, the distinction between a database and an information system tends to be blurred, and it is not clear that the principles underlining the study of information systems should be different than those for databases. The distinction between a database that manages data and answers queries, versus an information system that provides services, is schematically depicted in Figure 2.1. Figure 2.1(a) shows information sys-

Figure 2.1: Information Systems Vs. Database Systems (66)



tems as wrappers above database systems, while Figure 2.1(b) shows issues of concern from a database and from an information system perspective. For more detail, please refer (65, 66).

Information systems that incorporate multiple technologies and processes must be designed and developed according to rigorous engineering standards to ensure that they support the requirements of their respective application domains and that they operate rapidly, accurately, and efficiently. Today, an exponentially increasing amount of data is available for processing and analysis, the number of decisions that must be made based on analysis is growing, the number of analysts that can make these decisions is decreasing, the time frame to make the decisions is becoming smaller, the size of information systems that support decision systems is increasing, and information systems are becoming more vulnerable to attack. As the volume and heterogeneity of data play an increasingly prominent role in information systems, and as they continue to proliferate among many disparate organizations, advanced intelligent techniques must be exploited to simplify access to the data, accelerate data integration, and extract higher-level meaning from their content (176).

Data sources and their respective data management systems store and maintain information relevant to information systems. Data source representations are generally categorized into three: (*i*) structured (e.g., relational databases), (*ii*) semi-structured (e.g., XML documents), and (*iii*) unstructured (e.g., text documents). Information systems incorporating new or legacy data sources of all three categories will use tools

that provide simplified access to them and enable them to be integrated with other data sources and applications.

Database Terminologies

A database (55, 56) is a collection of related data. This definition of database is quite general; however, it has the following implicit properties:

- A database represents some aspect of the real world, sometimes called the miniworld or the universe of discourse (UoD). Changes to the miniworld are reflected in the database.
- A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.
- A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.

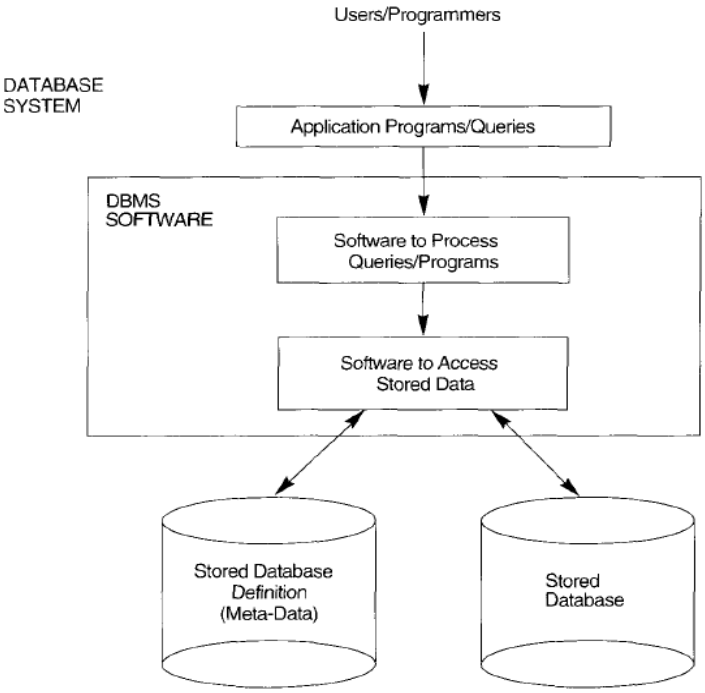
A database management system (DBMS) is a collection of programs that enables users to create and maintain a database. The DBMS is hence a general-purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing databases among various users and applications.

As a complete definitions, we call the database and DBMS software together a database system, depicted in Figure 2.2.

One fundamental characteristic of the database approach is that it provides some level of data abstraction by hiding details of data storage that are not needed by most database users. A data model is a collection of concepts that can be used to describe the structure (*i.e.*, data types, relationships, and constraints that should hold for the data) of a database that provides the necessary means to achieve this abstraction.

The DBMS provides a set of operations or a language called the data manipulation language (DML) for retrieval, insertion, deletion, and modification of the data. In current DBMSs, the preceding types of languages are usually not considered distinct languages; rather, a comprehensive integrated language is used that includes constructs for conceptual schema definition, view definition, and data manipulation. A typical example of a comprehensive database language is the SQL relational database

Figure 2.2: A simplified database system environment (56)



language, which represents a combination of data definition language (DDL), view definition language (VDL), and data manipulation language (DML), as well as statements for constraint specification, schema evolution, and other features. The name SQL is derived from Structured Query Language. Originally, SQL was called SEQUEL (for Structured English QUEry Language) and was designed and implemented at IBM Research as the interface for an experimental relational database system called SYSTEM R. SQL is now considered as the standard language and already implemented as interfaces to many commercial relational DBMSs, including Oracle, IBM's DB2 and SQL/DS, Microsoft's SQL Server and ACCESS, INGRES, INFORMIX, and SYBASE. Table 2.1 summarizes the syntax of various SQL statements. This summary is not meant to be comprehensive nor to describe every possible SQL construct; rather, it is meant to serve as a quick reference to the major types of constructs available in SQL, and it will be our references for the rest of our thesis.

Several techniques exist for including database interactions in application programs. The main approaches for database programming are the following:

1. Embedding database statements in a general-purpose programming language:

Table 2.1: Summary of SQL Syntax

```
SELECT [DISTINCT] < attribute list >
FROM (< table name > {< alias >} | < joined table >) {, (< table name > {< alias >} | < joined table >)}
[WHERE < condition >]
[GROUP BY < grouping attributes > [HAVING < group selection condition >]]
[ORDER BY < column name > [< order >] {, < columnname > | [< order >] } ]

< attribute list > ::= (* | (< column name > | < function > (([DISTINCT] < column name >) | *))
                        {, (< column name > | < function > (([DISTINCT] < column name >) | *))})
< grouping attributes > ::= < column name > {, < column name >}
< order > ::= (ASC | DESC)

INSERT INTO < table name > [(< columnname > {, < column name >})]
(VALUES (< constant value > {, < constant value >}){, (< constantvalue > {, < constantvalue >})}| < select statement >)

DELETE FROM < table name >
[WHERE < selection condition >]

UPDATE < table name >
SET < column name >=< value expression > {, < column name >=< value expression >}
[WHERE < selection condition >]
```

In this approach, database statements are embedded into the host programming language, but they are identified by a special prefix. For example, the prefix for embedded SQL is the string EXEC SQL, which precedes all SQL statements in a host language program.

2. Using a library of database functions: A library of functions is made available to the host programming language for database calls. For example, the functions to connect to a database, to execute SQL statements, and so on. This approach provides an Application Programming Interface (API) for accessing a database from application programs.

3. Designing a brand-new language: A database programming language is designed from scratch to be compatible with the database model and query language. Additional programming structures such as loops and conditional statements are added to the database language to convert it into a full-fledged programming language.

In practice, the first two approaches are more common, whereas the third approach is more appropriate for applications that have intensive database interaction.

Abstract Interpretation

Abstract interpretation (47, 48, 49, 64) formalizes the correspondence between the concrete semantics $S^c[\![P]\!]$ of syntactically correct program $P \in \mathbb{P}$ in a given programming language $\mathbb{P}$ and its abstract semantics $S^a[\![P]\!]$ which is a safe approximation of the concrete semantics $S^c[\![P]\!]$.

The concrete and abstract semantics domain $\mathfrak{D}^c$ and $\mathfrak{D}^a$ respectively often enjoy stronger properties, such as being complete partial orderings (CPO) or complete lattices.

> *A CPO is a poset $(D, \sqsubseteq)$ where the set D is equipped with an ordering relation $\sqsubseteq$ and satisfies the following property: every sequence of elements $x_0 \sqsubseteq x_1 \sqsubseteq ... \sqsubseteq x_n$ in D has a limit or least upper bound in D, that is, there is an element in $x \in D$ (written as $\bigsqcup_i x_i$) such that, (i) $\forall x_i,\ x_i \sqsubseteq x$, (ii) if x' is any other upper bound for the x_i, then $x \sqsubseteq x'$.*

> *A poset $(D, \sqsubseteq)$ is called a complete lattice (denoted $\langle D, \sqsubseteq, \sqcup, \sqcap, \top, \bot \rangle$) if every subset S of D has a least upper bound (written $\sqcup S$) and a greatest lower bound (written $\sqcap S$) in it, and it is bounded by a unique largest element $\top = \sqcup D$ and a unique smallest element $\bot = \sqcap D$.*

The correspondence between these two concrete and abstract semantics domains $\mathfrak{D}^c$ and $\mathfrak{D}^a$ form a Galois Connection $(\mathfrak{D}^c, \alpha, \gamma, \mathfrak{D}^a)$, where the function $\alpha : \mathfrak{D}^c \longrightarrow \mathfrak{D}^a$ and $\gamma : \mathfrak{D}^a \longrightarrow \mathfrak{D}^c$ form an adjunction, namely $\forall A \in \mathfrak{D}^a, \forall C \in \mathfrak{D}^c : \alpha(C) \sqsubseteq^a A \Leftrightarrow C \sqsubseteq^c \gamma(A)$ where $\alpha(\gamma)$ is the left(right) adjoint of $\gamma(\alpha)$. α and γ are called abstraction and concretization maps respectively.

An Upper Closure Operator (UCO) on C, or simply a Closure, is an operator $\rho : C \to C$ which is monotone, idempotent, and extensive. It maps the concrete values with their abstract properties, namely with the best possible approximation of the concrete value in the abstract domain. For example, the operator $PAR : \wp(Z) \to \wp(Z)$ associates each subset of integers with its parity, $PAR(\bot) = \bot$, $PAR(S) = EVEN = \{n \in Z \mid n \text{ is even}\}$ if $\forall n \in S$. n is even, $PAR(S) = ODD = \{n \in Z \mid n \text{ is odd}\}$ if $\forall n \in S$. n is odd, and $PAR(S) = I\,don't\,know = Z$ otherwise. We denote by $UCO(C)$ the set of closures on C. When C is a complete lattice then $\langle UCO(C), \sqsubseteq, \sqcap, \sqcup, \lambda x.\top, \lambda x.x \rangle$ is a complete lattice as well, where $\rho_1 \sqsubseteq \rho_2$ if and only if $\rho_1(C) \subseteq \rho_2(C)$, meaning that the abstract domain

specified by ρ_1 is more precise than the abstract domain specified by ρ_2. Let us recall that each closure ρ is uniquely determined by the set of its fixpoints, given by its image $\rho(C)$. A set $X \subseteq \rho(C)$ is the set of fixpoints of a closure operator if and only if X is a Moore family of C, i.e., $X = M(X) = \{\sqcap S | S \subseteq X\}$, where $\sqcap\emptyset = \top \in M(X)$. Given a Galois connection (C, α, γ, A), $\rho = \gamma \circ \alpha$ is the closure corresponding to the abstract domain A.

Let (C, α, γ, A) be a Galois connection, $f : C \to C$ be a concrete function and $f^\sharp : A \to A$ be an abstract function. $f^\sharp$ is a sound, i.e., correct approximation of f if $f \circ \gamma \sqsubseteq \gamma \circ f^\sharp$. When the soundness condition is strengthened to equality, i.e., when $f \circ \gamma = \gamma \circ f^\sharp$, the abstract function $f^\sharp$ is a complete approximation of f in A. This means that no loss of precision is accumulated in the abstract computation through $f^\sharp$. Given $A \in UCO(C)$ and a semantic function $f : C \to C$, the notation $f^A \triangleq \alpha \circ f \circ \gamma$ denotes the best correct approximation of f in A. It has been proved that, given an abstraction A, there exists a complete approximation of $f : C \to C$ in A if and only if the best correct approximation f^A is complete (64). This means that completeness of f^A is an abstract domain property, namely that it depends on the structure of the abstract domain only.

As a first approximation, abstract interpretation can be understood as a nonstandard semantics, i.e., one in which the domain of values is replaced by a domain of descriptions of values, and in which the operators are given a corresponding nonstandard interpretation. For example, rather than using integers as concrete values, an abstract interpretation may uses four different abstract values "*zero*", "*pos*", "*neg*", and "*num*", where "*zero*" indicates that the number is zero, "*pos*" that the number is positive, "*neg*" that the number is negative, and "*num*" that we don't know. We will call the set of these values "*Sign*". The rule-of-sign interpretation of addition and multiplication can then be specified in two tables as depicted in Table 2.2. These interpretations may be viewed as "abstract" additions and multiplications.

Table 2.2: Abstract additions and multiplications operations

$\oplus$: **Sign** × **Sign** → **Sign**

$\oplus$	zero	pos	neg	num
zero	zero	pos	neg	num
pos	pos	pos	num	num
neg	neg	num	neg	num
num	num	num	num	num

$\otimes$: **Sign** × **Sign** → **Sign**

$\otimes$	zero	pos	neg	num
zero	zero	zero	zero	zero
pos	zero	pos	neg	num
neg	zero	neg	pos	num
num	zero	num	num	num

Examples of other abstract domains include the domain of parity, the domain of intervals etc. Observe that the abstract domains just mentioned do not take into account any relationships between variables, and is thus a non-relational abstract domain. Non-relational abstract domains tend to be fast and simple to implement, but imprecise. Some examples of relational numerical abstract domains are:

- congruence relations on integers
- convex polyhedra with some high computational costs
- octagons
- difference-bound matrices
- linear equalities

and combinations thereof.

Chapter 3

Abstract Interpretation of Database Query Languages

[Part of this chapter is already published in (43, 79, 87)]

In the context of web-based services interacting with DBMS, there is a need of "sound approximation" of database query languages, in order to minimize the weight of database replicas on the web or in order to hide specific data values while giving them public access with larger granularity. There are many application areas where processing of database information at different level of abstraction plays important roles, like the applications where users are interested only in the query answers based on some properties of the database information rather than their exact values.

Given an exploratory nature of the applications, like decision support system, experiment management system etc, many of the queries end up producing no result of particular interest to the user. Wasted time can be saved if users are able to quickly see an approximate answer to their query, and only proceed with the complete execution if the approximate answer indicates something interesting. The sound approximation of the database and its query languages may also serve as a formal foundation of answering queries approximately as a way to reduce query response times, when the precise answer is not necessary or early feedback is helpful (108).

When a database is being populated with tuples, all tuples must satisfy some properties which are represented in terms of integrity constraints. For instance, the ages of the employees must be positive and must lie between 18 and 62. Any transaction

over the database must satisfy all these integrity constraints as well. The dynamic checking for any transaction to ensure whether it violates the integrity constraints of the database can increase the run-time overhead significantly, while managing the integrity constraint verification statically may have a significant impact in terms of efficiency. An interesting solution to this problem can be provided by extending to the database field the Abstract Interpretation framework (47, 48, 49, 64).

Relational databases enjoy mathematical formulations that yield to a semantic description using formal language like relational algebra or relational calculus. To handle the aggregate functions or NULL values, some extensions of existing relational algebra and relational calculus have been introduced (28, 117, 153, 154). However, this semantic description covers only a subset of SQL that are used for pure retrieval with no side effects on the database states (28, 30, 154). In particular, problems arise when dealing with UPDATE, INSERT or DELETE statements since operators originally proposed in relational algebra do not support them. This motivates our theoretical work aiming at defining a complete denotational semantics of programs embedding SQL statements, both at concrete and abstract level, as a basis to develop an Abstract Interpretation for sound approximation of relational database systems. This way, the semantics of SQL can be tuned according to suitable abstractions of the concrete domain of data. In this setting, we represent all the syntactic elements in SQL statements (for example, GROUP BY, ORDER BY, DISTINCT clauses, etc.) as functions and the semantics is described as a partial functions on states which specify how expressions are evaluated and instructions are executed. The functional representation of syntactic elements increases the power of expressibility of the semantics and facilitates us to provide a complete functional control on the corresponding domains of data. As far as we know, the impact of abstract interpretation for sound approximation of database query languages, with a notable exception in case of graphical query languages like G-Log (42), has not yet been investigated. This is the aim of this chapter.

The underlying concepts is that the applications embedding SQL code basically interact with two worlds or environments: *user world* and *database world*. Corresponding to these two worlds or environments we define two sets of variables: $\mathbb{V}_d$ and $\mathbb{V}_a$. The set $\mathbb{V}_d$ is the set of database variables (*i.e.*, the set of database attributes) and $\mathbb{V}_a$ is a distinct set of variables called application variables defined in the application. Variables from $\mathbb{V}_d$ are involved only in the SQL statements, whereas variables in $\mathbb{V}_a$ may

occur in all types of instructions of the application. We denote any SQL statement by a tuple $Q_{sql} = \langle A_{sql}, \phi \rangle$. We call the first component A_{sql} the *action part* and the second component ϕ the *pre-condition part* of Q_{sql}. In an abstract sense, any SQL statement Q_{sql} first identifies an active data set from the database using the pre-condition ϕ and then performs the appropriate operations on that data set using the SQL action A_{sql}. The pre-condition ϕ appears in Q_{sql} as a well-formed formula in first-order logic. The semantics defined this way can be lifted from the concrete domain of values to abstract representation of them by providing suitable abstract operators corresponding to the concrete ones.

The structure of this chapter is as follows: Section 3.1 discusses the related work in the literature. Section 3.2 recalls some preliminary concepts. Section 3.3 defines the abstract syntax of programs embedding SQL statements. In section 3.4, we define environments and states associated with the application. Section 3.5 describes the semantics of the arithmetic and boolean expressions, whereas section 3.6 and 3.7 describe the formal semantics of atomic and composite statements respectively. In section 3.8, the correspondence of the proposed denotational semantic approach with the relational algebra is discussed. In section 3.9, we lift the syntax and semantics of the query languages from concrete domain to an abstract domain of interest by discussing the soundness and completeness of the abstraction in details. In section 3.10, we discuss the formal semantics of SQL statements with correlated and non-correlated subquery.

3.1 Related Works

Most popular commercial and open source databases currently in use are based on the relational data model which serves as a formal basis for relational database system. The relational model of data was first proposed by E. F. Codd in 1970 (41). In (40), E. F. Codd defined a collections of operations on relations which is defined as Relational Algebra.

Relational Calculus is based on a branch of mathematical logic called predicate calculus. In 1967, possibly, Kuhns first used the idea of predicate calculus as the basis for query language in (125). The applied form of predicate calculus specifically tailored

to relational databases was proposed by E. F. Codd (40). A language explicitly based on that calculus called "*Data Sublanguage ALPHA*" was also presented by Codd in (39).

The relational algebra and relational calculus provide a foundation for designing relationally complete query languages. According to Codd's theorem, the relational algebra and the relational calculus are essentially logically equivalent: for any algebraic expression, there is an equivalent expression in the calculus, and vice versa. Codd's reduction algorithm (40) demonstrates that the relational algebra has at least the selective power of the relational calculus. Relational calculus provides a declarative way to specify database queries, whereas relational algebra provides a more procedural way for specifying queries.

In (154), the authors described the formal semantics of SQL using a formal model, called Extended Three Valued Predicate Calculus (E3PVC). This model is basically based on a set of rules that determine a syntax-driven translation of SQL queries. These rules allow the transformation of a general E3VPC expression to a canonical form, which can be manipulated using traditional, two-valued predicate calculus and solves the equivalence of SQL queries.

The Relational Algebra and Relational Calculus can be used as a model for designing many approaches to query optimization. Many works (28, 30, 111, 153) on semantics and optimization of SQL queries focused on the approach of translating SQL to a formal language.

Bültzingwloewen (28) provided a precise definition of the semantics of SQL queries having aggregate functions and identified some problems associated with optimization along with their solutions. Here the semantics is defined by translating SQL queries into extended relational calculus expressions based on the work in (117). Moreover, the NULL values are considered in the extended version of relational algebra and relational calculus. It has been proved that these extended relational calculus and relational algebra are equivalent and have the same expressive power.

In (30), the authors proposed a syntax directed translator of SQL into relational algebra. This is done in two steps: transforming SQL queries into equivalent SQL queries accepted by restricted grammar, and then transforming the restricted one into relational algebra expression. The translator can be in conjunction with an optimizer which operates on the relational algebra expression. This translator defines the semantics of the SQL language and can be used for the proof of equivalence of SQL

queries with different syntactic form. Translation of SQL into equivalent relational algebra via relational calculus is also presented in (111). However, this translation is not optimized.

Fraternali and Tanca (60) defined the semantics of active databases. First, they defined an extended event-condition-action (EECA) rule by encoding the whole variety of behaviors of active databases into a unique formalism in user-readable format. A rule from any existing system, such as Oracle, Sybase, SQL3, Postgres etc, can be rewritten in this formalism making all the semantic choices apparent. Then, a syntax-directed translation from EECA rules to a low-level logic-based core rules is performed. The semantics of core rules is specified as the fixpoint of a transformation described by a simple rule execution algorithm. Finally, the core rule execution model is embedded into a more general framework for transactions and updates in active databases.

In (10, 11, 12), the authors introduced the way to optimize integrity constraints checking for a transaction at compile-time to reduce the run-time overhead. They considered only the object-oriented databases where the initial databases are represented in the form of first order logic formulas (treated as abstract form of databases). They used predicate transformer as a way to provide the abstract interpretation of the transactions so as to collect the run-time behavior of the transaction at compile-time.

3.2 Preliminaries

In this section, we recall some basic mathematical notation used in the literature and some basic ideas about semantic interpretation of well-formed formulas in first-order logic (67).

Basic Mathematical Notation

If S and T are sets, then $\wp(S)$ denotes the powerset of S, $|S|$ the cardinality of S, $S \backslash T$ the set-difference between S and T, $S \times T$ the Cartesian product. A poset P with ordering relation $\sqsubseteq$ is denoted as $\langle P, \sqsubseteq \rangle$, while $\langle C, \sqsubseteq, \sqcup, \sqcap, \top, \bot \rangle$ denotes the complete lattice C with ordering $\sqsubseteq$, lub $\sqcup$, glb $\sqcap$, greatest element $\top$, and least element $\bot$.

Let e, t and f be an expression, a database table and a function respectively. We use the following functions in the subsequent section:

- *const(e)* returns the constants appearing in e.
- *var(e)* returns the variables appearing in e.
- *attr(t)* returns the attributes associated with t.
- *dom(f)* returns the domain of f.
- *target(f)* returns a subset of dom(f) on which the application of f is restricted.

Semantic Interpretation of Well-Formed Formulas in First-Order Language

Let F, R and C be the set of function symbols, relation symbols and constant symbols respectively in the first order language L. A semantic structure ς for L is a non-empty set D_ς, called the domain of the structure, along with the following:

1. For each function symbol $f_{n,m} \in F$, there is a function $f^\varsigma_{n,m} : D^n_\varsigma \to D_\varsigma$;
2. For each relation symbol $R_{n,m} \in R$, there is a subset $R^\varsigma_{n,m}$ of D^n_ς;
3. For each constant symbol $c_k \in C$, there is an element c^ς_k.

The subscript n in the notation $f_{n,m}$ and $R_{n,m}$ gives the number of arguments of the corresponding function or relation, whereas subscript m says it is the m^{th} of the symbols requiring n arguments. In general, a subset of D^n_ς is called an n-place relation on D_ς, so that the subsets $R^\varsigma_{n,m}$ are just described as the relations on ς, and the c^ς_ks are called constants of ς. The functions $f^\varsigma_{n,m} \in F_\varsigma$, relations $R^\varsigma_{n,m} \in R_\varsigma$ and constants $c^\varsigma_k \in C_\varsigma$ are called the interpretations in the structure ς of the corresponding symbols of L.

We shall often write semantic structures using the notation, $\varsigma = \{D_\varsigma, C_\varsigma, F_\varsigma, R_\varsigma\}$. Let L be a language with equality. A structure for L is said to be normal if the interpretation of = is equality on its domain.

Suppose that the variables $x_1,\ x_2, ...,\ x_n$ are interpreted respectively by elements $a_1,\ a_2, ...,\ a_n$ of D_ς. We shall abbreviate this interpretation by $\vec{a}/\vec{x}$. Then the interpretation in ς of each term $\tau \in \mathbb{T}$ of the first-order language L under this interpretation of the variables, which we write as $\tau[\vec{a}/\vec{x}]^\varsigma$, is defined recursively as follows:

- For each variable x_i, we define $x_i[\vec{a}/\vec{x}]^\varsigma = a_i$.
- For each constant symbol c_k, we define $c_k[\vec{a}/\vec{x}]^\varsigma = c^\varsigma_k$.
- If $f_{n,m}$ is a function symbol in first-order language L and $\tau_1,\ \tau_2, \ldots,\ \tau_n \in \mathbb{T}$, then $f_{n,m}(\tau_1,\ \tau_2, \ldots,\ \tau_n)[\vec{a}/\vec{x}]^\varsigma = f^\varsigma_{n,m}(\tau_1[\vec{a}/\vec{x}]^\varsigma, \ldots,\ \tau_n[\vec{a}/\vec{x}]^\varsigma)$.

Now let ϕ be a well-formed-formula of L. The relation $\varsigma \models \phi[\vec{a}/\vec{x}]$ is read as "the formula ϕ is true in, or is satisfied by, the structure ς when $x_1, x_2, \ldots, x_n$ are interpreted by $a_1, a_2, \ldots, a_n$". This is defined recursively on the construction of ϕ as follows:

- Atomic formulas:
  (a) for each relation symbol $R_{n,m}$ in L and terms $\tau_1, \tau_2, \ldots, \tau_n$: $\varsigma \models R_{n,m}(\tau_1, \tau_2, \ldots, \tau_n)[\vec{a}/\vec{x}]$ if and only if $(\tau_1[\vec{a}/\vec{x}]^\varsigma, \ldots, \tau_n[\vec{a}/\vec{x}]^\varsigma) \in R^\varsigma_{n,m}$
  (b) if $\tau_1 = \tau_2$ are terms, then $\varsigma \models (\tau_1 = \tau_2)[\vec{a}/\vec{x}]$ if and only if $\tau_1[\vec{a}/\vec{x}]^\varsigma = \tau_2[\vec{a}/\vec{x}]^\varsigma$
- For any formula of one of the forms $\neg\phi$, $\phi_1 \vee \phi_2$, $\phi_1 \wedge \phi_2$, $\forall x_i\ \phi$, $\exists x_i\ \phi$ truth tables laws are followed, e.g.
  (a) $\varsigma \models (\neg\phi)[\vec{a}/\vec{x}]$ if and only if it is not the case that $\varsigma \models \phi[\vec{a}/\vec{x}]$
  (b) $\varsigma \models (\phi_1 \vee \phi_2)[\vec{a}/\vec{x}]$ if and only if $\varsigma \models \phi_1[\vec{a}/\vec{x}]$ or $\varsigma \models \phi_2[\vec{a}/\vec{x}]$.
  (c) $\varsigma \models (\phi_1 \wedge \phi_2)[\vec{a}/\vec{x}]$ if and only if $\varsigma \models \phi_1[\vec{a}/\vec{x}]$ and $\varsigma \models \phi_2[\vec{a}/\vec{x}]$.
  (d) $\varsigma \models (\forall x_i\ \phi)[\vec{a}/\vec{x}]$ if and only if for all $b \in D_\varsigma$, $\varsigma \models \phi[\vec{a}/\vec{x}[b/x_i]]$.
  (e) $\varsigma \models (\exists x_i\ \phi)[\vec{a}/\vec{x}]$ if and only if there is some $b \in D_\varsigma$, $\varsigma \models \phi[\vec{a}/\vec{x}[b/x_i]]$.

3.3 Abstract Syntax of Programs embedding SQL Statements

The abstract syntax of the application programs embedding SQL statements is depicted in Table 3.1. It is based on the following syntactic sets:

$n : \mathbb{Z}$	Integer
$k : \mathbb{S}$	String
$c : \mathbb{C}$	Constants
$v_a : \mathbb{V}_a$	Application Variables
$v_d : \mathbb{V}_d$	Database Variables (attributes) involved in SQL statements
$v : \mathbb{V} \triangleq \mathbb{V}_d \cup \mathbb{V}_a$	Variables
$e : \mathbb{E}$	Arithmetic Expressions
$b : \mathbb{B}$	Boolean Expressions
$A_{sql} : \mathbb{A}_{sql}$	Action Part of SQL statements
$\tau : \mathbb{T}$	Terms
$a_f : \mathbb{A}_f$	Atomic Formulas
$\phi : \mathbb{W}$	Well-formed formulas (Pre-condition part of SQL statements)
$Q_{sql} : \mathbb{Q}_{sql}$	SQL statements
$I : \mathbb{I}$	Instructions

Table 3.1: Abstract syntax of the application programs embedding SQL statements

$c ::=$	$n \mid k$
$e ::=$	$c \mid v_d \mid v_a \mid op_u\ e \mid e_1\ op_b\ e_2$, where op_u and op_b represent unary and binary arithmetic operators respectively.
$b ::=$	$e_1 = e_2 \mid e_1 \geq e_2 \mid e_1 \leq e_2 \mid e_1 > e_2 \mid e_1 < e_2 \mid e_1 \neq e_2 \mid \neg b \mid b_1 \vee b_2 \mid b_1 \wedge b_2 \mid true \mid false$
$\tau ::=$	$c \mid v_a \mid v_d \mid f_n(\tau_1, \tau_2, ..., \tau_n)$, where f_n is an n-ary function.
$a_f ::=$	$R_n(\tau_1, \tau_2, ..., \tau_n) \mid \tau_1 = \tau_2$, where R_n is an n-ary relation: $R_n(\tau_1, \tau_2, ..., \tau_n) \in \{true, false\}$
$\phi ::=$	$a_f \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \forall x_i\ \phi \mid \exists x_i\ \phi$
$g(\vec{e}) ::=$	`GROUP BY`$(\vec{e}) \mid id$
$r ::=$	`DISTINCT` | `ALL`
$s ::=$	`AVG` | `SUM` | `MAX` | `MIN` | `COUNT`
$h(e) ::=$	$s \circ r(e) \mid$ `DISTINCT`$(e) \mid id$
$h(*) ::=$	`COUNT`(*)
$\vec{h}(\vec{x}) ::=$	$\langle h_1(x_1), ..., h_n(x_n)\rangle$, where $\vec{h} = \langle h_1, ..., h_n\rangle$ and $\vec{x} = \langle x_1, ..., x_n\rangle$
$f(\vec{e}) ::=$	`ORDER BY ASC`$(\vec{e}) \mid$ `ORDER BY DESC`$(\vec{e}) \mid id$
$A_{sql} ::=$	$select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi,\ g(\vec{e})) \mid update(\vec{v_d},\ \vec{e}) \mid insert(\vec{v_d},\ \vec{e}) \mid delete(\vec{v_d})$
$Q_{sql} ::=$	$\langle A_{sql}, \phi\rangle \mid Q'_{sql}$ `UNION` $Q''_{sql} \mid Q'_{sql}$ `INTERSECT` $Q''_{sql} \mid Q'_{sql}$ `MINUS` Q''_{sql}
$I ::=$	$skip \mid v_a := e \mid v_a :=? \mid Q_{sql} \mid if\ b\ then\ I_1\ else\ I_2 \mid while\ b\ do\ I \mid I_1; I_2$

Any constant $c \in \mathbb{C}$ appearing in SQL statements Q_{sql} is either an integer $n \in \mathbb{Z}$ or a string $k \in \mathbb{S}$. The *pre-condition* ϕ of Q_{sql} is a well-formed formula in first order logic. We deal with only Data Manipulation Language (DML) for the *action part* A_{sql}, that is, an SQL action is the application of either SELECT, or UPDATE, or INSERT, or DELETE. Observe that the database variables from the set $\mathbb{V}_d$ can appear in Q_{sql} only. Since the variables from $\mathbb{V}_d$ represent the attributes of the database tables, we assume that no two tables have the same attributes.

The function `GROUP BY`$(\vec{e})[t]$ where $\vec{e}$ represents an ordered sequence of arithmetic

expressions, is applied on a table t and depending on the values of $\vec{e}$ over the tuples of t, it results into maximal partition of the tuples of t. The functions `ORDER BY ASC`$(\vec{e})[t]$ and `ORDER BY DESC`$(\vec{e})[t]$ sort the tuples of table t in ascending or descending order based on the value of $\vec{e}$ over the tuples in t respectively. Observe that, A_{sql} of SELECT statement may or may not use `GROUP BY` and `ORDER BY` functions, and this fact is reflected in the abstract syntax of g and f respectively in Table 3.1.

The aggregate functions in SELECT statement are represented by s. The clauses `DISTINCT` and `ALL` are used to deal with duplicate values. We denote `DISTINCT` and `ALL` by the function r. By $\vec{h}(\vec{x})$, we denote an ordered sequence of functions operating on an ordered sequence of arguments $\vec{x}$, *i.e.*, each function $h_i \in \vec{h}$ operates on the corresponding argument $x_i \in \vec{x}$. The argument x_i is an expression e or a sequence of all attributes of the table denoted by $*$ in SQL.

It should be noted that, if SELECT statement uses `GROUP BY`$(\vec{e})$, then there must be an $\vec{h}(\vec{x})$ which is evaluated on each partition obtained by `GROUP BY` operation, yielding to a single tuple. In such case, the i^{th} element $h_i \in \vec{h}$ must be `DISTINCT` function if the corresponding i^{th} element of $\vec{x}$ (i.e. x_i) belongs to $\vec{x} \cap \vec{e}$, or h_i must be `COUNT` if x_i is $*$, otherwise $h_i(x_i)$ is $s \circ r(e)$ where $e \in \vec{x} \wedge e \notin (\vec{x} \cap \vec{e})$. That is,

$$h_i \triangleq \begin{cases} \texttt{COUNT} \quad \text{if } x_i = * \text{ or,} \\ \texttt{DISTINCT} \quad \text{if } x_i \in \vec{x} \cap \vec{e} \text{ or,} \\ s \circ r \quad \text{otherwise} \end{cases}$$

When the SELECT statement does not use any `GROUP BY`$(\vec{e})$ function, we have two situations: (i) if $\vec{h} \neq \vec{id}$, then the set of all tuples in the table for which ϕ satisfies is considered as a single group and $\vec{h}(\vec{x})$ is evaluated on that group. In that case, $h_i \in \vec{h}$ is defined as follows:

$$h_i \triangleq \begin{cases} \texttt{COUNT} \quad \text{if } x_i = * \text{ or,} \\ s \circ r \quad \text{otherwise} \end{cases}$$

(ii) if $\vec{h} = \vec{id}$, then each tuple in the table for which ϕ satisfies is considered as an individual group and $\vec{h}(\vec{x}) = \vec{x}$ is evaluated on each of these groups.

Note that, the function r involved in $h_i \in \vec{h}$ deals with duplicate values of the argument expression e, whereas the function r in $r(\vec{h}(\vec{x}))$ occurring in the action part A_{sql} of SELECT statement deals with duplicate results obtained after performing $\vec{h}$ over the group(s).

The formula ϕ and the variable v_a appearing in A_{sql} of the SELECT statement represent the `HAVING` clause and a Record/ResultSet type application variable respectively. v_a has an ordered sequence of fields $\vec{w}$ where the type of each field $w_i \in \vec{w}$ is the same as the return type of the corresponding function $h_i(x_i) \in \vec{h}(\vec{x})$. By the vector notation $\vec{v_d}$, we denote an ordered sequence of database variables. In case of DELETE statement, observe that, in general, $\vec{v_d}$ includes all attributes of the target table.

Finally, we introduce a particular assignment "v_a :=?", called random assignment, in the instruction set, that models the insertion of input values at run time by an external user.

3.4 Environment and State

In this section, we introduce different type of environments and states associated with programs embedding SQL code. Consider a database instance $dB1$ consisting of two tables t_{emp} and t_{dept}, depicted in Table 3.2.

Table 3.2: Database $dB1$

(a) t_{emp}

eID	*Name*	*Age*	*Dno*	*Pno*	*Sal*	*Child-no*
1	Matteo	30	2	1	2000	4
2	Alice	22	1	2	1500	2
3	Joy	50	2	3	2300	3
4	luca	10	1	2	1700	1
5	Deba	40	3	4	3000	5
6	Andrea	70	1	2	1900	2
7	Alberto	18	3	4	800	1
8	Bob	14	2	3	4000	3

(b) t_{dept}

Deptno	*Dname*	*Loc*	*MngrID*
1	Math	Turin	4
2	Computer	Venice	1
3	Physics	Mestre	5

Environment

The program P embedding SQL statements acts on a set of constants $const(P) \in \wp(\mathbb{C})$ and set of variables $var(P) \in \wp(\mathbb{V})$, where $\mathbb{V} = \mathbb{V}_d \cup \mathbb{V}_a$. These variables take their values from semantic domain $\mathfrak{D}_\mho$, where $\mathfrak{D}_\mho = \{\mathfrak{D} \cup \{\mho\}\}$ and $\mho$ represents the undefined value.

Now we define two environments $\mathfrak{E}_d$ and $\mathfrak{E}_a$ corresponding to the database and application variable sets $\mathbb{V}_d$ and $\mathbb{V}_a$ respectively.

Definition 1 (Application Environment) *An application environment ρ_a is a partial function $\rho_a : \mathbb{V}_a \longmapsto \mathfrak{D}_{\mho}$. We denote by $\mathfrak{E}_a$ the set of all application environments.*

Definition 2 (Database Environment) *A database is a set of tables $\{t_i \mid i \in I_x\}$ for a given set of indexes I_x. We may define a function ρ_d whose domain is I_x, such that for $i \in I_x$, $\rho_d(i) = t_i$.*

In the example depicted in Table 3.2, the index set I_x is $\{emp, dept\}$, and the database d is the set $\{t_{emp}, t_{dept}\}$. So, $\rho_d(emp) = t_{emp}$, for example.

Definition 3 (Table Environment) *Given a database environment ρ_d and a table $t \in d$. We define $attr(t) = \{a_1, a_2, ..., a_k\}$. So, $t \subseteq D_1 \times D_2 \times .... \times D_k$ 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 $a_i \in attr(t)$,*

$$\rho_t(a_i) = \langle \pi_i(l_j) \mid l_j \in t \rangle$$

Where π is the projection operator, i.e. $\pi_i(l_j)$ is 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.

In the example of Table 3.2, $dom(\rho_{t_{emp}})$={*eID*, *Name*, *Age*, *Dno*, *Pno*, *Sal*, *Child-no*}. So, for example, $\rho_{t_{emp}}(Age)$= $\langle$30, 22, 50, 10, 40, 70, 18, 14$\rangle$.

Relation between Database Environment and Table Environment. Given a database d and a table $t_i \in d$ with $\vec{a} = attr(t_i)$. Then, $\rho_d(i) = \rho_{t_i}(\vec{a})$.

State and State Transition

Given a program P embedding SQL code, we define a state $\sigma \in \Im$ as a triplet $\langle I, \rho_d, \rho_a \rangle$ where $I \in \mathbb{I}$ is the instruction to be executed, ρ_d and ρ_a are the database environment and application environment respectively on which I is executed. Thus,

$$\Im \triangleq \mathbb{I} \times \mathfrak{E}_d \times \mathfrak{E}_a$$

where $\mathfrak{E}_d$ denotes the set of all database environments, and $\mathfrak{E}_a$ denotes the set of all application environments. The set of states of a program P is defined as:

$$\Im[\![P]\!] \triangleq P \times \mathfrak{E}[\![P]\!]$$

where $\mathfrak{E}[\![P]\!]$ is the set of environment of the program P whose domain is the set of program variables.

The state transition relation is defined as $\Gamma \triangleq \mathfrak{S} \longmapsto \wp(\mathfrak{S})$. The transitional semantics of a program P is, thus, defined as $\Gamma[\![P]\!] \triangleq \mathfrak{S}[\![P]\!] \longmapsto \wp(\mathfrak{S}[\![P]\!])$.

In the next sections, we will describe in details the semantic functions $E[\![.]\!]$ and $B[\![.]\!]$ for evaluating arithmetic and boolean expressions respectively, and $S[\![.]\!]$ for evaluating SQL statements.

3.5 Formal Semantics of Expressions

The evaluation of arithmetic expressions is defined by distinguishing different basic cases:

1. $E[\![c]\!](\rho_d, \rho_a) = c$

2. $E[\![v_a]\!](\rho_d, \rho_a) = \rho_a(v_a)$

3. $E[\![v_d]\!](\rho_d, \rho_a)$
   Let $\exists t \in dom(\rho_d) : \ v_d = a_i \in attr(t)$ in
   $= E[\![v_d]\!](\rho_t, \rho_a)$
   $= \rho_t(a_i)$

4. $E[\![v_d \ op \ c]\!](\rho_d, \rho_a)$ where op represents the arithmetic operation.
   Let $\exists t \in dom(\rho_d) : \ v_d = a_i \in attr(t)$ and $op : \ D_i \times D_j \to D_k$ in
   $= E[\![v_d \ op \ c]\!](\rho_t, \rho_a)$
   $= \langle (m \ op \ c) \in D_k \mid m \in \rho_t(a_i) \wedge a_i \in D_i \wedge c \in D_j \rangle$

5. $E[\![v_d \ op \ v_a]\!](\rho_d, \rho_a)$
   Let $\exists t \in dom(\rho_d) : \ v_d = a_i \in attr(t)$ and $op : D_i \times D_j \to D_k$ in
   $= E[\![v_d \ op \ v_a]\!](\rho_t, \rho_a)$
   $= \langle (m \ op \ n) \in D_k \mid m \in \rho_t(a_i) \wedge \rho_a(v_a) = n \wedge a_i \in D_i \wedge v_a \in D_j \rangle$

6. $E[\![v_{d_1} \ op \ v_{d_2}]\!](\rho_d, \rho_a)$
   Let $\exists t \in dom(\rho_d) : \ v_{d_1} = a_i, v_{d_2} = a_j, \{a_i, a_j\} \subseteq attr(t)$ and $op : \ D_i \times D_j \to D_k$ in
   $= E[\![v_{d_1} \ op \ v_{d_2}]\!](\rho_t, \rho_a)$
   $= \langle m_r \in D_k | \ m_r = \pi_i(l_r) \ op \ \pi_j(l_r)$ where l_r is the r^{th} row of $t \rangle$

7. $E[\![e_1\ op\ e_2]\!](\rho_d, \rho_a)$=
$$\begin{cases} \textbf{Case 1} : \exists!\ t \in dom(\rho_d) :\ \text{if } v_d \text{ occurs in } e_1 \text{ or } e_2 \text{ and } v_d = a \in attr(t), \\ = E[\![e_1\ op\ e_2]\!](\rho_t, \rho_a) \\ = E[\![e_1]\!](\rho_t, \rho_a)\ op\ E[\![e_2]\!](\rho_t, \rho_a) \\ \\ \textbf{Case 2} : \text{Let } T = \{t \in dom(\rho_d) \mid \exists v_d \text{ occurring in } e_1 \text{ or } e_2\ s.t.\ v_d = a \in attr(t)\}, \\ \text{Let } T = \{t_1, t_2, ..., t_n\} \text{ and } t' = t_1 \times t_2 \times ... \times t_n, \\ = E[\![e_1\ op\ e_2]\!](\rho_{t'}, \rho_a) \end{cases}$$

We generalize the arithmetic operation op on lists as follows: Suppose op is a binary arithmetic operation over two lists S_1 and S_2. Also assume, $S' \subseteq S_1, s_1 \in S_1$ and $S'' \subseteq S_2$, $s_2 \in S_2$ with $|S'| = |S''|$, then the generalization of op is defined by:

$$op \triangleq \begin{cases} S'\ op\ s_2 = \langle s\ op\ s_2 \mid s \in S' \rangle \\ s_1\ op\ S'' = \langle s_1\ op\ s \mid s \in S'' \rangle \\ S'\ op\ S'' = \langle s'_i\ op\ s''_i \mid s'_i \text{ and } s''_i \text{ are the } i^{th} \text{ element of } S' \text{ and } S'' \text{ respectively} \rangle \end{cases}$$

Finally, the evaluation of boolean expressions is defined by:

1. $B[\![true]\!](\rho_d, \rho_a) = true$
2. $B[\![false]\!](\rho_d, \rho_a) = false$
3. $B[\![e_1\ op_r\ e_2]\!](\rho_d, \rho_a) = E[\![e_1]\!](\rho_d, \rho_a)\ op_r\ E[\![e_2]\!](\rho_d, \rho_a)$, where op_r represents the relational operator.
4. $B[\![\neg b]\!](\rho_d, \rho_a) = \neg B[\![b]\!](\rho_d, \rho_a)$
5. $B[\![b_1 \vee b_2]\!](\rho_d, \rho_a) = B[\![b_1]\!](\rho_d, \rho_a) \vee B[\![b_2]\!](\rho_d, \rho_a)$
6. $B[\![b_1 \wedge b_2]\!](\rho_d, \rho_a) = B[\![b_1]\!](\rho_d, \rho_a) \wedge B[\![b_2]\!](\rho_d, \rho_a)$

3.6 Formal Semantics of Program Statements

Semantics $S[\![I]\!](\rho_d, \rho_a)$ of an instruction I in a program embedding SQL code defines the effect of executing this instruction on the environment ρ_a or (ρ_d, ρ_a). There are two types of instructions: one executed only on ρ_a and other executed on both database and application environment (ρ_d, ρ_a) together. The SQL statements Q_{sql} belong to the second category, whereas all other instructions of the application belong to the first category.

3.6.1 Semantics of SELECT Statement

In this section, we describe the semantics of SELECT statement and we illustrate it with an example.

Consider the database of Table 3.2 and the following SELECT statement Q_{select}:

SELECT DISTINCT *Dno*, *Pno*, MAX(*Sal*), AVG(DISTINCT *Age*), COUNT(*) FROM t_{emp} INTO v_a WHERE *Sal* >1000 GROUP BY *Dno*, *Pno* HAVING MAX(*Sal*)<4000 ORDER BY AVG(DISTINCT *Age*), *Dno*

An equivalent formulation of Q_{select} is:

SELECT DISTINCT(⟨DISTINCT(*Dno*), DISTINCT(*Pno*), MAX∘ALL(*Sal*), AVG∘DISTINCT(*Age*), COUNT(*)⟩) FROM t_{emp} INTO v_a WHERE *Sal* >1000 GROUP BY ⟨*Dno*, *Pno*⟩ HAVING (MAX∘ALL(*Sal*))<4000 ORDER BY ⟨AVG∘DISTINCT(*Age*), *Dno*⟩

According to the abstract syntax, we get:

- ϕ_1 = *Sal* >1000
- $\vec{e}$ = ⟨*Dno*, *Pno*⟩
- $g(\vec{e})$ = GROUP BY(⟨*Dno*, *Pno*⟩)
- ϕ_2 = (MAX∘ALL(*Sal*))<4000
- $\vec{h}$ = ⟨DISTINCT, DISTINCT, MAX∘ALL, AVG∘DISTINCT, COUNT⟩
  $\vec{x}$ = ⟨*Dno*, *Pno*, *Sal*, *Age*, ∗⟩
  $\vec{h}(\vec{x})$ = ⟨DISTINCT(*Dno*), DISTINCT(*Pno*), MAX∘ALL(*Sal*), AVG∘DISTINCT(*Age*), COUNT(*)⟩
- $\vec{e'}$ = ⟨AVG∘DISTINCT(*Age*), *Dno*⟩, where AVG∘DISTINCT(*Age*) simply represents an expression after the application of $\vec{h}(\vec{x})$.
- $f(\vec{e'})$ = ORDER BY ASC(⟨AVG∘DISTINCT(*Age*), *Dno*⟩)
- v_a = Record or ResultSet type application variable with fields $\vec{w}$=⟨w_1, w_2, w_3, w_4, w_5⟩. The type of w_1, w_2, w_3, w_4, w_5 are same as the return type of DISTINCT(*Dno*), DISTINCT(*Pno*), MAX∘ALL(*Sal*), AVG∘DISTINCT(*Age*), COUNT(*) respectively. For instance, in java as a host language, v_a represents the object of the type *ResultSet*. Observe that, here we use the term INTO to understand the assignment into the application variable.

Thus, Q_{select} is of the form as follows:

SELECT $r(\vec{h}(\vec{x}))$ FROM t_{emp} INTO $v_a(\vec{w})$ WHERE ϕ_1 GROUP BY $\vec{e}$ HAVING ϕ_2 ORDER BY ASC $\vec{e'}$

We now describe the semantics of SELECT statement step by step using the above example.

Recall from Table 3.1 that the syntax of SELECT statement is defined as:

$$\langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2,\ g(\vec{e})),\ \phi_1\rangle$$

The Semantics of SELECT statement is described below:

$$S[\![\langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2,\ g(\vec{e})),\ \phi_1\rangle]\!]_\varsigma(\rho_d, \rho_a)$$
$$= \begin{cases} S[\![\langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2,\ g(\vec{e})),\ \phi_1\rangle]\!]_\varsigma(\rho_t, \rho_a) \\ \text{if } \exists!\ t \in dom(\rho_d) :\ target(\langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2,\ g(\vec{e})),\ \phi_1\rangle) = \{t\} \\ \\ S[\![\langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2,\ g(\vec{e})),\ \phi_1\rangle]\!]_\varsigma(\rho_{t'}, \rho_a) \text{ otherwise,} \\ \text{where } T = \{t_1, \ldots, t_n \in dom(\rho_d) \mid t_i \text{ occurs in } Q_{select}\} \text{ and } t' = t_1 \times t_2 \times \cdots \times t_n. \end{cases}$$

Below the semantics of SELECT statement is unfolded step by step:

Step 1. Absorbing ϕ_1:

$$S[\![\langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2,\ g(\vec{e})),\ \phi_1\rangle]\!]_\varsigma(\rho_{t_0}, \rho_a)$$
$$= S[\![\langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2,\ g(\vec{e})),\ true\rangle]\!]_\varsigma(\rho_{t'}, \rho_a), \text{ where}$$

$$t' = \langle\ l_i \in t_0 \mid \text{let } var(\phi_1) = \vec{v'_d} \cup \vec{v'_a} \text{ with } \vec{v'_d} = \vec{a} \subseteq attr(t_0) :\ \varsigma \models \phi_1[\ \pi_{\vec{a}}(l_i)/\vec{v'_d}\ ][\ \rho_a(\vec{v'_a})/\vec{v'_a}\ ]\ \rangle$$

Example: Since in our example $target(Q_{select}) = \{t_{emp}\}$, we apply `WHERE` clause ϕ_1=*Sal* >1000 on t_{emp}. The result is depicted in Table 3.3(a). The row "*eID*:7; *Name*:Alberto; *Age*:18; *Dno*:3; *Pno*:4; *Sal*:800; *Child* − *no*:1" is disregarded from the result because, $\varsigma \not\models \phi_1[800/Sal]$. In fact, the semantic structure ς does not satisfy ϕ_1 when the variable '*Sal*' is substituted by the value '800' of the corresponding row.

Step 2. Grouping:

$$S[\![\langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi,\ g(\vec{e})),\ true\rangle]\!]_\varsigma(\rho_t, \rho_a)$$
$$= S[\![\langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi,\ id),\ true\rangle]\!]_\varsigma(\rho_T, \rho_a), \text{ where}$$

$g(\vec{e}) =$ `Group By`$(\vec{e})$ and $g(\vec{e})[t]$ is the maximal partition $T = \{t_1, t_2, ..., t_n\}\ of\ t,\ s.t.$
$\forall t_i \in T,\ t_i \subseteq t$ and $\forall e_j \in \vec{e},\ \forall m_k, m_l \in E[\![e_j]\!](\rho_{t_i}, \rho_a) :\ m_k = m_l$

Example: Applying the grouping function $g(\vec{e})$=`GROUP BY`($\langle Dno, Pno\rangle$) on the result of step 1 based on the argument $\langle Dno, Pno\rangle$, we get 4 different partitions with $\langle 2,1\rangle$, $\langle 1,2\rangle$, $\langle 2,3\rangle$ and $\langle 3,4\rangle$ as the values of $\langle Dno, Pno\rangle$, depicted in Table 3.3(b).

Step 3. Absorbing ϕ:

$S[\![\langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi,\ id),\ true\rangle]\!]_\varsigma(\rho_T, \rho_a)$
$= S[\![\langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ true,\ id),\ true\rangle]\!]_\varsigma(\rho_{T'}, \rho_a)$, where

T' is defined as follows: there is a sequence of functions $\vec{h'}$ occuring in ϕ,

operating on groups, such that : $\vec{h'}(\vec{x'}) \ni h'_i(x'_i) \triangleq \begin{cases} \texttt{COUNT(*)} \text{ or,} \\ \texttt{DISTINCT}(e) \text{ or,} \\ s \circ r(e) \end{cases}$

Let $\vec{v'_a}$ be a sequence of application variables occurring in ϕ and,
$\forall t_i \in T,\ \vec{h'}(\langle E[\![\vec{x'}]\!](\rho_{t_i}, \rho_a)\rangle) = \vec{c_i}$ and $T' = \{t_i \in T \mid \varsigma \models \phi[\vec{c_i}/\vec{h'}(\vec{x'})][\rho_a(\vec{v'_a})/\vec{v'_a}]\}$

Example: We apply the `HAVING` clause ϕ_2=`MAX`$\circ$`ALL`(Sal)<4000 over all the groups in step 2. One group with the value of $\langle Dno, Pno\rangle$ equal to $\langle 2,3\rangle$ has been disregarded, since the maximum salary of that group is not less than 4000. That is, the semantic structure ς does not satisfy ϕ_2 after interpreting `MAX`$\circ$`ALL`(Sal) in ϕ_2 with the value which is returned by the function `MAX`$\circ$`ALL`(Sal) applying on that group. The result is shown in Table 3.3(c).

Step 4. Applying $r(\vec{h}(\vec{x}))$ on each group in T:

$= S[\![\langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ true,\ id),\ true\rangle]\!]_\varsigma(\rho_T, \rho_a)$
$= S[\![\langle select(v_a,\ f(\vec{e'}),\ id,\ true,\ id),\ true\rangle]\!]_\varsigma(\rho_t, \rho_a)$, where

$$t' = \langle\ \vec{h}(\ E[\![\vec{x}]\!](\rho_{t_i}, \rho_a)) \mid t_i \in T\ \rangle \text{ and } t = r[t']$$

As we mentioned earlier that the generation of maximal partitions T of the tuples depends on whether (i) the function g is present or not, and (ii) $\vec{h}$ is $\vec{id}$ or not.

Example: In the example, $r(\vec{h}(\vec{x})) = \texttt{DISTINCT}(\langle \texttt{DISTINCT}(Dno), \texttt{DISTINCT}(Pno), \texttt{MAX} \circ \texttt{ALL}(Sal), \texttt{AVG} \circ \texttt{DISTINCT}(Age), \texttt{COUNT(*)} \rangle)$. We perform $r(\vec{h}(\vec{x}))$ on each group resulting from step 3. We have three steps:

(a) On each group, perform $\vec{h}(\vec{x}) = \langle \texttt{DISTINCT}(Dno), \texttt{DISTINCT}(Pno), \texttt{MAX} \circ \texttt{ALL}(Sal), \texttt{AVG} \circ \texttt{DISTINCT}(Age), \texttt{COUNT(*)} \rangle$: After applying $\vec{h}(\vec{x})$ on each group, we get the result as in Table 3.3(d).

(b) Get the table t out of these results obtained in step (a): This is shown in Table 3.3(e).

(c) Apply r=DISTINCT on the rows of table t obtained in step (b): We get Table 3.3(f) which is equal to the Table 3.3(e), since there is no duplicate rows.

Step 5. Possibly applying the ordering:

$$S[\![\langle select(v_a,\ f(\vec{e'}),\ id,\ true,\ id),\ true \rangle]\!]_\varsigma(\rho_t, \rho_a)$$
$$= S[\![\langle select(v_a,\ id,\ id,\ true,\ id),\ true \rangle]\!]_\varsigma(\rho_{t'}, \rho_a), \text{ where } t' = f(\vec{e})[t]$$

Example: Performing $f(\vec{e'})$=ORDER BY ASC($\langle$AVG$\circ$DISTINCT(Age), $Dno\rangle$) on Table 3.3(f), we get Table 3.3(g).

Step 6. Set the resulting values to the Record/ResultSet type application variable v_a with fields $\vec{w}$:

$$S[\![\langle select(v_a,\ id,\ id,\ true,\ id),\ true \rangle]\!]_\varsigma(\rho_t, \rho_a)$$
$$= (\rho_{t_0}, \rho_{a'}), \text{ where}$$

$$\rho_{a'} = \rho_a[\rho_t(\vec{a})/v_a(\vec{w})] \text{ with } \vec{a} = attr(t) \text{ and } t_0 \text{ is the initial table of step 1.}$$
Here, the i^{th} field $w_i \in \vec{w}\ of\ v_a$ is substituted by i^{th} attribute $a_i \in \vec{a}$ of t.

Example: Finally, the result obtained in step 5 is assign to the application variable v_a with fields $\vec{w} = \langle w_1, w_2, w_3, w_4, w_5 \rangle$. The result is shown in Table 3.3(h).

Table 3.3: Operations of Q_{select}

(a) Absorbing `WHERE` Clause ϕ_1

eID	*Name*	*Age*	*Dno*	*Pno*	*Sal*	*Child − no*
1	Matteo	30	2	1	2000	4
2	Alice	22	1	2	1500	2
3	Joy	50	2	3	2300	3
4	luca	10	1	2	1700	1
5	Deba	40	3	4	3000	5
6	Andrea	70	1	2	1900	2
~~7~~	~~Alberto~~	~~18~~	~~3~~	~~4~~	~~800~~	~~1~~
8	Bob	14	2	3	4000	3

(b) `Grouping`

eID	*Name*	*Age*	*Dno*	*Pno*	*Sal*	*Child − no*
1	Matteo	30	2	1	2000	4
2	Alice	22	1	2	1500	2
4	luca	10	1	2	1700	1
6	Andrea	70	1	2	1900	2
3	Joy	50	2	3	2300	3
8	Bob	14	2	3	4000	3
5	Deba	40	3	4	3000	5

(c) Absorbing `HAVING` clause ϕ_2

eID	*Name*	*Age*	*Dno*	*Pno*	*Sal*	*Child − no*
1	Matteo	30	2	1	2000	4
2	Alice	22	1	2	1500	2
4	luca	10	1	2	1700	1
6	Andrea	70	1	2	1900	2
~~3~~	~~Joy~~	~~50~~	~~2~~	~~3~~	~~2300~~	~~3~~
~~8~~	~~Bob~~	~~14~~	~~2~~	~~3~~	~~4000~~	~~3~~
5	Deba	40	3	4	3000	5

(d) Performing $\vec{h}(\vec{x})$

Dno	*Pno*	*MAX*(*Sal*)	*AVG*(*DISTINCT Age*)	*COUNT*(∗)
2	1	2000	30	1
1	2	1900	34	3
3	4	3000	40	1

(e) Getting table out of the result from (d)

Dno	*Pno*	*MAX*(*Sal*)	*AVG*(*DISTINCT Age*)	*COUNT*(∗)
2	1	2000	30	1
1	2	1900	34	3
3	4	3000	40	1

Table 3.3: Operations of Q_{select}

(f) Elimination of duplicates

Dno	*Pno*	*MAX*(*Sal*)	*AVG*(*DISTINCT Age*)	*COUNT*(∗)
2	1	2000	30	1
1	2	1900	34	3
3	4	3000	40	1

(g) Ordering

Dno	*Pno*	*MAX*(*Sal*)	*AVG*(*DISTINCT Age*)	*COUNT*(∗)
2	1	2000	30	1
1	2	1900	34	3
3	4	3000	40	1

(h) Assign to $\vec{v_a}$

w_1	w_2	w_3	w_4	w_5
2	1	2000	30	1
1	2	1900	34	3
3	4	3000	40	1

3.6.2 Semantics of UPDATE Statement

Consider the database of Table 3.2 and the following UPDATE statement Q_{update}:

$$\texttt{UPDATE}\ t_{emp}\ \texttt{SET}\ Age\texttt{:=}Age\texttt{+2},\ Sal\texttt{:=}Sal\texttt{+}Sal\texttt{×0.5}\ \texttt{WHERE}\ Sal\texttt{>1500}$$

According to the abstract syntax we get:

- $\phi_1 = Sal > 1500$
- $\vec{v_d} = \langle Age, Sal \rangle$
- $\vec{e} = \langle Age{+}2, Sal{+}Sal{\times}0.5 \rangle$

Thus, Q_{update} is of the form as below:

$$\texttt{UPDATE}\ t_{emp}\ \texttt{SET}\ \vec{v_d} := \vec{e}\ \texttt{WHERE}\ \phi$$

Recall from Table 3.1 that the syntax of UPDATE statement is defined as:

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

The semantics of UPDATE statement is described as follows: The update statement always targets an individual table. Let

$$target(\langle update(\vec{v_d},\ \vec{e}),\ \phi\rangle) = \{t\}$$

where, $t \in dom(\rho_d)$. Therefore,

$$S[\![\langle update(\vec{v_d},\ \vec{e}),\ \phi\rangle]\!]_\varsigma(\rho_d, \rho_a) = S[\![\langle update(\vec{v_d},\ \vec{e}),\ \phi\rangle]\!]_\varsigma(\rho_t, \rho_a)$$

Below the semantics of UPDATE statement is unfolded step by step:

Step 1: Absorbing ϕ:

$S[\![\langle update(\vec{v_d},\ \vec{e}),\ \phi\rangle]\!]_\varsigma(\rho_t, \rho_a)$
$= S[\![\langle update(\vec{v_d},\ \vec{e}),\ true\rangle]\!]_\varsigma(\rho_{t'}, \rho_a)$, where

$t' = \langle\ l_i \in t \mid \text{let } var(\phi) = \vec{v'_d} \cup \vec{v'_a} \text{ with } \vec{v'_d} = \vec{a} \subseteq attr(t) :\ \varsigma \models \phi[\ \pi_{\vec{a}}(l_i)/\vec{v'_d}\ ][\ \rho_a(\vec{v'_a})/\vec{v'_a}\ ]\ \rangle$

Example: In the example, $target(Q_{update}) = \{t_{emp}\}$. Applying the `WHERE` clause ϕ=*sal* >1500 on t_{emp}, we get the result t'_{emp} depicted in Table 3.4(a). Observe that two rows are disregarded as they don't satisfy the semantic structure ς of ϕ. That is, $\varsigma \not\models \phi$[1500/*Sal*] and $\varsigma \not\models \phi$[800/*Sal*].

Step 2: Update:

$S[\![\langle update(\vec{v_d},\ \vec{e}),\ true\rangle]\!]_\varsigma(\rho_t, \rho_a)$
$= (\rho_{t'}, \rho_a)$, where

let $\vec{v_d} = \vec{a} \subseteq attr(t)$ and $\vec{e} = \langle e_1, e_2, ..., e_h\rangle$ and $E[\![\vec{e}]\!](\rho_t, \rho_a) = \langle \vec{m_i} \mid i = 1, ..., h\rangle$, and let m_i^j be the j^{th} element of the sequence $\vec{m_i}$ and a_i be the i^{th} element of the sequence $\vec{a}$, and $t' = \langle\ l_j[m_i^j/a_i] \mid l_j \in t\ \rangle$

Example: Performing the update operation ($\vec{v_d} := \vec{e}$)=($\langle$*Age* := *Age*+2, *Sal* := *Sal* + *Sal*×0.5$\rangle$) on table t'_{emp} of step 1, we get the updated table t''_{emp} as shown in 3.4(b). Here, two expressions (*Age*+2) and (*Sal*+*Sal*×0.5) are evaluated over the environment ($\rho_{t'_{emp}}$, ρ_a) first, and then for each rows of the table, two attributes '*Age*' and '*Sal*' are updated with the corresponding evaluated results respectively.

Table 3.4: Operations of Q_{update}

(a) Table t'_{emp}: After absorbing `WHERE` clause ϕ

eID	*Name*	*Age*	*Dno*	*Pno*	*Sal*	*Child − no*
1	Matteo	30	2	1	2000	~~4~~
~~2~~	~~Alice~~	~~22~~	~~1~~	~~2~~	~~1500~~	~~2~~
3	Joy	50	2	3	2300	3
4	luca	10	1	2	1700	1
5	Deba	~~40~~	3	~~4~~	3000	5
6	Andrea	70	1	2	1900	2
~~7~~	~~Alberto~~	~~18~~	~~3~~	~~4~~	~~800~~	~~1~~
8	Bob	14	2	3	4000	3

(b) Table t''_{emp}: after update

eID	*Name*	*Age*	*Dno*	*Pno*	*Sal*	*Child − no*
1	Matteo	32	2	1	3000	4
3	Joy	52	2	3	3450	3
4	luca	12	1	2	2550	1
5	Deba	42	3	4	4500	5
6	Andrea	72	1	2	2850	2
8	Bob	16	2	3	6000	3

Evaluation of the expression (Age+2) over the environment ($\rho_{t'_{emp}}, \rho_a$) gives the following results:

$$E[\![Age + 2]\!](\rho_{t'_{emp}}, \rho_a) = \langle 32, 52, 12, 42, 72, 16 \rangle$$

$$E[\![Sal + Sal \times 0.5]\!](\rho_{t'_{emp}}, \rho_a) = \langle 3000, 3450, 2550, 4500, 2850, 6000 \rangle$$

Now the updation of the attribute 'Age' is done for all rows as follows:

$$\langle l_1(32/Age), l_2(52/Age), l_3(12/Age), l_4(42/Age), l_5(72/Age), l_6(16/Age) \rangle$$

We do the same for Sal:=Sal+Sal×0.5.

3.6.3 Semantics of INSERT Statement

Consider the database of Table 3.2 and the following INSERT statement Q_{insert}:

`INSERT INTO` t_{dept} `VALUES (4, 'Electronins', 'Trieste', 2)`

According to the abstract syntax we get:

- $\vec{v_d} = \langle Deptno, Dname, Loc, MngrID \rangle$
- $\vec{e} = \langle 4, 'Electronins', 'Trieste', 2 \rangle$

Thus, Q_{insert} is of the following form:

$$\texttt{INSERT INTO } \vec{v_d} \texttt{ VALUES } \vec{e}$$

Recall from Table 3.1 that the Syntax of INSERT statement is defined as:

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

The Semantics of INSERT Statement is described as follows: The insert statement always targets an individual table. Let

$$t \in dom(\rho_d) :\ target(\langle insert(\vec{v_d},\ \vec{e}),\ \Omega\rangle) = \{t\}$$

Therefore,

$S[\![\langle insert(\vec{v_d},\ \vec{e}),\ \phi\rangle]\!]_\varsigma(\rho_d, \rho_a)$
$= S[\![\langle insert(\vec{v_d},\ \vec{e}),\ \phi\rangle]\!]_\varsigma(\rho_t, \rho_a)$
$= S[\![\langle insert(\vec{v_d},\ \vec{e}),\ true\rangle]\!]_\varsigma(\rho_t, \rho_a)$
$= (\rho_{t'}, \rho_a)$, where

$$\text{let } \vec{v_d} = \vec{a} = \langle a_1, a_2, ..., a_n\rangle \subseteq attr(t) \text{ and } E[\![\vec{e}]\!](\rho_a) = \vec{x} = \langle x_1, x_2, ..., x_n\rangle,$$
$$l_{new} = \langle x_1/a_1, x_2/a_2, ..., x_n/a_n\rangle \text{ in } t' = t \cup \{l_{new}\}$$

Observe that we suppose $\vec{v_d}$ includes all attributes of the target table t. Although there exists alternative syntax where we can insert the values for selective attributes only, we can easily convert the alternative syntax into the one mentioned above by inserting undefined value υ in $\vec{e}$ for the unspecified attributes.

Example: In the example, $target(Q_{insert}) = \{t_{dept}\}$. Since, $E[\![\vec{e}]\!](\rho_a) = \langle 4, 'Electronics', 'Trieste', 2\rangle$ and $\vec{v_d}$=$\vec{a}$ $= \langle Deptno, Dname, Loc, MngrID\rangle$, we get $l_{new} = \langle 4/Deptno, 'Electronics'/Dname, 'Trieste'/Loc, 2/MngrID\rangle$. After inserting the new row l_{new}, we get the resulting table t'_{dept} as shown in Table 3.5 while the application environment ρ_a keeps unchanged.

Table 3.5: Operation of Q_{insert}

Deptno	*Dname*	*Loc*	*MngrID*
1	Math	Turin	4
2	Computer	Venice	1
3	Physics	Mestre	5
4	Electronins	Trieste	2

3.6.4 Semantics of DELETE Statement

Consider the database of Table 3.2 and the following DELETE statement Q_{delete}:

$$\texttt{DELETE FROM } t_{emp} \texttt{ WHERE } Sal \geq 1800$$

According to the abstract syntax we get:

$$\texttt{DELETE FROM } t_{emp} \texttt{ WHERE } \phi$$

where, ϕ represents the first-order formula "$Sal \geq$1800".

Recall from Table 3.1 that the syntax of DELETE statement is defined as:

$$\langle\ delete(\vec{v_d}),\ \phi\ \rangle$$

where $\vec{v_d}$ includes all attributes of the target table. The semantics of DELETE Statement is described as follows: The DELETE statement always targets an individual table. Let

$$t \in dom(\rho_d) :\ target(\langle delete(\vec{v_d}),\ \phi\rangle) = \{t\}$$

Therefore,

$S[\![\langle delete(\vec{v_d}),\ \phi\rangle]\!]_\varsigma(\rho_d, \rho_a)$
$= S[\![\langle delete(\vec{v_d}),\ \phi\rangle]\!]_\varsigma(\rho_t, \rho_a)$
$= (\rho_{t'}, \rho_a)$, where

$t_d = \langle\ l_i \in t \mid \text{let } var(\phi) = \vec{v'_d} \cup \vec{v'_a} \text{ with } \vec{v'_d} = \vec{a} \subseteq attr(t) :\ \varsigma \models \phi[\ \pi_{\vec{a}}(l_i)/\vec{v'_d}\ ][\ \rho_a(\vec{v'_a})/\vec{v'_a}\ ]\ \rangle$

$t' = t \backslash t_d$

Example: In the example, $target(Q_{delete}) = \{t_{emp}\}$. Applying ϕ=$Sal \geq$1800 and deleting the rows which satisfy ϕ, we get t'_{emp} as shown in Table 3.6. Here, five rows are deleted from the table as they satisfy ϕ.

Table 3.6: Operation of Q_{delete}

eID	*Name*	*Age*	*Dno*	*Pno*	*Sal*	*Child – no*
~~1~~	~~Matteo~~	~~30~~	~~2~~	~~1~~	~~2000~~	~~4~~
2	Alice	22	1	2	1500	2
~~3~~	~~Joy~~	~~50~~	~~2~~	~~3~~	~~2300~~	~~3~~
4	luca	10	1	2	1700	1
~~5~~	~~Deba~~	~~40~~	~~3~~	~~4~~	~~3000~~	~~5~~
~~6~~	~~Andrea~~	~~70~~	~~1~~	~~2~~	~~1900~~	~~2~~
7	Alberto	18	3	4	800	1
~~8~~	~~Bob~~	~~14~~	~~2~~	~~3~~	~~4000~~	~~3~~

3.6.5 Formal Semantics of Non-SQL Statements

$$S[\![skip]\!]_\varsigma(\rho_d, \rho_a) \triangleq (\rho_d, \rho_a)$$

$$S[\![v_a := e]\!]_\varsigma(\rho_d, \rho_a) \triangleq (\rho_d, \rho_a[E[\![e]\!](\rho_a)/v_a])$$

$$S[\![v_a :=?]\!]_\varsigma(\rho_d, \rho_a) \triangleq ((\rho_d, \rho_{a'}) \text{ where, } b \text{ is any value in } dom(v_a) \text{ in } \rho_{a'} = \rho_a[b/v_a]$$

3.7 Some Inference Rules for Composite Statements

The inference rules for composite instructions are obtained by induction:

$$\frac{S[\![Q_1]\!](\rho_d, \rho_a) = t_1 \qquad S[\![Q_2]\!](\rho_d, \rho_a) = t_2}{S[\![Q_1 \texttt{ UNION } Q_2]\!](\rho_d, \rho_a) = t_1 \cup t_2}$$

$$\frac{S[\![Q_1]\!](\rho_d, \rho_a) = t_1 \qquad S[\![Q_2]\!](\rho_d, \rho_a) = t_2}{S[\![Q_1 \texttt{ INTERSECT } Q_2]\!](\rho_d, \rho_a) = t_1 \cap t_2}$$

$$\frac{S[\![Q_1]\!](\rho_d, \rho_a) = t_1 \qquad S[\![Q_2]\!](\rho_d, \rho_a) = t_2}{S[\![Q_1 \texttt{ MINUS } Q_2]\!](\rho_d, \rho_a) = t_1 \backslash t_2}$$

$$\frac{S[\![I_1]\!](\rho_d, \rho_a) = (\rho_{d'}, \rho_{a'}) \qquad S[\![I_2]\!](\rho_{d'}, \rho_{a'}) = (\rho_{d''}, \rho_{a''})}{S[\![I_1; I_2]\!](\rho_d, \rho_a) = (\rho_{d''}, \rho_{a''})}$$

Consider the auxiliary conditional statement *cond*:

$$cond(B[\![b]\!], S[\![I_1]\!], S[\![I_2]\!])(\rho_d, \rho_a) = (\rho_{d'}, \rho_{a'})$$

where, either $B[\![b]\!](\rho_d, \rho_a) = true$ and $S[\![I_1]\!](\rho_d, \rho_a) = (\rho_{d'}, \rho_{a'})$, or $B[\![b]\!](\rho_d, \rho_a) = false$ and $S[\![I_2]\!](\rho_d, \rho_a) = (\rho_{d'}, \rho_{a'})$

The semantics of "$if\ b\ then\ I_1\ else\ I_2$" statement is expressed using the conditional statement *cond* as follows:

$$S[\![if\ b\ then\ I_1\ else\ I_2]\!](\rho_d, \rho_a) = cond(B[\![b]\!], S[\![I_1]\!], S[\![I_2]\!])(\rho_d, \rho_a)$$

The semantics of the "*while b do I*" statement is expressed as follows: Since "*while b do I*" ≡ "$if\ b\ then$ (*I*; *while b do I*) *else skip*", we can write:

$$S[\![while\ b\ do\ I]\!](\rho_d, \rho_a) = S[\![if\ b\ then\ (I;\ while\ b\ do\ I)\ else\ skip]\!](\rho_d, \rho_a) = FIX\ F$$

where, $F(g) = cond(B[\![b]\!], g \circ S[\![I]\!],\ id)(\rho_d, \rho_a)$ and *FIX* is a fix-point operator.

Definition 4 (Equivalence of Instructions) *Let the environments* (ρ_d, ρ_a) *and* $(\rho_{d'}, \rho_{a'})$ *be denoted by* ρ_x *and* $\rho_{x'}$ *respectively. Two instructions* I_1 *and* I_2 *are said to be equivalent if,* $\{(\rho_x, \rho_{x'}) \mid S[\![I_1]\!](\rho_x) = \rho_{x'}\} = \{(\rho_x, \rho_{x'}) \mid S[\![I_2]\!](\rho_x) = \rho_{x'}\}$. *In other words,* $I_1 \equiv I_2$ *if* I_1 *and* I_2 *determine the same partial function on states.*

3.8 Soundness of the Denotational Semantics of SQL with respect to the Standard Semantics

The abstract syntax and the denotational semantics of SQL introduced in the previous sections correspond to the standard syntax and semantics of SQL as defined by ANSI (107) and the Relational Algebra. In particular, we can prove a correspondence between our denotational approach to the standard relational model approach to each SQL statement. For instance, consider the following basic SQL statements embedded in Java:

Statement stmt = conn.createStatement();
String Q="`SELECT` $a_1, a_2, \ldots, a_n$ `FROM` t `WHERE` C";
ResultSet rs = stmt.executeQuery(Q);

Where $a_1, a_2, \ldots, a_n$ represents the attributes of the table t and C is a condition. An equivalent representation of the above SQL statement in Relational Algebra is:

$$\begin{array}{ll} t' = \sigma_C(t) & \\ t'' = \pi_{\vec{a}}(t') & \text{where, } \vec{a} = \langle a_1, a_2, \ldots, a_n \rangle \\ rs = t'' & \end{array}$$

In our proposed denotational approach, an equivalent formulation is:
$\langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2,\ g(\vec{e})),\ \phi_1\rangle$
$= \langle select(rs,\ id,\ \texttt{ALL}(\vec{id}(\vec{a})),\ true,\ id),\ C\rangle$, where $\vec{id}(\vec{a}) = \langle id(a_1), id(a_2), \ldots, id(a_n)\rangle$

Given an environment (ρ_d, ρ_a), the semantics of Q is described as follows:
$S[\![\langle select(rs,\ id,\ \texttt{ALL}(\vec{id}(\vec{a})),\ true,\ id),\ C\rangle]\!]_\varsigma(\rho_d, \rho_a)$
$= S[\![\langle select(rs,\ id,\ \texttt{ALL}(\vec{id}(\vec{a})),\ true,\ id),\ C\rangle]\!]_\varsigma(\rho_t, \rho_a)$, where $target(Q) = t \in d$
$= S[\![\langle select(rs,\ id,\ \texttt{ALL}(\vec{id}(\vec{a})),\ true,\ id),\ true\rangle]\!]_\varsigma(\rho_{t'}, \rho_a)$, where

$$t' = \langle\ l_i \in t \mid \text{let } var(C) = \vec{v'_d} \cup \vec{v'_a} \text{ with } \vec{v'_d} = \vec{x} \subseteq attr(t) :\ \varsigma \models C[\ \pi_{\vec{x}}(l_i)/\vec{v'_d}\ ][\ \rho_a(\vec{v'_a})/\vec{v'_a}\ ]\ \rangle$$
$$= \sigma_C(t) \tag{3.1}$$

$= S[\![\langle select(rs,\ id,\ id,\ true,\ id),\ true\rangle]\!]_\varsigma(\rho_{t''}, \rho_a)$, where

$$t'' = \texttt{ALL}[\ \langle\ \vec{id}(\ E[\![\vec{a}]\!](\rho_{t'}, \rho_a)\ )\ \rangle\ ]$$
$$= E[\![\vec{a}]\!](\rho_{t'}, \rho_a),\ \text{since } \texttt{ALL} \text{ does not remove or modify any element.}$$
$$= \pi_{\vec{a}}(t'),\ \text{according to the semantics of expressions.} \tag{3.2}$$

$= (\rho_t, \rho_{a'})$, where

$$\rho_{a'}(rs) = t'' \tag{3.3}$$

Observe that the equations 3.1, 3.2 and 3.3 shows the correspondence between the Relational Algebra and Denotational semantic approaches.

3.9 Abstract Semantics of Programs embedding SQL Statements

In this section, we lift the semantics of SQL operations defined so far to an abstract setting, where instead of working on the concrete databases, SQL statements are applied to abstract databases, in which some information are disregarded and concrete values are possibly represented by suitable abstractions.

Abstract Databases

Generally, traditional databases are *concrete databases* as they contain data from concrete domains, whereas *abstract databases* are obtained by replacing concrete values by the elements from abstract domains representing specific properties of interest. We may distinguish "*partial abstract databases*" in contrast to "$full\ abstract\ databases$", as in the former case only a subset of the data in the databases is abstracted. The values of the data cells belonging to an attribute x are abstracted by using the Galois Connection $(\wp(D_x^{con}), \alpha_x, \gamma_x, D_x^{abs})$, where $\wp(D_x^{con})$ and D_x^{abs} represent the powerset of concrete domain of x and an abstract domain of x respectively, whereas α_x and γ_x represent the corresponding abstraction and concretization functions (denoted $\alpha_x : \wp(D_x^{con}) \rightarrow D_x^{abs}$ and $\gamma_x : D_x^{abs} \rightarrow \wp(D_x^{con})$) respectively. In particular, partial abstract databases are special case of fully abstract databases where abstraction and concretization functions for some attributes x are identity function id, and thus, use the Galois Connection $(\wp(D_x^{con}), id, id, \wp(D_x^{con}))$. Let us illustrate it by an example.

Example 1 *The database in Table 3.2 consists of a concrete table t_{emp} that provides information about the employees of a company. We assume that the ages, salaries, and number of children of the employees lie between 5 to 100, between 500 to 10000 and between 0 to 10 respectively. Considering an abstraction where ages and salaries of the employees are abstracted by the elements from the domain of intervals, and the number of children in the attribute 'Child-no' are abstracted by the abstract values from the abstract domain $D^{abs}_{Child-no}$=$\{\bot, Zero, Few, Medium, Many, \top\}$ where $\top$ represents "any" and $\bot$ represents "undefined". The abstract table $t^{\sharp}_{emp}$ corresponding to t_{emp} w.r.t. these abstractions is shown in Table 3.7. Observe that the number of abstract tuples in an abstract database may be less than the number of tuples in the corresponding concrete database if the primary key is abstracted. The correspondence between concrete and abstract values of the attribute, for instance, 'Child-no' can be formally expressed by the abstraction and concretization functions $\alpha_{child-no}$ and $\gamma_{child-no}$ respectively as follows:*

$$\alpha_{child-no}(X) \triangleq \begin{cases} \bot & \text{if } X = \emptyset \\ Zero & \text{if } X = \{0\} \\ Few & \text{if } \forall x \in X:\ 1 \leq x \leq 2 \\ Medium & \text{if } \forall x \in X:\ 3 \leq x \leq 4 \\ many & \text{if } \forall x \in X:\ 5 \leq x \leq 10 \\ \top & \text{otherwise} \end{cases}$$

$$\gamma_{child-no}(y) \triangleq \begin{cases} \emptyset & \text{if } y{=}\bot \\ \{0\} & \text{if } y{=}Zero \\ \{x:\ 1 \le x \le 2\} & \text{if } y{=}Few \\ \{x:\ 3 \le x \le 4\} & \text{if } y{=}Medium \\ \{x:\ 5 \le x \le 10\} & \text{if } y{=}Many \\ \{x:\ 0 \le x \le 10\} & \text{if } y{=}\top \end{cases}$$

We can similarly define the abstraction-concretization functions for other attributes as well. The corresponding abstract lattices for the attributes 'Age', 'Sal' and 'Child-no' are shown in figure 3.1(a), 3.1(b) and 3.1(c) respectively.

Table 3.7: Abstract table $t^{\sharp}_{emp}$

$eID^{\sharp}$	$Name^{\sharp}$	$Age^{\sharp}$	$Dno^{\sharp}$	$Pno^{\sharp}$	$Sal^{\sharp}$	$Child-no^{\sharp}$
1	Matteo	[25,59]	2	1	[1500,2499]	Medium
2	Alice	[12,24]	1	2	[1500,2499]	Few
3	Joy	[25,59]	2	3	[1500,2499]	Medium
4	luca	[5,11]	1	2	[1500,2499]	Few
5	Deba	[25,59]	3	4	[2500,10000]	Many
6	Andrea	[60,100]	1	2	[1500,2499]	Few
7	Alberto	[12,24]	3	4	[500,1499]	Few
8	Bob	[12,24]	2	3	[2500,10000]	Medium

Definition 5 (Abstract Database) *Let dB be a database. The database $dB^{\sharp} = \alpha(dB)$ where α is an abstraction function, is said to be an abstract version of dB if there exists a representation function γ, called concretization function, such that for all tuple $\langle x_1, x_2, \ldots, x_n\rangle \in dB$ there exists a tuple $\langle y_1, y_2, \ldots, y_n\rangle \in dB^{\sharp}$ such that $\forall i \in [1 \ldots n],\ \ x_i \in id(y_i) \vee x_i \in \gamma(y_i)$.*

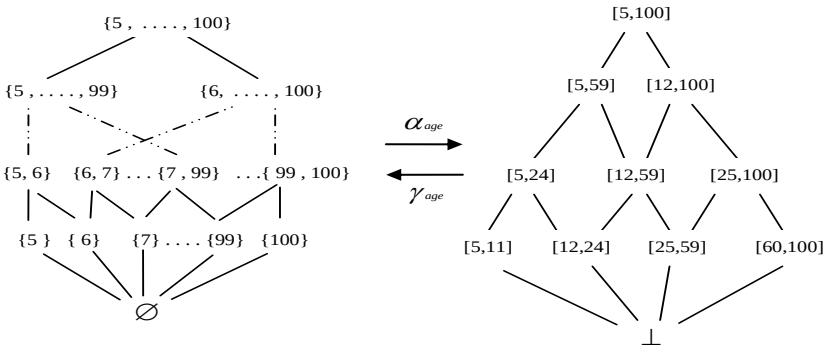
Syntax and Semantics of Statements in Abstract Domain

We now define the syntax and semantics of the applications embedding SQL statements in an abstract domain. We denote by the apex $^{\sharp}$, the syntactic elements of the abstract semantics. For each concrete element z, whenever we use the syntax $z^{\sharp}$, this means that there is a monotonic representation function γ from an abstract to the concrete domain such that $z \sqsubseteq \gamma(z^{\sharp})$.

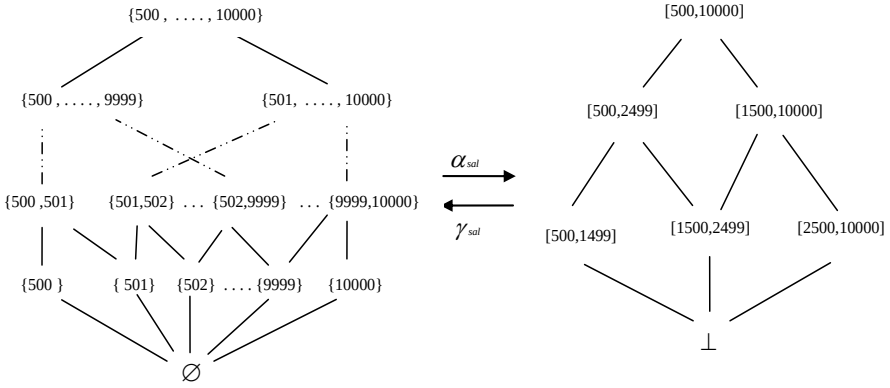
The syntax of SQL statement $Q^{\sharp}$ and SQL action $A^{\sharp}$ over an abstract domain corresponding to the concrete SQL statement Q_{sql} and action A_{sql} represented as below:

Figure 3.1: Abstract Lattices for attributes '*Age*', '*Sal*' and '*Child-no*'

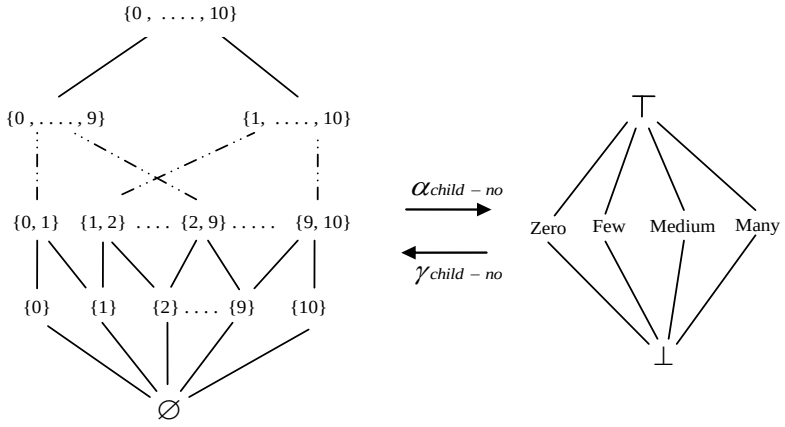
(a) Abstract lattice for attribute '*Age*'



(b) Abstract lattice for attribute '*Sal*'



(c) Abstract lattice for attribute '*Child − no*'



$Q^\sharp ::= \langle A^\sharp, \phi^\sharp \rangle \mid Q_1^\sharp \ \texttt{UNION}^\sharp \ Q_2^\sharp \mid Q_1^\sharp \ \texttt{INTERSECT}^\sharp \ Q_2^\sharp \mid Q_1^\sharp \ \texttt{MINUS}^\sharp \ Q_2^\sharp$

$A^\sharp ::= select^\sharp(v_a^\sharp, f^\sharp(\vec{e'^\sharp}), r^\sharp(\vec{h^\sharp}(\vec{x^\sharp})), \phi^\sharp, g^\sharp(\vec{e^\sharp})) \mid update^\sharp(\vec{v_d^\sharp}, \vec{e^\sharp}) \mid insert^\sharp(\vec{v_d^\sharp}, \vec{e^\sharp}) \mid delete^\sharp(\vec{v_d^\sharp})$

Arithmetic expressions over abstract domain are defined as expected, whereas boolean expressions are evaluated into a 3-valued logics $\{true, false, \top\}$, where $\top$ means "*either true or false*".

$c^\sharp ::= n^\sharp \mid k^\sharp$
$e^\sharp ::= c^\sharp \mid v_d^\sharp \mid v_a^\sharp \mid op^\sharp \ e^\sharp \mid e_1^\sharp \ op^\sharp \ e_2^\sharp$, where $op^\sharp$ represents abstract arithmetic operator.
$b^\sharp ::= e_1^\sharp \ op_r^\sharp \ e_2^\sharp \mid \neg b^\sharp \mid b_1^\sharp \ \vee \ b_2^\sharp \mid b_1^\sharp \ \wedge \ b_2^\sharp \mid true \mid false \mid \top$, where $op_r^\sharp$ represents abstract relational operator.

Abstract elements in abstract pre-condition $\phi^\sharp$ are defined as follows:

$\tau^\sharp ::= c^\sharp \mid v_a^\sharp \mid v_d^\sharp \mid f_n^\sharp(\tau_1^\sharp, \tau_2^\sharp, ..., \tau_n^\sharp)$, where $f_n^\sharp$ is an abstract *n-ary* function.
$a_f^\sharp ::= R_n^\sharp(\tau_1^\sharp, \tau_2^\sharp, ..., \tau_n^\sharp) \mid \tau_1^\sharp = \tau_2^\sharp$, where $R_n^\sharp$ is an abstract *n-ary* relation: $R_n^\sharp(\tau_1^\sharp, \tau_2^\sharp, ..., \tau_n^\sharp) \in \{true, false, \top\}$.
$\phi^\sharp := a_f^\sharp \mid \neg\phi_1^\sharp \mid \phi_1^\sharp \vee \phi_2^\sharp \mid \phi_1^\sharp \wedge \phi_2^\sharp \mid \forall x_i^\sharp \ \phi_1^\sharp \mid \exists x_i^\sharp \ \phi_1^\sharp$

Different abstract functions involved in $A^\sharp$ are shown below:

$g^\sharp ::= \texttt{GROUP BY}^\sharp \mid id$
$r^\sharp ::= \texttt{DISTINCT}^\sharp \mid \texttt{ALL}^\sharp$
$s^\sharp ::= \texttt{AVG}^\sharp \mid \texttt{SUM}^\sharp \mid \texttt{MAX}^\sharp \mid \texttt{MIN}^\sharp \mid \texttt{COUNT}^\sharp$
$h^\sharp(e^\sharp) ::= s^\sharp \circ r^\sharp(e^\sharp) \mid \texttt{DISTINCT}^\sharp(e^\sharp) \mid id$
$h^\sharp(*) ::= \texttt{COUNT}^\sharp(*)$
$f^\sharp ::= \texttt{ORDER BY ASC}^\sharp \mid \texttt{ORDER BY DESC}^\sharp \mid id$

Instructions over an abstract domain are defined as follows:

$I^\sharp ::= skip \mid v_a^\sharp := e^\sharp \mid v_a^\sharp :=? \mid Q^\sharp \mid if \ b^\sharp \ then \ I_1^\sharp \ else \ I_2^\sharp \mid while \ b^\sharp \ do \ I^\sharp \mid I_1^\sharp ; I_2^\sharp$

In the subsequent sections, we define abstract syntactic functions appearing in various abstract SQL statements so as to preserve the soundness in an abstract domain of interest. This way, we prove the soundness of abstract SQL statements with respect to their concrete counter-part. The soundness and completeness of an abstract function $f^\sharp$ *w.r.t.* its concrete version f is defined in Definition 6.

Definition 6 (Soundness & Completeness) *Let γ be a concretization function from an abstract domain to a concrete one. The soundness and completeness conditions for an abstract functions $f^\sharp$ with respect to the corresponding concrete function f are,*

$$f^\sharp \text{ is sound if } \gamma \circ f^\sharp \sqsupseteq f \circ \gamma$$

$$f^\sharp \text{ is complete if } \gamma \circ f^\sharp = f \circ \gamma$$

Abstract Pre-conditions

The pre-condition ϕ in Q_{sql} follows first order logic, and are defined by the n-ary function f_n on constants and variables. Soundness (and completeness eventually) of its abstract version $f_n^\sharp$ rely on the local correctness of the operations in the abstract domain. For example, consider an abstract domain for parity represented by $PAR = \{\top, even, odd, \bot\}$. The '$\times$' operation over the concrete domain is mapped to its abstract version as follows: $odd(\times^\sharp)odd = odd$, $even(\times^\sharp)odd = even$, and $even(\times^\sharp)even = even$. Similarly, in case of abstract domain of sign represented by $SIGN = \{\top, +, -, \bot\}$, the corresponding operation would be $-(\times^\sharp)- = +$, $+(\times^\sharp)- = -$, and $+(\times^\sharp)+ = +$.

Given a set of terms $\{\tau_1, \cdots, \tau_n\}$, the relation $R_n(\tau_1, \cdots, \tau_n)$ appearing in ϕ results into either *true* or *false*. However, an abstract relation $R_n^\sharp(\tau_1^\sharp, \cdots, \tau_n^\sharp)$ corresponding to R_n follows three valued logic $\{true, false, \top\}$, where $\top$ represents either *true* or *false*. The correspondence between the relation R_n and its abstract version $R_n^\sharp$ should guarantee that, if $R_n^\sharp(\tau_1^\sharp, \cdots, \tau_n^\sharp)$ is true, then $\forall \tau_1 \in \gamma(\tau_1^\sharp), \cdots, \tau_n \in \gamma(\tau_n^\sharp) : R_n(\tau_1, \cdots, \tau_n)$ is true and if $R_n^\sharp(\tau_1^\sharp, \cdots, \tau_n^\sharp)$ is false, then $\forall \tau_1 \in \gamma(\tau_1^\sharp), \cdots, \tau_n \in \gamma(\tau_n^\sharp) : R_n(\tau_1, \cdots, \tau_n)$ is false. For instance, if we consider the binary relation '$>$' among integers, its abstract version '$>^\sharp$' on the domain of intervals is defined as follows:

$$[l_i, h_i] >^\sharp [l_j, h_j] \triangleq \begin{cases} \text{true} & \text{if } l_i > h_j \\ \text{false} & \text{if } l_j \geq h_i \\ \top & \text{otherwise} \end{cases}$$

Similarly, '$\geq^\sharp$' is defined as:

$$[l_i, h_i] \geq^\sharp [l_j, h_j] \triangleq \begin{cases} \text{true} & \text{if } l_i \geq h_j \\ \text{false} & \text{if } h_i < l_j \\ \top & \text{otherwise} \end{cases}$$

Thus, abstract pre-condition $\phi^\sharp$ appearing in $Q^\sharp$ identifies the set of active data from abstract database for which it evaluates to either *true* or $\top$.

Example 2 *Consider the database of Figure 3.2 containing a concrete table t_{emp} and consider the following SELECT statement:*

$$Q_1 = \texttt{SELECT}\ Age,\ Dno,\ Sal\ \texttt{FROM}\ t_{emp}\ \texttt{WHERE}\ Sal > 1600$$

If we execute Q_1 on t_{emp}, we get the result ξ_1 shown in Table 3.8.

Table 3.8: ξ_1: Result of Q_1 (concrete)

Age	*Dno*	*Sal*
30	2	2000
50	2	2300
10	1	1700
40	3	3000
70	1	1900
14	2	4000

The abstract version of Q_1 (denoted $Q_1^\sharp$), using the abstract mapping α defined in Example 1, is shown below:

$$Q_1^\sharp = \texttt{SELECT}^\sharp\ Age^\sharp,\ Dno^\sharp,\ Sal^\sharp\ \texttt{FROM}\ t_{emp}^\sharp\ \texttt{WHERE}\ Sal^\sharp >^\sharp [1500, 2499]$$

The result of $Q_1^\sharp$ on the abstract table $t_{emp}^\sharp$ (Table 3.7) is depicted in Table 3.9. Observe that one row corresponding to $eID^\sharp = 7$ has been disregarded because the abstract well formed formula $[500, 1499] \geq^\sharp [1500, 2499]$ does not satisfy the semantic structure $\varsigma^\sharp$, as $1500 \geq 1499$ is true. Soundness is preserved, i.e. $\xi_1 \in \gamma(\xi_1^\sharp)$, as we include the rows (in this example, the row corresponding to $eID^\sharp = 2$) where the evaluation of the relation $\geq^\sharp$ yields $\top$; this might introduce inaccuracies, of course, in the abstract calculus, that results into a sound overapproximation of the concrete one.

Table 3.9: $\xi_1^\sharp$: Result of $Q_1^\sharp$ (abstract)

$Age^\sharp$	$Dno^\sharp$	$Sal^\sharp$
[25,59]	2	[1500,2499]
[12,24]	1	[1500,2499]
[25,59]	2	[1500,2499]
[5,11]	1	[1500,2499]
[25,59]	3	[2500,10000]
[60,100]	1	[1500,2499]
[12,24]	2	[2500,10000]

Abstract Syntactic Functions in Abstract SELECT Statements

We now describe the correspondence between concrete and abstract functions involved in SELECT statement. Observe that many of these abstract functions differ from the corresponding concrete ones only on the domain and range, while their functionality are the same.

Abstract GROUP BY Function: We denote by g the `GROUP BY` function in SELECT statement. The function $g(\vec{e})[t]$ where $\vec{e}$ represents an ordered sequence of arithmetic expressions, is applied on a table t and depending on the values of $\vec{e}$ over the tuples of the table t, it results into maximal partitions of the tuples in t. The tuples in the same partition will have the same values for $\vec{e}$, whereas the tuples in different partitions will have different values for $\vec{e}$. The `GROUP BY` function g is identity function id when no `GROUP BY` clause is present in SELECT statement. The function g and its abstract version $g^\sharp$ are shown below:

$$\begin{aligned} g &::= \texttt{GROUP BY} \mid id \\ g^\sharp &::= \texttt{GROUP BY}^\sharp \mid id \end{aligned}$$

Abstract `GROUP BY` function $g^\sharp$ works in the similar way, but it is applied on abstract tables $t^\sharp$, instead of concrete ones. It partitions the abstract tuples of $t^\sharp$ based on the abstract values of $\vec{e^\sharp}$ over the tuples.

Lemma 1 *Let γ be a concretization function. The abstract `GROUP BY` function $g^\sharp$ is sound with respect to γ, i.e. $\gamma \circ g^\sharp \sqsupseteq g \circ \gamma$, where g is the concrete counter-part of $g^\sharp$.*

Proof Let $t^\sharp$ be an abstract table and $\vec{e^\sharp}$ be an ordered sequence of abstract expressions. Let $t \in \gamma(t^\sharp)$ and $\vec{e} \in \gamma(\vec{e^\sharp})$, where γ is the concretization function.

Suppose $\{l_1, l_2, \ldots, l_n\}$ is a set of concrete partitions obtained from $g(\vec{e})[t]$, whereas $\{s_1, s_2, \ldots, s_m\}$ is the set of abstract partitions obtained from $g^\sharp(\vec{e^\sharp})[t^\sharp]$.

To prove the soundness of $g^\sharp$, we have to show that

$$\forall l_i,\ \exists s_j :\ l_i \subseteq \gamma(s_j)\ and\ m \leq n$$

Consider a concrete partition $l_i \subseteq t$. From the Definition 5, we know that $\forall x \in t,\ \exists y \in t^\sharp : x \in \gamma(y)$. Thus, we have

$$\forall x_{i1}, x_{i2} \in l_i,\ \ \exists y^\sharp_{j1}, y^\sharp_{j2} \in t^\sharp :\ x_{i1} \in \gamma(y^\sharp_{j1}) \wedge x_{i2} \in \gamma(y^\sharp_{j2}) \tag{3.4}$$

We know that the values of $\vec{e}$ for all tuples in a partition are same, *i.e.*,

$$\forall x_{i1}, x_{i2} \in l_i,\ \ \pi_{\vec{e}}(x_{i1}) = \pi_{\vec{e}}(x_{i2})$$

By the def. of abstraction, we get

$$\alpha(\pi_{\vec{e}}(x_{i1})) = \alpha(\pi_{\vec{e}}(x_{i2})) \text{ where } \alpha \text{ is abstraction function.} \tag{3.5}$$

From equation 3.4 and 3.5, we can write

$$\pi_{\vec{e^\sharp}}(y^\sharp_{j1}) = \pi_{\vec{e^\sharp}}(y^\sharp_{j2}) \tag{3.6}$$

Equation 3.6 says that $y^\sharp_{j1}$ and $y^\sharp_{j2}$ belongs to the same partition $s_j \subseteq t^\sharp$, as the properties of $\vec{e^\sharp}$ in $y^\sharp_{j1}$ and $y^\sharp_{j2}$ are same. Therefore,

$$\forall x_{i1}, x_{i2} \in l_i \subseteq t,\ \exists y^\sharp_{j1}, y^\sharp_{j2} \in s_j \subseteq t^\sharp :\ x_{i1} \in \gamma(y^\sharp_{j1}), x_{i2} \in \gamma(y^\sharp_{j2})$$

or,

$$\forall l_i,\ \exists s_j :\ l_i \subseteq \gamma(s_j)$$

Since an abstraction function α might be surjective, two different concrete partitions l_i and l_j $(i \neq j)$ might be mapped into the same abstract partition s_k, if

$$x_i \in l_i,\ x_j \in l_j,\ i \neq j :\ \alpha(\pi_{\vec{e}}(x_i)) = \alpha(\pi_{\vec{e}}(x_j))$$

Thus, the number of abstract partitions is less than or equal to the number of concrete partitions, *i.e.*, $m \leq n$.

Abstract ALL and Abstract DISTINCT: SELECT statement sometimes uses `DISTINCT` or `ALL` clause denoted by the function r which deals with duplicate tuples or duplicate values of expressions. Its abstract version $r^\sharp$ also works similarly, *i.e.*, deals with the duplicate elements in the list of abstract tuples or abstract values of expressions. The concrete function r and its abstract version are shown below:

$$\begin{aligned} r &::= \texttt{DISTINCT} \mid \texttt{ALL} \\ r^\sharp &::= \texttt{DISTINCT}^\sharp \mid \texttt{ALL}^\sharp \end{aligned}$$

Lemma 2 *Let γ be a concretization function. $\texttt{ALL}^\sharp$ is complete, i.e. $\gamma \circ \texttt{ALL}^\sharp = \texttt{ALL} \circ \gamma$.*

Proof When applying $\texttt{ALL}^\sharp$ to a list of abstract tuples $\langle l_i^\sharp : i \in I \rangle$, none of the tuple is removed or modified, and the same holds for `ALL`. Therefore,

$$\begin{aligned} & \gamma \circ \texttt{ALL}^\sharp(\langle l_i^\sharp : i \in I \rangle) \\ =& \gamma(\texttt{ALL}^\sharp(\langle l_i^\sharp : i \in I \rangle)) \\ =& \gamma(\langle l_i^\sharp : i \in I \rangle) \\ =& \langle \gamma(l_i^\sharp) : i \in I \rangle \\ =& \texttt{ALL}(\langle \gamma(l_i^\sharp) : i \in I \rangle) \\ =& \texttt{ALL}(\gamma(\langle l_i^\sharp : i \in I \rangle)) \\ =& \texttt{ALL} \circ \gamma(\langle l_i^\sharp : i \in I \rangle) \end{aligned}$$

Lemma 3 *If γ is injective, then $\texttt{DISTINCT}^\sharp$ is complete, i.e. $\gamma \circ \texttt{DISTINCT}^\sharp = \texttt{DISTINCT} \circ \gamma$.*

Proof Suppose, after applying $\texttt{DISTINCT}^\sharp$ function on an abstract table $t^\sharp$, we get the abstract table $t_u^\sharp$ containing only unique rows. That means,

$$\forall l_1^\sharp, l_2^\sharp \in t_u^\sharp,\ \exists a^\sharp \in attr(t_u^\sharp) :\ \pi_{a^\sharp}(l_1^\sharp) \neq \pi_{a^\sharp}(l_2^\sharp)$$

In other words, any two rows in $t_u^\sharp$ differ by the property in at least one attribute position. Thus, concretization of $t_u^\sharp$ results into a concrete table t_u containing unique rows only, as γ is injective.

If we first apply γ on $t^\sharp$ before applying $\texttt{DISTINCT}^\sharp$, it results into a concrete table t containing duplicate rows if $t^\sharp$ has duplicate abstract rows. But after applying `DISTINCT` on t, we always get the same concrete table t_u. Therefore, the function $\texttt{DISTINCT}^\sharp$ is complete if γ is injective.

Abstract ORDER BY Function: We denote by f the `ORDER BY` function appearing in SELECT statement. The operation $f(\vec{e})[t]$ sorts the tuples of the table t in ascending or descending order based on the values of $\vec{e}$ over those tuples. An abstract version $f^\sharp$ also works in similar way, but it is applied on abstract tables $t^\sharp$ and sorts the abstract tuples in ascending or descending order based on the abstract values of $\vec{e^\sharp}$ over the tuples in $t^\sharp$. The concrete functions f and their abstract versions are defined as:

$$\begin{array}{ll} f & ::= \texttt{ORDER BY ASC} \mid \texttt{ORDER BY DESC} \mid id \\ f^\sharp & ::= \texttt{ORDER BY ASC}^\sharp \mid \texttt{ORDER BY DESC}^\sharp \mid id \end{array}$$

Lemma 4 *If the representation function γ is monotone and injective, the functions $f^\sharp$ above are complete, i.e. $\gamma \circ f^\sharp = f \circ \gamma$*

Proof Given an abstract table $t^\sharp$ and an ordered sequence of abstract expressions $\vec{e^\sharp}$. Suppose for two tuples $l_i^\sharp, l_j^\sharp \in t^\sharp$, we have

$$\pi_{\vec{e^\sharp}}(l_i^\sharp) > \pi_{\vec{e^\sharp}}(l_j^\sharp) \tag{3.7}$$

Suppose $f^\sharp$::= `ORDER BY ASC`$^\sharp$. Therefore, application of $f^\sharp$ sorts them in ascending order denoted by the ordered list $\langle l_j^\sharp, l_i^\sharp \rangle$. Since γ is injective, the concretization of this ordered list of abstract tuples always yield to an ordered list of concrete tuples denoted by $\langle l_j, l_i \rangle$, where $l_i \in \gamma(l_i^\sharp)$ and $l_j \in \gamma(l_j^\sharp)$.

Since concretization function γ is monotone, it preserves the ordering while mapping from abstract domain to concrete co-domain. Thus, from equation 3.7 we get

$$\gamma(\pi_{\vec{e^\sharp}}(l_i^\sharp)) > \gamma(\pi_{\vec{e^\sharp}}(l_j^\sharp))$$

or,

$$\pi_{\vec{e}}(l_i) > \pi_{\vec{e}}(l_j) \tag{3.8}$$

where, $\vec{e} \in \gamma(\vec{e^\sharp})$ and $l_i \in \gamma(l_i^\sharp)$ and $l_j \in \gamma(l_j^\sharp)$.

From equation 3.8, we get that the application of f (::= `ORDER BY ASC`) on l_i and l_j also yield to the same ordered list of concrete tuples *i.e.* $\langle l_j, l_i \rangle$.

Thus, $\gamma \circ f^\sharp$ will result into the same order of the elements as obtained by function $f \circ \gamma$. Hence, $f^\sharp$ is complete if γ is monotone and injective.

Abstract Aggregate Functions: In section 3.3, we mentioned that the ordered sequence of functions $\vec{h}(\vec{e})$ where $\vec{h} \neq \vec{id}$, are applied on each partition obtained by `GROUP BY` function g, or on a single partition containing tuples for which pre-condition ϕ evaluates to *true* when no `GROUP BY` function is used. After performing $\vec{h}(\vec{e})$ on each partition, it results into a single concrete tuple.

The aggregate functions `MAX`, `MIN`, `AVG`, `SUM`, `COUNT` appear as $h_i \in \vec{h}$, and are denoted by s. Aggregate functions return a single value when applied on a group of values.

Similarly, abstract aggregate functions $s^\sharp$ are applied on a set of abstract values, resulting into a single abstract value. The concrete aggregate functions s and its abstract version $s^\sharp$ are defined below:

$$\begin{array}{ll} s ::= & \texttt{AVG} \mid \texttt{SUM} \mid \texttt{MAX} \mid \texttt{MIN} \mid \texttt{COUNT} \\ s^\sharp ::= & \texttt{AVG}^\sharp \mid \texttt{SUM}^\sharp \mid \texttt{MAX}^\sharp \mid \texttt{MIN}^\sharp \mid \texttt{COUNT}^\sharp \end{array}$$

We now define the abstract aggregate functions aiming at preserving the soundness of them *w.r.t.* their corresponding concrete aggregate functions.

Given a set of concrete numerical values $X = \{a_1, a_2, \ldots, a_n\}$, the concrete aggregate functions s are defined as follows:

$$\texttt{AVG(X)} \triangleq \frac{\sum_1^n a_i}{n}$$

$$\texttt{SUM(X)} \triangleq \sum_1^n a_i$$

$$\texttt{MAX(X)} \triangleq a_i \in X, \text{where } \forall j \in [1..n], i \neq j : \ a_i \geq a_j$$

$$\texttt{MIN(X)} \triangleq a_i \in X, \text{where } \forall j \in [1..n], i \neq j : \ a_i \leq a_j$$

$$\texttt{COUNT(X)} \triangleq \#X, \text{where \# denotes the cardinality of the set.}$$

Corresponding to each s, we consider a concrete function fn equivalent to s, that is, $fn(X) \equiv s(X)$. The function fn and its abstract versions $fn^\sharp$ corresponding to the aggregate functions are defined as follows:

$$\begin{array}{ll} fn ::= & \texttt{average} \mid \texttt{summation} \mid \texttt{maximum} \mid \texttt{minimum} \mid \texttt{count} \\ fn^\sharp ::= & \texttt{average}^\sharp \mid \texttt{summation}^\sharp \mid \texttt{maximum}^\sharp \mid \texttt{minimum}^\sharp \mid \texttt{count}^\sharp \end{array}$$

For instance, let $X^\sharp$ be a set of abstract values from the domain of intervals, *i.e.*, $X^\sharp = \{\ [l_i, h_i] \mid i \in [1...n],\ l_i, h_i \in \mathbb{Z};\ l_i \leq h_i\ \}$. Let us denote $L = \{l_i \mid [l_i, h_i] \in X^\sharp\}$ and $H = \{h_i \mid [l_i, h_i] \in X^\sharp\}$. The abstract functions $fn^\sharp$ on $X^\sharp$ are defined as follows:

$$\texttt{average}^\sharp(X^\sharp) \triangleq \texttt{[average(L), average(H)]}$$

$$\texttt{summation}^\sharp(X^\sharp) \triangleq \texttt{[summation(L), summation(H)]}$$

$$\texttt{maximum}^\sharp(X^\sharp) \triangleq [\texttt{maximum(L), maximum(H)}]$$

$$\texttt{minimum}^\sharp(X^\sharp) \triangleq [\texttt{minimum(L), minimum(H)}]$$

$$\texttt{count}^\sharp(X^\sharp) \triangleq [\texttt{count(L), count(H)}]$$

Formally, $fn^\sharp(X^\sharp) = [fn(L), fn(H)]$

We already know that in abstract domain we select only those tuples for which the abstract pre-condition $\phi^\sharp$ evaluates to either *true* or $\top$. Thus, unlike concrete domain, the abstract groups on which abstract aggregate functions are applied contain a set of tuples that yield $\phi^\sharp$ to either *true* or $\top$. Let us denote by $G^\sharp$ an abstract group containing a set of abstract tuples. We can partition $G^\sharp$ into two parts: $G^\sharp_{yes}$ for which $\phi^\sharp$ evaluates to *true*, and $G^\sharp_{may}$ for which $\phi^\sharp$ evaluates to $\top$. Thus, we can write $G^\sharp = G^\sharp_{yes} \cup G^\sharp_{may}$. Observe that $G^\sharp_{yes} \cap G^\sharp_{may} = \emptyset$.

To ensure the soundness, the computation of abstract aggregate functions $s^\sharp$ on $G^\sharp$ are defined as follows: the result of $s^\sharp(e^\sharp)$ on $G^\sharp$ is denoted by an interval as below:

$$s^\sharp(e^\sharp)[G^\sharp] = [\texttt{min}^\sharp(a^\sharp), \texttt{max}^\sharp(b^\sharp)]$$

where

$$a^\sharp = fn^\sharp(e^\sharp)[G^\sharp_{yes}] \ \textit{and} \ \ b^\sharp = fn^\sharp(e^\sharp)[G^\sharp]$$

By $fn^\sharp(e^\sharp)[G^\sharp_{yes}]$, we mean that function $fn^\sharp$ is applied on the set of abstract values obtained by evaluating $e^\sharp$ over the tuples in $G^\sharp_{yes}$, yielding a single abstract value as result. Similarly in $fn^\sharp(e^\sharp)[G^\sharp]$, $fn^\sharp$ is applied on the set of abstract values obtained by evaluating $e^\sharp$ over the tuples in $G^\sharp = G^\sharp_{yes} \cup G^\sharp_{may}$. The computation of $fn^\sharp$ is defined differently by considering two different situations that can arise: (*i*) when the primary key is abstracted, yielding two or more tuples mapped into a single abstract tuple, and (*ii*) when the primary key is not abstracted and the identity of each tuples are preserved in abstract domain.

Both the functions $\texttt{min}^\sharp$ and $\texttt{max}^\sharp$ takes as parameter a single abstract value $a^\sharp$ and $b^\sharp$ respectively obtained from $fn^\sharp$, and returns a concrete numerical value as output. $\texttt{min}^\sharp(a^\sharp)$ returns the minimum concrete value from $\gamma(a^\sharp)$, whereas $\texttt{max}^\sharp(b^\sharp)$ returns the maximum concrete value from $\gamma(b^\sharp)$, where γ is the concretization function.

Example 3 *Consider the database of Table 3.2 containing the concrete table t_{emp} and the following SELECT statement:*

$$Q_2 = \texttt{SELECT}\ \texttt{AVG}(Age),\ Dno,\ \texttt{MAX}(Sal),\ \texttt{COUNT}(*)\ \texttt{FROM}\ t_{emp}\ \texttt{WHERE}\ Sal \geq 1500\ \texttt{GROUP BY}\ Dno$$

If we execute Q_2 on t_{emp}, we get result ξ_2 shown in Table 3.10.

Table 3.10: ξ_2: Result of Q_2 (concrete)

AVG(Age)	*Dno*	*MAX(Sal)*	*COUNT(*)*
34	1	1900	3
31.33	2	4000	3
40	3	3000	1

The abstract version of Q_2 i.e. $Q_2^\sharp$, using the abstract mapping α defined in Example 1, is defined as below:

$$\begin{aligned} Q_2^\sharp \quad = \quad & \texttt{SELECT}^\sharp\ \texttt{AVG}^\sharp(Age^\sharp),\ Dno^\sharp,\ \texttt{MAX}^\sharp(Sal^\sharp),\ \texttt{COUNT}^\sharp(*)\ \texttt{FROM}\ t_{emp}^\sharp \\ & \texttt{WHERE}\ Sal^\sharp \geq^\sharp [1500, 2499]\ \texttt{GROUP BY}^\sharp\ Dno^\sharp \end{aligned}$$

The result of $Q_2^\sharp$ on $t_{emp}^\sharp$ is shown in Table 3.11. Observe that, $G_{yes}^\sharp = \emptyset$ because $\phi^\sharp$ evaluates to $\top$ for all abstract tuples in the group with $Dno^\sharp$=1. Thus, $\texttt{AVG}^\sharp(Age^\sharp)$ is computed as follows:

$$a^\sharp = \texttt{average}^\sharp(\emptyset) = NULL$$

$$b^\sharp = \texttt{average}^\sharp([12, 24], [5, 11], [60, 100]) = [25.66, 45]$$

Since $\texttt{min}^\sharp(a^\sharp)$ and $\texttt{max}^\sharp(b^\sharp)$ return minimum value from $\gamma(a^\sharp)$ and maximum value from $\gamma(b^\sharp)$ respectively, we get $\texttt{min}^\sharp(a^\sharp)$=$\texttt{min}^\sharp(NULL)$=0 and $\texttt{max}^\sharp(b^\sharp)$=$\texttt{max}^\sharp([25.66, 45])$=45. Thus, for group with $Dno^\sharp$=1, $\texttt{AVG}^\sharp(Age^\sharp)$=[$\texttt{min}^\sharp(a^\sharp)$, $\texttt{max}^\sharp(b^\sharp)$]=[0, 45].

Similarly, for the group with $Dno^\sharp$=2, first two tuples belong to $G_{may}^\sharp$, whereas last tuple belongs to $G_{yes}^\sharp$. Thus, $\texttt{MAX}^\sharp(Sal^\sharp)$ is computed as follows:

$$a^\sharp = \texttt{maximum}^\sharp([2500, 10000]) = [2500, 10000]$$

$$b^\sharp = \texttt{maximum}^\sharp([1500,2499], [1500,2499], [2500, 10000]) = [2500, 10000]$$

Thus, for group with $Dno^\sharp$=2, $\texttt{MAX}^\sharp(Sal^\sharp)$=[$\texttt{min}^\sharp(a^\sharp)$, $\texttt{max}^\sharp(b^\sharp)$]=[2500,10000]. Observe that abstraction is sound i.e. $\xi_2 \in \gamma(\xi_2^\sharp)$.

Table 3.11: $\xi_2^\sharp$: Result of $Q_2^\sharp$ (abstract)

$AVG^\sharp(Age^\sharp)$	$Dno^\sharp$	$MAX^\sharp(Sal^\sharp)$	$COUNT^\sharp(*)$
[0, 45]	1	[0,2499]	[0, 3]
[12, 47.33]	2	[2500, 10000]	[1, 3]
[25, 59]	3	[2500, 10000]	[1, 1]

Lemma 5 *Let γ be a concretization function from the domain of intervals to a concrete numerical domain. The abstract functions $fn^\sharp$ are sound w.r.t. fn if they satisfy*

$$\gamma(fn^\sharp(X^\sharp)) \supseteq \{fn(X) \mid X \in \gamma(X^\sharp)\}$$

Proof Let $X^\sharp$ be a set of abstract values from the domain of intervals, *i.e.*,

$$X^\sharp = \{\ [l_i, h_i] \mid i \in [1..n];\ l_i, h_i \in \mathbb{Z};\ l_i \leq h_i\ \}$$

Consider two sets L and H, where $L = \{l_i \mid [l_i, h_i] \in X^\sharp\}$ and $H = \{h_i \mid [l_i, h_i] \in X^\sharp\}$. The abstract functions $fn^\sharp$ are defined on $X^\sharp$ as follows:

$$\texttt{average}^\sharp(X^\sharp) \triangleq [\texttt{average(L)},\ \texttt{average(H)}]$$

$$\texttt{summation}^\sharp(X^\sharp) \triangleq [\texttt{summation(L)},\ \texttt{summation(H)}]$$

$$\texttt{maximum}^\sharp(X^\sharp) \triangleq [\texttt{maximum(L)},\ \texttt{maximum(H)}]$$

$$\texttt{minimum}^\sharp(X^\sharp) \triangleq [\texttt{minimum(L)},\ \texttt{minimum(H)}]$$

$$\texttt{count}^\sharp(X^\sharp) \triangleq [\texttt{count(L)},\ \texttt{count(H)}]$$

Formally, we can write

$$fn^\sharp(X^\sharp) = [fn(L), fn(H)] \tag{3.9}$$

$$= [s(L), s(H)] \quad \textit{Since},\ fn \equiv s \tag{3.10}$$

For a given set of concrete numerical values $X = \{a_1, a_2, \ldots, a_n\}$, the functions $fn \equiv s$ are defined as:

$$\texttt{average(X)} \equiv \texttt{AVG(X)} \triangleq \frac{\Sigma_1^n a_i}{n}$$

$$\texttt{summation(X)} \equiv \texttt{SUM(X)} \triangleq \Sigma_1^n a_i$$

$$\texttt{maximum(X)} \equiv \texttt{MAX(X)} \triangleq a_i \in X, \text{ where } \forall j \in [1..n], i \neq j :\ a_i \geq a_j$$

$$\texttt{minimum(X)} \equiv \texttt{MIN(X)} \triangleq a_i \in X, \text{ where } \forall j \in [1..n], i \neq j :\ a_i \leq a_j$$

$$\texttt{count(X)} \equiv \texttt{COUNT(X)} \triangleq \#X, \text{ where \# denotes the cardinality of the set.}$$

Given two sets of numerical values $X = \{a_1, a_2, \ldots, a_n\}$ and $X' = \{a'_1, a'_2, \ldots, a'_n\}$. We say X is less than or equal to X' (denoted $X \sqsubseteq X'$) which is defined component-wise, *i.e.* if $\forall i \in [1..n],\ a_i \leq a'_i$, then $X \sqsubseteq X'$.

Since the functions fn are monotone, we get

$$\text{if } X \sqsubseteq X', \text{ then } fn(X) \leq fn(X') \tag{3.11}$$

Let $X = \{b_i \mid [l_i, h_i] \in X^\sharp,\ l_i \leq b_i \leq h_i\} \in \gamma(X^\sharp)$. Since, $\forall b_i \in X$ and $\forall [l_i, h_i] \in X^\sharp$: $l_i \leq b_i \leq h_i$, we can write:

$$\forall X \in \gamma(X^\sharp) :\ L \sqsubseteq X \sqsubseteq H$$

According to equation 3.11,

$$\forall X \in \gamma(X^\sharp) :\ fn(L) \leq fn(X) \leq fn(H) \tag{3.12}$$

From equation 3.9 and 3.12, we get

$$\forall X \in \gamma(X^\sharp) :\ fn(X) \in \gamma(fn^\sharp(X^\sharp))$$

or,

$$fn(\gamma(X^\sharp)) \subseteq \gamma(fn^\sharp(X^\sharp))$$

This implies that the abstract functions $fn^\sharp$ are sound *w.r.t.* fn.

Lemma 6 *Let γ be a concretization function from the domain of intervals to a concrete numerical domain. Abstract aggregate functions $s^\sharp$ are sound w.r.t. s, i.e.*

$$\forall X \in X^\sharp :\ s(X) \in \gamma(s^\sharp(X^\sharp))$$

Proof The computation of abstract aggregate functions $s^\sharp$ over a group of abstract tuples $G^\sharp = G_{yes} \cup G_{may}$ are defined as follows: $s^\sharp(e^\sharp)[G^\sharp]$ is denoted by an interval

$$s^\sharp(e^\sharp)[G^\sharp] = [\texttt{min}^\sharp(a^\sharp), \texttt{max}^\sharp(b^\sharp)]$$

where,

$$a^\sharp = fn^\sharp(e^\sharp)[G^\sharp_{yes}] \text{ and } b^\sharp = fn^\sharp(e^\sharp)[G^\sharp]$$

From Lemma 5, we have that $fn^\sharp$ is sound *w.r.t.* fn, and therefore,

$$\forall G_y \in \gamma(G^\sharp_{yes}) : \ a \in \gamma(a^\sharp)$$
$$or, \ a \geq \texttt{min}^\sharp(a^\sharp)$$

where, $a = fn(e)[G_y]$ and

$$\forall G \in \gamma(G^\sharp) : \ b \in \gamma(b^\sharp)$$
$$or, \ b \leq \texttt{max}^\sharp(b^\sharp)$$

where, $b = fn(e)[G]$.

We know that $G^\sharp = G_{yes} \cup G_{may}$ contains the abstract tuples for which $\phi^\sharp$ evaluates to either *true* and $\top$. Given an abstract tuple $t^\sharp \in G^\sharp$ and abstract pre-condition $\phi^\sharp$, for any concrete tuple $t \in \gamma(t^\sharp)$ the corresponding concrete pre-condition $\phi \in \gamma(\phi^\sharp)$ evaluated to either *true* or *false*, since we loose precision when moving from concrete to a domain of abstraction. Thus, $\forall G \in \gamma(G^\sharp)$ where $G^\sharp = G_{yes} \cup G_{may}$, we can write:

$$G = G_y \cup G'_y \cup G_f, \text{ where } G_y \in \gamma(G_{yes}) \text{ and } G'_y \cup G_f \in \gamma(G_{may})$$

where G_y and G'_y are the set of concrete tuples for which ϕ evaluates to *true*, and G_f is the set of concrete tuples for which ϕ evaluates to *false*.

Since the concrete aggregate functions s are always applied on $(G_y \cup G'_y)$ for which ϕ evaluates to *true*, we get

$$G_y \subseteq (G_y \cup G'_y) \subseteq G$$

From the monotonicity property of fn, we get

$$fn(e)[G_y] \leq fn(e)[G_y \cup G'_y] \leq fn(e)[G]$$

or,

$$fn(e)[G_y] \leq s(e)[G_y \cup G'_y] \leq fn(e)[G], \quad Since \ fn \equiv s$$

or,

$$a \leq s(X) \leq b$$

where, X is obtained by evaluating e over the tuples of $(G_y \cup G'_y)$.

or,

$$\texttt{min}^\sharp(a^\sharp) \leq s(X) \leq \texttt{max}^\sharp(b^\sharp)$$

or,

$$s(X) \in \gamma([\mathtt{min}^\sharp(a^\sharp), \mathtt{max}^\sharp(b^\sharp)])$$

or,

$$s(X) \in \gamma(s^\sharp(X^\sharp))$$

Thus, the abstract aggregate functions $s^\sharp$ are sound *w.r.t.* s.

Abstract UPDATE, INSERT, and DELETE Statements

The abstract semantics of UPDATE, INSERT and DELETE statements in an abstract domain of interest are defined below:

Abstract UPDATE Statement Let $Q_{update} = \langle update(\vec{v_d}, \vec{e}), \phi \rangle$ be an UPDATE statement with $target(Q_{update}) = t$. Let $Q^\sharp_{update} = \langle update^\sharp(\vec{v^\sharp_d}, \vec{e^\sharp}), \phi^\sharp \rangle$ and $t^\sharp$ be their abstract versions in an abstract domain corresponding to Q_{update} and t respectively, such that $target(Q^\sharp_{update}) = t^\sharp$. According to the abstract semantics of $Q^\sharp_{update}$, we get

$$S^\sharp[\![Q^\sharp_{update}]\!](\rho_{t^\sharp}, \rho_{a^\sharp}) = S^\sharp[\![\langle update^\sharp(\vec{v^\sharp_d}, \vec{e^\sharp}), \phi^\sharp \rangle]\!](\rho_{t^\sharp}, \rho_{a^\sharp}) = (\rho_{t^\sharp_1}, \rho_{a^\sharp})$$

where,

$$\rho_{t^\sharp_1}(x^\sharp) = \begin{cases} \rho_{t^\sharp \downarrow_T \phi^\sharp}(x^\sharp) \cup \rho_{t^\sharp \downarrow_U \phi^\sharp}(x^\sharp) \cup \rho_{t^\sharp \downarrow_F \phi^\sharp}(x^\sharp) & \text{if } x^\sharp \notin \vec{v^\sharp_d} \\ E^\sharp[\![e^\sharp_i]\!](\rho_{t^\sharp \downarrow_T \phi^\sharp}, \rho_{a^\sharp}) \cup (\sqcup(E^\sharp[\![e^\sharp_i]\!](\rho_{t^\sharp \downarrow_U \phi^\sharp}, \rho_{a^\sharp}), E^\sharp[\![x^\sharp]\!](\rho_{t^\sharp \downarrow_U \phi^\sharp}))) \cup \rho_{t^\sharp \downarrow_F \phi^\sharp}(x^\sharp) \\ \text{if } x^\sharp \text{ is the } i^{th} \text{ component of } \vec{v^\sharp_d} \text{ and } e^\sharp_i \text{ is the } i^{th} \text{ component of } \vec{e^\sharp} \end{cases}$$

By the notations $t^\sharp \downarrow_T \phi^\sharp$, $t^\sharp \downarrow_U \phi^\sharp$ and $t^\sharp \downarrow_F \phi^\sharp$ we denote the set of abstract tuples in $t^\sharp$ for which $\phi^\sharp$ evaluates to *true*, *unknown* and *false* respectively. The operator $\sqcup$ stands for computing least upper bound component-wise, *i.e.* $\sqcup(X^\sharp, Y^\sharp) = \{lub(x^\sharp_i, y^\sharp_i) \mid x^\sharp_i \in X^\sharp \wedge y^\sharp_i \in Y^\sharp\}$.

Abstract INSERT Statement Let $Q^\sharp_{insert} = \langle insert^\sharp(\vec{v^\sharp_d}, \vec{e^\sharp}), \phi^\sharp \rangle$ and $t^\sharp$ be an abstract INSERT statement and an abstract table corresponding to their concrete versions Q_{insert} and t respectively, such that $target(Q^\sharp_{insert}) = t^\sharp$. According to the abstract semantics of $Q^\sharp_{insert}$, we get

$$S^\sharp[\![Q^\sharp_{insert}]\!](\rho_{t^\sharp}, \rho_{a^\sharp}) = S^\sharp[\![\langle insert^\sharp(\vec{v^\sharp_d}, \vec{e^\sharp}), \phi^\sharp \rangle]\!](\rho_{t^\sharp}, \rho_{a^\sharp}) = (\rho_{t^\sharp_1}, \rho_{a^\sharp})$$

where,

$$\text{let } \vec{v_d^\sharp} = \langle a_1^\sharp, a_2^\sharp, \ldots, a_n^\sharp \rangle = attr(t^\sharp), \text{ and } E^\sharp[\![\vec{e^\sharp}]\!](\rho_{a^\sharp}) = \vec{r^\sharp} = \langle r_1^\sharp, r_2^\sharp, \ldots, r_n^\sharp \rangle,$$
$$\text{and } l_{new}^\sharp = \langle r_1^\sharp / a_1^\sharp, r_2^\sharp / a_2^\sharp, \ldots, r_n^\sharp / a_n^\sharp \rangle, \text{ and } \rho_{t_1^\sharp}(\vec{x^\sharp}) = \rho_{t^\sharp \cup l_{new}^\sharp}(\vec{x^\sharp})$$

Abstract DELETE Statement Given an abstract delete statement $Q_{delete}^\sharp = \langle delete^\sharp(\vec{v_d^\sharp}), \phi^\sharp \rangle$ with $target(Q_{delete}^\sharp) = t^\sharp$ corresponding to the concrete statement Q_{delete} and concrete table t respectively. According to the abstract semantics of $Q_{delete}^\sharp$, we get

$$S^\sharp[\![Q_{delete}^\sharp]\!](\rho_{t^\sharp}, \rho_{a^\sharp}) = S^\sharp[\![\langle delete^\sharp(\vec{v_d^\sharp}), \phi^\sharp \rangle]\!](\rho_{t^\sharp}, \rho_{a^\sharp}) = (\rho_{t_1^\sharp}, \rho_{a^\sharp})$$

where, $\rho_{t_1^\sharp}(x^\sharp) = \rho_{t^\sharp \downarrow_U \phi^\sharp}(\vec{x^\sharp}) \cup \rho_{t^\sharp \downarrow_F \phi^\sharp}(\vec{x^\sharp})$

Soundness of Abstract SQL Statements

Given an abstraction, let T and $T^\sharp$ be a concrete and abstract table respectively. The correspondence between T and $T^\sharp$ are described using the concretization and abstraction maps γ and α respectively. If Q_{sql} and $Q^\sharp$ are representing the SQL statements on concrete and abstract domain respectively, let T_{res} and $T_{res}^\sharp$ are the results of applying Q_{sql} and $Q^\sharp$ on the T and $T^\sharp$ respectively. The following fact illustrate the soundness condition of abstraction:

$$\begin{array}{ccc} T & \xrightarrow{Q_{sql}} & T_{res} \sqsubseteq \gamma(T_{res}^\sharp) \\ \uparrow \gamma & & \uparrow \gamma \\ T^\sharp & \xrightarrow[Q^\sharp]{} & T_{res}^\sharp \end{array}$$

Lemma 7 *Let $T^\sharp$ be an abstract table and $Q^\sharp$ be an abstract SQL statement. $Q^\sharp$ is sound if* $\forall T \in \gamma(T^\sharp).\ \forall Q_{sql} \in \gamma(Q^\sharp) : Q_{sql}(T) \subseteq \gamma(Q^\sharp(T^\sharp))$.

Proof The computation of an $Q^\sharp$ on $T^\sharp$ can be defined as the computation of the composite function formed from its syntactic functional components. Consider the following abstract SELECT statement

$$Q_{select}^\sharp = \langle select^\sharp(f^\sharp(\vec{e^\sharp}),\ r^\sharp(\vec{h^\sharp}(\vec{x^\sharp})),\ \phi_1^\sharp,\ g^\sharp(\vec{e^\sharp})), \phi^\sharp \rangle$$

and an abstract table $T^\sharp$ where $target(Q^\sharp_{select}) = T^\sharp$. We get the abstract result as follows:

$$\xi^\sharp = \text{func}^\sharp_{sel}[T^\sharp] = (f^\sharp(\vec{e'^\sharp}) \circ r^\sharp(\vec{h^\sharp}(\vec{x^\sharp})) \circ \phi^\sharp_1 \circ g^\sharp(\vec{e^\sharp}) \circ \phi^\sharp)[T^\sharp]$$

where $\text{func}^\sharp_{sel} = f^\sharp(\vec{e'^\sharp}) \circ r^\sharp(\vec{h^\sharp}(\vec{x^\sharp})) \circ \phi^\sharp_1 \circ g^\sharp(\vec{e^\sharp}) \circ \phi^\sharp$.
Let $Q_{select} \in \gamma(Q^\sharp_{select})$ and $T \in \gamma(T^\sharp)$. The computation of Q_{select} on T is defined as

$$\xi = \text{func}_{sel}[T] = (\ f(\vec{e'}) \circ r(\vec{h}(\vec{x})) \circ \phi_1 \circ g(\vec{e})) \circ \phi)[T]$$

where $\text{func}_{sel} = f(\vec{e'}) \circ r(\vec{h}(\vec{x})) \circ \phi_1 \circ g(\vec{e})) \circ \phi$.
We already proved that all syntactic abstract functional components in $Q^\sharp_{select}$ are sound with respect to their corresponding concrete counter-part. As the composition of sound abstract functions always yield to another sound abstract function, we get the abstract function $\text{func}^\sharp_{sel}$ is sound *w.r.t.* func_{sel}. Thus, $Q^\sharp_{select}$ is sound, *i.e.*, $\xi \in \gamma(\xi^\sharp)$. Similarly, we can prove the soundness for other SQL statements as well. Therefore, $\forall T \in \gamma(T^\sharp).\ \forall Q_{sql} \in \gamma(Q^\sharp) : Q_{sql}(T) \subseteq \gamma(Q^\sharp(T^\sharp))$.

Abstract UNION, INTERSECTION, MINUS Operations

Given an abstract query $Q^\sharp$, the result of it over an abstract database can be denoted by the tuple

$$\xi^\sharp = \langle \xi^\sharp_{yes}, \xi^\sharp_{may} \rangle$$

where $\xi^\sharp_{yes}$ is the part of the result for which semantic structure of $\phi^\sharp$ evaluates to *true* and $\xi^\sharp_{may}$ represents the remaining part for which $\phi^\sharp$ evaluates to $\top$[1].

Now we describe how to treat `UNION`, `INTERSECTION` and `MINUS` operation over an abstract domain so as to preserve the soundness.

Abstract UNION Operation: Let, $Q = Q_l$ `UNION` Q_r be a concrete query and dB be a concrete database. Let $\xi_l = S[\![Q_l]\!](\rho_{dB}, \rho_a)$ and $\xi_r = S[\![Q_r]\!](\rho_{dB}, \rho_a)$ be the result of the evaluation of Q_l and Q_r on dB respectively. Clearly, $\xi = S[\![Q]\!](\rho_{dB}, \rho_a) = \xi_l \cup \xi_r$.

When we move from a concrete to an abstract domain of interest, let $Q^\sharp_l$ and $Q^\sharp_r$ be the corresponding abstract versions of Q_l and Q_r respectively. Let $dB^\sharp$ be an abstract

[1]When a query uses aggregate functions $s^\sharp$, application of $s^\sharp$ over a group $G^\sharp$ yields a single row in $\xi^\sharp$. This row belongs to $\xi^\sharp_{may}$ only if all rows of that group belong to $G^\sharp_{may}$, otherwise it belongs to $\xi^\sharp_{yes}$

database corresponding to dB *w.r.t.* this abstraction. We can denote the result of the execution of $Q_l^\sharp$ and $Q_r^\sharp$ on $dB^\sharp$ as follows:

$$\xi_l^\sharp = S^\sharp[\![Q_l^\sharp]\!](\rho_{dB^\sharp}, \rho_{a^\sharp}) = \langle \xi_{yes_l}^\sharp, \xi_{may_l}^\sharp \rangle$$

$$\xi_r^\sharp = S^\sharp[\![Q_r^\sharp]\!](\rho_{dB^\sharp}, \rho_{a^\sharp}) = \langle \xi_{yes_r}^\sharp, \xi_{may_r}^\sharp \rangle$$

The abstract version of Q is defined as $Q^\sharp = Q_l^\sharp \ \texttt{UNION}^\sharp \ Q_r^\sharp$, where the abstract union operation $\texttt{UNION}^\sharp$ is defined as:

$$\begin{aligned}
\xi^\sharp &= S^\sharp[\![Q^\sharp]\!](\rho_{dB^\sharp}, \rho_{a^\sharp}) \\
&= S^\sharp[\![Q_l^\sharp \ \texttt{UNION}^\sharp \ Q_r^\sharp]\!](\rho_{dB^\sharp}, \rho_{a^\sharp}) \\
&= S^\sharp[\![Q_l^\sharp]\!](\rho_{dB^\sharp}, \rho_{a^\sharp}) \ \texttt{UNION}^\sharp \ S^\sharp[\![Q_r^\sharp]\!](\rho_{dB^\sharp}, \rho_{a^\sharp}) \\
&= \xi_l^\sharp \ \texttt{UNION}^\sharp \ \xi_r^\sharp \\
&= \langle \xi_{yes_l}^\sharp, \xi_{may_l}^\sharp \rangle \ \texttt{UNION}^\sharp \ \langle \xi_{yes_r}^\sharp, \xi_{may_r}^\sharp \rangle \\
&= \langle (\xi_{yes_l}^\sharp \cup \xi_{yes_r}^\sharp), ((\xi_{may_l}^\sharp \cup \xi_{may_r}^\sharp) \backslash (\xi_{yes_l}^\sharp \cup \xi_{yes_r}^\sharp)) \rangle
\end{aligned}$$

Observe that the first component of $\xi^\sharp$, *i.e.*, $(\xi_{yes_l}^\sharp \cup \xi_{yes_r}^\sharp)$ represents the *yes*-part of the result for which abstract pre-condition evaluates to *true*, whereas the second component $((\xi_{may_l}^\sharp \cup \xi_{may_r}^\sharp) \backslash (\xi_{yes_l}^\sharp \cup \xi_{yes_r}^\sharp))$ represents the *may*-part of the result for which the abstract pre-condition evaluates to $\top$.

Example 4 *Consider the database of Table 3.2 that contains the concrete table t_{emp} and consider the following SELECT statement Q_3:*

$$\begin{aligned}
Q_3 &= Q_l \ \texttt{UNION} \ Q_r \\
&= \texttt{SELECT} \ * \ \texttt{FROM} \ t_{emp} \ \texttt{WHERE} \ Age > 15 \ \texttt{UNION} \ \texttt{SELECT} \ * \ \texttt{FROM} \ t_{emp} \ \texttt{WHERE} \ Age > 42
\end{aligned}$$

where,

$$Q_l = \texttt{SELECT} \ * \ \texttt{FROM} \ t_{emp} \ \texttt{WHERE} \ Age > 15$$

$$Q_r = \texttt{SELECT} \ * \ \texttt{FROM} \ t_{emp} \ \texttt{WHERE} \ Age > 42$$

If we execute Q_3 on t_{emp}, we get the result ξ_3 shown in Table 3.12. By following the same abstraction and concretization mapping of Example 1, we get the abstract version of Q_3 i.e. $Q_3^\sharp$ as follows:

$$\begin{aligned}
Q_3^\sharp &= Q_l^\sharp \ \texttt{UNION}^\sharp \ Q_r^\sharp \\
&= \texttt{SELECT}^\sharp \ * \ \texttt{FROM} \ t_{emp}^\sharp \ \texttt{WHERE} \ Age^\sharp >^\sharp [12, 24] \ \texttt{UNION}^\sharp \ \texttt{SELECT}^\sharp \ * \ \texttt{FROM} \ t_{emp}^\sharp \\
&\quad \texttt{WHERE} \ Age^\sharp >^\sharp [25, 59]
\end{aligned}$$

Table 3.12: ξ_3: Result of Q_3 (concrete)

eID	*Name*	*Age*	*Dno*	*Pno*	*Sal*	*Child − no*
1	Matteo	30	2	1	2000	4
2	Alice	22	1	2	1500	2
3	Joy	50	2	3	2300	3
5	Deba	40	3	4	3000	5
6	Andrea	70	1	2	1900	2
7	Alberto	18	3	4	800	1

The execution of $Q_l^\sharp$ on Table 3.7 yields to the result shown in Table 3.13(a), where the tuples with eID$^\sharp$ equal to 2, 7, 8 belong to ξ_{may_l} and the tuples with eID$^\sharp$ equal to 1, 3, 5, 6 belong to ξ_{yes_l}. Similarly, the execution of the query $Q_r^\sharp$ yield to the result shown in Table 3.13(b), where the tuples with eID$^\sharp$ equal to 1, 3, 5 belong to ξ_{may_r} and one tuple with eID$^\sharp$ equal to 6 belongs to ξ_{yes_r}. Thus, the result of abstract computation of $Q_3^\sharp$ involving `UNION`$^\sharp$ *is depicted in Table 3.13(c). Observe that in the result $\xi_3^\sharp$, the yes-part $\xi_{yes_3}^\sharp = (\xi_{yes_l}^\sharp \cup \xi_{yes_r}^\sharp)$ contains the tuples with eID$^\sharp$ equal to 1, 3, 5, 6 and the may-part $\xi_{may_3}^\sharp = ((\xi_{may_l}^\sharp \cup \xi_{may_r}^\sharp) \backslash (\xi_{yes_l}^\sharp \cup \xi_{yes_r}^\sharp))$ contains the tuples with eID$^\sharp$ equal to 2, 7, 8. Here the abstraction is sound i.e. $\xi_3 \in \gamma(\xi_3^\sharp)$.*

Abstract INTERSECTION Operation: Let, $\xi = S[\![Q]\!](\rho_{dB}, \rho_a)$ be the result of executing a concrete query Q on a database dB, where $Q = Q_l$ `INTERSECT` Q_r. It is clear that $\xi = \xi_l \cap \xi_r$, where $\xi_l = S[\![Q_l]\!](\rho_{dB}, \rho_a)$ and $\xi_r = S[\![Q_r]\!](\rho_{dB}, \rho_a)$, according to the concrete intersection operation `INTERSECT`.

Let $Q_l^\sharp$, $Q_r^\sharp$ and $dB^\sharp$ be abstract queries and abstract database corresponding to Q_l, Q_r and dB respectively *w.r.t.* an abstract domain of interest. Let $\xi_l^\sharp = S^\sharp[\![Q_l^\sharp]\!](\rho_{dB^\sharp}, \rho_{a^\sharp}) = \langle \xi_{yes_l}^\sharp, \xi_{may_l}^\sharp \rangle$ and $\xi_r^\sharp = S^\sharp[\![Q_r^\sharp]\!](\rho_{dB^\sharp}, \rho_{a^\sharp}) = \langle \xi_{yes_r}^\sharp, \xi_{may_r}^\sharp \rangle$.

The abstract version of Q is, thus, defined as $Q^\sharp = Q_l^\sharp$ `INTERSECT`$^\sharp$ $Q_r^\sharp$, where abstract intersection operation `INTERSECT`$^\sharp$ is defined as follows:

$$
\begin{aligned}
\xi^\sharp &= S^\sharp[\![Q^\sharp]\!](\rho_{dB^\sharp}, \rho_{a^\sharp}) \\
&= S^\sharp[\![Q_l^\sharp \texttt{ INTERSECT}^\sharp\ Q_r^\sharp]\!](\rho_{dB^\sharp}, \rho_{a^\sharp}) \\
&= S^\sharp[\![Q_l^\sharp]\!](\rho_{dB^\sharp}, \rho_{a^\sharp}) \texttt{ INTERSECT}^\sharp\ S^\sharp[\![Q_r^\sharp]\!](\rho_{dB^\sharp}, \rho_{a^\sharp}) \\
&= \xi_l^\sharp \texttt{ INTERSECT}^\sharp\ \xi_r^\sharp \\
&= \langle \xi_{yes_l}^\sharp, \xi_{may_l}^\sharp \rangle \texttt{ INTERSECT}^\sharp\ \langle \xi_{yes_r}^\sharp, \xi_{may_r}^\sharp \rangle \\
&= \langle (\xi_{yes_l}^\sharp \cap \xi_{yes_r}^\sharp), ((\xi_{may_l}^\sharp \cap \xi_r^\sharp) \cup (\xi_{may_r}^\sharp \cap \xi_l^\sharp)) \rangle
\end{aligned}
$$

Table 3.13: Abstract computation of $Q_3^\sharp$

(a) $\xi_l^\sharp$: Result of $Q_l^\sharp$

$eID^\sharp$	$Name^\sharp$	$Age^\sharp$	$Dno^\sharp$	$Pno^\sharp$	$Sal^\sharp$	$Child-no^\sharp$
1	Matteo	[25,59]	2	1	[1500,2499]	Medium
2	Alice	[12,24]	1	2	[1500,2499]	Few
3	Joy	[25,59]	2	3	[1500,2499]	Medium
5	Deba	[25,59]	3	4	[2500,10000]	Many
6	Andrea	[60,100]	1	2	[1500,2499]	Few
7	Alberto	[12,24]	3	4	[500,1499]	Few
8	Bob	[12,24]	2	3	[2500,10000]	Medium

(b) $\xi_r^\sharp$: Result of $Q_r^\sharp$

$eID^\sharp$	$Name^\sharp$	$Age^\sharp$	$Dno^\sharp$	$Pno^\sharp$	$Sal^\sharp$	$Child-no^\sharp$
1	Matteo	[25,59]	2	1	[1500,2499]	Medium
3	Joy	[25,59]	2	3	[1500,2499]	Medium
5	Deba	[25,59]	3	4	[2500,10000]	Many
6	Andrea	[60,100]	1	2	[1500,2499]	Few

(c) $\xi_3^\sharp$: Performing $\texttt{UNION}^\sharp$ between $\xi_l^\sharp$ & $\xi_r^\sharp$

$eID^\sharp$	$Name^\sharp$	$Age^\sharp$	$Dno^\sharp$	$Pno^\sharp$	$Sal^\sharp$	$Child-no^\sharp$
1	Matteo	[25,59]	2	1	[1500,2499]	Medium
2	Alice	[12,24]	1	2	[1500,2499]	Few
3	Joy	[25,59]	2	3	[1500,2499]	Medium
5	Deba	[25,59]	3	4	[2500,10000]	Many
6	Andrea	[60,100]	1	2	[1500,2499]	Few
7	Alberto	[12,24]	3	4	[500,1499]	Few
8	Bob	[12,24]	2	3	[2500,10000]	Medium

where the first component ($\xi^\sharp_{yes_l} \cap \xi^\sharp_{yes_r}$) represents the *yes*-part of the result, whereas the second component (($\xi^\sharp_{may_l} \cap \xi^\sharp_r$) $\cup$ ($\xi^\sharp_{may_r} \cap \xi^\sharp_l$)) represents the *may*-part of the result.

Example 5 *Consider the concrete table t_{emp} in Table 3.2 and the following SELECT statement:*

$$\begin{aligned} Q_4 &= Q_l \ \texttt{INTERSECT}\ Q_r \\ &= \texttt{SELECT * FROM}\ t_{emp}\ \texttt{WHERE}\ Age > 15\ \texttt{INTERSECT SELECT * FROM}\ t_{emp}\ \texttt{WHERE}\ Age > 42 \end{aligned}$$

where,

$$Q_l = \texttt{SELECT * FROM}\ t_{emp}\ \texttt{WHERE}\ Age > 15$$

$$Q_r = \texttt{SELECT * FROM}\ t_{emp}\ \texttt{WHERE}\ Age > 42$$

If we execute Q_4 on t_{emp}, we get the result ξ_4 shown in Table 3.14. The corresponding abstract query $Q_4^\sharp$, by following the same abstraction and concretization mapping of Example 1, is as

Table 3.14: ξ_4: Result of Q_4 (concrete)

eID	*Name*	*Age*	*Dno*	*Pno*	*Sal*	*Child − no*
3	Joy	50	2	3	2300	3
6	Andrea	70	1	2	1900	2

follows:

$$\begin{aligned} Q_4^\sharp &= Q_l^\sharp \ \mathit{INTERSECT}^\sharp \ Q_r^\sharp \\ &= \mathit{SELECT}^\sharp \ * \ \mathit{FROM}\ t_{emp}^\sharp \ \mathit{WHERE}\ Age^\sharp >^\sharp [12, 24] \ \mathit{INTERSECT}^\sharp \ \mathit{SELECT}^\sharp \ * \ \mathit{FROM}\ t_{emp}^\sharp \\ &\quad \mathit{WHERE}\ Age^\sharp >^\sharp [25, 59] \end{aligned}$$

The execution of $Q_l^\sharp$ and $Q_r^\sharp$ on Table 3.7 yield to the result shown in Table 3.15(a) and 3.15(b) respectively. The result of abstract computation of $Q_4^\sharp$ involving $\mathit{INTERSECT}^\sharp$ is depicted in Table 3.15(c), where the yes-part $\xi_{yes_4}^\sharp = (\xi_{yes_l}^\sharp \cap \xi_{yes_r}^\sharp)$ contains only one tuple with eID$^\sharp$ equal to 6 and the may-part $\xi_{may_4}^\sharp = ((\xi_{may_l}^\sharp \cap \xi_r^\sharp) \cup (\xi_{may_r}^\sharp \cap \xi_l^\sharp))$ contains the tuples with eID$^\sharp$ equal to 1, 3, 5. Here the abstraction is sound i.e. $\xi_4 \in \gamma(\xi_4^\sharp)$.

Abstract MINUS Operation: If we treat an abstract minus operation $\texttt{MINUS}^\sharp$ in a similar manner as of concrete MINUS, we can not preserve the soundness. This happens due to the overapproximated results of the query on right side of $\texttt{MINUS}^\sharp$ operation that removes more information from the result of the query on the left side of $\texttt{MINUS}^\sharp$. So in order to preserve the soundness, we have to treat $\texttt{MINUS}^\sharp$ differently.

Consider an abstract query of the form $Q^\sharp = Q_l^\sharp \ \mathit{MINUS}^\sharp \ Q_r^\sharp$. Let the result for $Q_l^\sharp$ and $Q_r^\sharp$ be $\xi_l^\sharp = \langle \xi_{yes_l}^\sharp, \xi_{may_l}^\sharp \rangle$ and $\xi_r^\sharp = \langle \xi_{yes_r}^\sharp, \xi_{may_r}^\sharp \rangle$ respectively.

The difference operation $\texttt{MINUS}^\sharp$ over an abstract domain is defined as follows:

$$\begin{aligned} \xi^\sharp &= \xi_l^\sharp \ \texttt{MINUS}^\sharp \ \xi_r^\sharp \\ &= \langle \xi_{yes_l}^\sharp, \xi_{may_l}^\sharp \rangle \ \texttt{MINUS}^\sharp \ \langle \xi_{yes_r}^\sharp, \xi_{may_r}^\sharp \rangle \\ &= \langle (\xi_{yes_l}^\sharp \setminus (\xi_{yes_l}^\sharp \cap \xi_{yes_r}^\sharp)), (\xi_{may_l}^\sharp \setminus (\xi_{may_l}^\sharp \cap \xi_{yes_r}^\sharp)) \rangle \end{aligned}$$

Observe that the first component $(\xi_{yes_l}^\sharp \setminus (\xi_{yes_l}^\sharp \cap \xi_{yes_r}^\sharp))$ represents the *yes*-part for which the abstract pre-condition strictly evaluates to *true*, whereas the second component $(\xi_{may_l}^\sharp \setminus (\xi_{may_l}^\sharp \cap \xi_{yes_r}^\sharp))$ represents the *may*-part for which the abstract pre-condition evaluates to $\top$.

Table 3.15: Abstract computation of $Q_4^\sharp$

(a) $\xi_l^\sharp$: Result of $Q_l^\sharp$

$eID^\sharp$	$Name^\sharp$	$Age^\sharp$	$Dno^\sharp$	$Pno^\sharp$	$Sal^\sharp$	$Child-no^\sharp$
1	Matteo	[25,59]	2	1	[1500,2499]	Medium
2	Alice	[12,24]	1	2	[1500,2499]	Few
3	Joy	[25,59]	2	3	[1500,2499]	Medium
5	Deba	[25,59]	3	4	[2500,10000]	Many
6	Andrea	[60,100]	1	2	[1500,2499]	Few
7	Alberto	[12,24]	3	4	[500,1499]	Few
8	Bob	[12,24]	2	3	[2500,10000]	Medium

(b) $\xi_r^\sharp$: Result of $Q_r^\sharp$

$eID^\sharp$	$Name^\sharp$	$Age^\sharp$	$Dno^\sharp$	$Pno^\sharp$	$Sal^\sharp$	$Child-no^\sharp$
1	Matteo	[25,59]	2	1	[1500,2499]	Medium
3	Joy	[25,59]	2	3	[1500,2499]	Medium
5	Deba	[25,59]	3	4	[2500,10000]	Many
6	Andrea	[60,100]	1	2	[1500,2499]	Few

(c) $\xi_4^\sharp$: Performing $\texttt{INTERSECT}^\sharp$ between $\xi_l^\sharp$ & $\xi_r^\sharp$

$eID^\sharp$	$Name^\sharp$	$Age^\sharp$	$Dno^\sharp$	$Pno^\sharp$	$Sal^\sharp$	$Child-no^\sharp$
1	Matteo	[25,59]	2	1	[1500,2499]	Medium
3	Joy	[25,59]	2	3	[1500,2499]	Medium
5	Deba	[25,59]	3	4	[2500,10000]	Many
6	Andrea	[60,100]	1	2	[1500,2499]	Few

Example 6 *Consider the database of Table 3.2 that contains concrete table t_{emp} and consider the following SELECT statement:*

$$\begin{aligned} Q_5 &= Q_l \ \texttt{MINUS}\ Q_r \\ &= \texttt{SELECT}\ *\ \texttt{FROM}\ t_{emp}\ \texttt{WHERE}\ Age > 15\ \texttt{MINUS}\ \texttt{SELECT}\ *\ \texttt{FROM}\ t_{emp}\ \texttt{WHERE}\ Age > 42 \end{aligned}$$

where,

$$Q_l = \texttt{SELECT}\ *\ \texttt{FROM}\ t_{emp}\ \texttt{WHERE}\ Age > 15$$

$$Q_r = \texttt{SELECT}\ *\ \texttt{FROM}\ t_{emp}\ \texttt{WHERE}\ Age > 42$$

If we execute Q_5 on t_{emp}, we get the result ξ_5 shown in Table 3.16. By following the same abstraction and concretization mapping as of Example 1, we get the abstract version of Q_5 as follows:

$$\begin{aligned} Q_5^\sharp &= Q_l^\sharp\ \texttt{MINUS}^\sharp\ Q_r^\sharp \\ &= \texttt{SELECT}^\sharp\ *\ \texttt{FROM}\ t_{emp}^\sharp\ \texttt{WHERE}\ Age^\sharp >^\sharp [12,24]\ \texttt{MINUS}^\sharp\ \texttt{SELECT}^\sharp\ *\ \texttt{FROM}\ t_{emp}^\sharp \\ &\quad \texttt{WHERE}\ Age^\sharp >^\sharp [25,59] \end{aligned}$$

Table 3.16: ξ_5: Result of Q_5 (concrete)

eID	*Name*	*Age*	*Dno*	*Pno*	*Sal*	*Child − no*
1	Matteo	30	2	1	2000	4
2	Alice	22	1	2	1500	2
5	Deba	40	3	4	3000	5
7	Alberto	18	3	4	800	1

The execution of $Q_l^\sharp$ and $Q_r^\sharp$ on Table 3.7 yields to the results shown in Table 3.17(a) and Table 3.17(b) respectively. In Table 3.17(a), the tuples with eID$^\sharp$ equal to 2, 7, 8 belongs to $\xi^\sharp_{may_l}$, whereas the remaining four tuples belong to $\xi^\sharp_{yes_l}$. Similarly, in Table 3.17(b), the tuple with eID$^\sharp$ equal to 6 belongs to $\xi^\sharp_{yes_r}$, whereas the remaining three tuples belong to $\xi^\sharp_{may_r}$. Thus, $(\xi^\sharp_{yes_l} \backslash (\xi^\sharp_{yes_l} \cap \xi^\sharp_{yes_r}))$ contains the tuples with eID$^\sharp$ equal to 1, 3, 5, whereas $(\xi^\sharp_{may_l} \backslash (\xi^\sharp_{may_l} \cap \xi^\sharp_{yes_r})$ contains the tuples with eID$^\sharp$ equal to 2, 7, 8. The result of $Q_5^\sharp$ involving `MINUS`$^\sharp$ *is depicted in Table 3.17(c). Observe that the abstraction is sound i.e. $\xi_5 \in \gamma(\xi_5^\sharp)$.*

Table 3.17: Abstract computation of $Q_5^\sharp$

(a) $\xi_l^\sharp$: Result of $Q_l^\sharp$

eID$^\sharp$	*Name*$^\sharp$	*Age*$^\sharp$	*Dno*$^\sharp$	*Pno*$^\sharp$	*Sal*$^\sharp$	*Child − no*$^\sharp$
1	Matteo	[25,59]	2	1	[1500,2499]	Medium
2	Alice	[12,24]	1	2	[1500,2499]	Few
3	Joy	[25,59]	2	3	[1500,2499]	Medium
5	Deba	[25,59]	3	4	[2500,10000]	Many
6	Andrea	[60,100]	1	2	[1500,2499]	Few
7	Alberto	[12,24]	3	4	[500,1499]	Few
8	Bob	[12,24]	2	3	[2500,10000]	Medium

(b) $\xi_r^\sharp$: Result of $Q_r^\sharp$

eID$^\sharp$	*Name*$^\sharp$	*Age*$^\sharp$	*Dno*$^\sharp$	*Pno*$^\sharp$	*Sal*$^\sharp$	*Child − no*$^\sharp$
1	Matteo	[25,59]	2	1	[1500,2499]	Medium
3	Joy	[25,59]	2	3	[1500,2499]	Medium
5	Deba	[25,59]	3	4	[2500,10000]	Many
6	Andrea	[60,100]	1	2	[1500,2499]	Few

(c) $\xi_5^\sharp$: Performing `MINUS`$^\sharp$ between $\xi_l^\sharp$ & $\xi_r^\sharp$

eID$^\sharp$	*Name*$^\sharp$	*Age*$^\sharp$	*Dno*$^\sharp$	*Pno*$^\sharp$	*Sal*$^\sharp$	*Child − no*$^\sharp$
1	Matteo	[25,59]	2	1	[1500,2499]	Medium
2	Alice	[12,24]	1	2	[1500,2499]	Few
3	Joy	[25,59]	2	3	[1500,2499]	Medium
5	Deba	[25,59]	3	4	[2500,10000]	Many
7	Alberto	[12,24]	3	4	[500,1499]	Few
8	Bob	[12,24]	2	3	[2500,10000]	Medium

Abstract Control Statements

Given an abstraction, the correspondence between the instructions I and its abstract versions $I^\sharp$ for the *conditional* and *while* statements are:

- "$if\ b\ then\ I_1\ else\ I_2$" is abstracted by

$$if\ (b^\sharp = true)\ then\ I_1^\sharp\ elseif\ (b^\sharp = false)\ then\ I_2^\sharp\ else\ I_1^\sharp \sqcup I_2^\sharp$$

- "*while b do I*" is abstracted by $FIX\ F^\sharp$ where $F^\sharp$ is the functional corresponding to the concrete *while* statement.

3.10 Formal Semantics of SQL with Co-related and Non Co-related Subquery

A subquery is a query that is nested inside a SELECT, UPDATE, INSERT, or DELETE statement, or inside another subquery. Subquery can be nested inside a WHERE or HAVING clause of an outer statement, or inside another subquery. A subquery can appear anywhere where an expression can be used, if it returns a single value. However, in practice, there is a limit on the levels of nesting based on the available memory and the complexity of the other expressions in the query.

In the SQL statements that include a co-related subquery (also known as a repeating subquery), the subquery depends on the outer statement for its values. That means that the subquery is executed repeatedly, once for each row that might be selected by outer statement.

The following example illustrates the co-related subquery which finds the name and location of those department under which the average salary of all employees is greater than or equal to 1000:

$\texttt{SELECT}\ Dname,\ Loc\ \texttt{FROM}\ t_{dept}\ \texttt{WHERE}\ 1000 \leq (\texttt{SELECT}\ \texttt{AVG}(Sal)\ \texttt{FROM}\ t_{emp}\ \texttt{WHERE}\ t_{emp}.Dno = t_{dept}.Deptno)$

Here the subquery is *co-related* because the value of the subquery depends on the value of the attribute ($t_{dept}.Deptno$) which is the part of a table in the outer statement.

But the following subquery is non co-related:

$\texttt{SELECT}\ Dname,\ Loc\ \texttt{FROM}\ t_{dept}\ \texttt{WHERE}\ Deptno = \texttt{SOME}(\texttt{SELECT}\ Dno\ \texttt{FROM}\ t_{emp}\ \texttt{WHERE}\ Sal \geq 1500)$

Let Q_{sql} be a SQL statement having Q'_{sql} as a subquery. Suppose, $T^{out} = \{t_1, t_2, ..., t_n\}$ and $T^{in} = \{t'_1, t'_2, ..., t'_m\}$ are the set of tables explicitly appears in Q_{sql} and Q'_{sql} respectively, where $t^{out} = t_1 \times t_2 \times ... \times t_n$ and $t^{in} = t'_1 \times t'_2 \times \cdots \times t'_m$.

Definition 7 *(Co-related Subquery) Q'_{sql} is co-related if $\exists x \in attr(t^{out})$ such that x is used in Q'_{sql}.*

The syntax of the $Q_{sql} = \langle A_{sql}, \phi \rangle$ with one level nested subquery is:

1. $\langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2(Q''_{select}),\ \vec{g}(\vec{e})),\ \phi_1(Q'_{select})) \rangle$
2. $\langle update(\vec{v_d},\ \vec{e}),\ \phi(Q_{select}) \rangle$
3. $\langle insert(\vec{v_d},\ \vec{e}),\ \phi(Q_{select}) \rangle$
4. $\langle delete,\ \phi(Q_{select}) \rangle$

where Q_{select}, Q'_{select} and Q''_{select} don't have any nested subquery.

We use the following idea to describe the semantics of SQL statement with co-related nested subquery:

Suppose t^{out} is partitioned into a set of mutual exclusive tables t_i^{out}, i ranges over the number of rows of t^{out}. Each table t_i^{out} contains a distinct row of t^{out}. So, if there are k rows in t^{out}, after partitioning we get k tables $t_1^{out}, t_2^{out}, \ldots, t_k^{out}$.

Now the following steps are executed k times for $i = 1, \ldots, k$:

1. $t_i = t_i^{out} \times t^{in}$.
2. Execute the subquery Q'_{sql} on the environment (ρ_{t_i}, ρ_a) with $target(Q'_{sql}) = t_i$
3. Substitute the result obtained in step (2) at the place of the subquery Q'_{sql} and execute the outer SQL statement Q_{sql} on the environment $(\rho_{t_i^{out}}, \rho_a)$ with $target(Q_{sql}) = t_i^{out}$
4. Get the final result by taking union of all the results for all i obtained in step 3.

3.10.1 SELECT statement with co-related subquery

In this section, we describe the semantics of SELECT statements nested with co-related subqueries with an example.

Table 3.18: A database $dB2$

(a) t_{emp}

eID	*Name*	*Age*	*Dno*	*Pno*	*Sal*
1	Matteo	28	2	1	2000
2	Stefano	30	1	2	1500
3	luca	25	1	2	1700
4	Alberto	35	3	4	800

(b) t_{dept}

Deptno	*Dname*	*Loc*	*MngrID*
1	Math	Turin	4
2	Computer	Venice	1
3	Physics	Mestre	5

Formal semantics of SELECT statement with co-related subqueries:

$S[\![\langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2(Q''_{select}),\ \vec{g}(\vec{e})),\ \phi_1(Q'_{select})\rangle]\!]_\varsigma(\rho_d, \rho_a)$
$= S[\![\langle select(v_a,\ r(\vec{h}(\vec{x})),\ f(\vec{e'}),\ \phi_2(Q''_{select}),\ \vec{g}(\vec{x},\vec{e})),\ \phi_1(Q'_{select})\rangle]\!]_\varsigma(\rho_{t^{out}} \cup \rho_{t_1^{in}} \cup \rho_{t_2^{in}}, \rho_a)$
where $target(Q'_{select}) = \{t_1^{in}\}$ and $target(Q''_{select}) = \{t_2^{in}\}$.
$= \bigcup_i S[\![\langle select(v_a,\ r(\vec{h}(\vec{x})),\ f(\vec{e'}),\ \phi_2^i,\ \vec{g}(\vec{x},\vec{e})),\ \phi_1^i\rangle]\!]_\varsigma(\rho_{t_i^{out}}, \rho_a)$, where

Let $t_i^1 = t_i^{out} \times t_1^{in}$ and $t_i^2 = t_i^{out} \times t_2^{in}$ and $S[\![Q'_{select}]\!](\rho_{t_i^1}, \rho_a) = \xi'_i$ and $S[\![Q''_{select}]\!](\rho_{t_i^2}, \rho_a) = \xi''_i$
Let $\phi_1^i = \phi_1[\xi'_i/Q'_{select}]$ and $\phi_2^i = \phi_2[\xi''_i/Q''_{select}]$

Illustration of the semantics of SQL statement with co-related subquery using an example:

Consider the database instance $dB2$ depicted in Table 3.18 and the following SELECT statement with a co-related subquery:

SELECT *Dname*, *Loc* FROM t_{dept} WHERE $1000 \leq$ (SELECT AVG(*Sal*) FROM t_{emp} WHERE *Dno* = *Deptno*)

From the above query we get the following information:

- $t^{out} = t_{dept}$
- $t^{in} = t_{emp}$
- Q = SELECT *Dname, Loc* FROM t_{dept} WHERE $1000 \leq (Q')$
  where Q'= SELECT AVG(*Sal*) FROM t_{emp} WHERE *Dno = Deptno*.

Now we illustrate the operations step by step:

Step 1: Partition the table t^{out} into a set of table t_i^{out} each containing one distinct row:

In the example, $t^{out} = t_{dept}$ with three rows. So, after partitioning we have three distinct table t_1^{out}, t_2^{out} and t_3^{out} depicted in Tables 3.19(a), 3.19(b) and 3.19(c) respectively.

Table 3.19: Partitions of Table t_{dept}

(a) t_1^{out}

Deptno	*Dname*	*Loc*	*MngrID*
1	Math	Turin	4

(b) t_2^{out}

Deptno	*Dname*	*Loc*	*MngrID*
2	Computer	Venice	1

(c) t_3^{out}

Deptno	*Dname*	*Loc*	*MngrID*
3	Physics	Mestre	5

Step 2: Execute the following steps for i=1, 2, 3:

Step (2a): Perform $t_i = t_i^{out} \times t^{in}$ for i=1, 2, 3:

In the example, $t^{in} = t_{emp}$. Thus, performing the above operation for three partition t_1^{out}, t_2^{out} and t_3^{out}, we get t_1, t_2 and t_3 depicted in Tables 3.20(a), 3.20(b) and 3.20(c) respectively.

Step (2b): Execute the inner query Q' on the environment (ρ_{t_i}, ρ_a) with $target(Q') = \{t_i\}$ and get the results ξ_i for i=1, 2, 3:

In the example, the execution of the inner query Q' on (ρ_{t_i}, ρ_a) yields to the results ξ_i for i=1, 2, 3, depicted in Tables 3.21(a), 3.21(b) and 3.21(c) respectively.

Step (2c): Substitute the results ξ_i in place of the subquery Q' and get the corresponding outer SQL statements Q_i with $target(Q_i) = \{t_i^{out}\}$ for i=1, 2, 3:

In the example, we have ξ_1, ξ_2 and ξ_3 equals to 1600, 2000 and 800 respectively. After substituting them in place of the subquery Q', we get the following three:

(i) Q_1 = `SELECT` *Dname*, *Loc* `FROM` t_1^{out} `WHERE` 1000 ≤ (1600)

(ii) Q_2 = `SELECT` *Dname*, *Loc* `FROM` t_2^{out} `WHERE` 1000 ≤ (2000)

(iii) Q_3 = `SELECT` *Dname*, *Loc* `FROM` t_3^{out} `WHERE` 1000 ≤ (800)

Table 3.20: Table $t_i = t_i^{out} \times t_{emp}$ for $i = 1, 2, 3$

(a) $t_1 = t_1^{out} \times t_{emp}$

Deptno	*Dname*	*Loc*	*MngrID*	*eID*	*Name*	*Age*	*Dno*	*Pno*	*Sal*
1	Math	Turin	4	1	Matteo	28	2	1	2000
1	Math	Turin	4	2	Stefano	30	1	2	1500
1	Math	Turin	4	3	luca	25	1	2	1700
1	Math	Turin	4	4	Alberto	35	3	4	800

(b) $t_2 = t_2^{out} \times t_{emp}$

Deptno	*Dname*	*Loc*	*MngrID*	*eID*	*Name*	*Age*	*Dno*	*Pno*	*Sal*
2	Computer	Venice	1	1	Matteo	28	2	1	2000
2	Computer	Venice	1	2	Stefano	30	1	2	1500
2	Computer	Venice	1	3	luca	25	1	2	1700
2	Computer	Venice	1	4	Alberto	35	3	4	800

(c) $t_3 = t_3^{out} \times t_{emp}$

Deptno	*Dname*	*Loc*	*MngrID*	*eID*	*Name*	*Age*	*Dno*	*Pno*	*Sal*
3	Physics	Mestre	5	1	Matteo	28	2	1	2000
3	Physics	Mestre	5	2	Stefano	30	1	2	1500
3	Physics	Mestre	5	3	luca	25	1	2	1700
3	Physics	Mestre	5	4	Alberto	35	3	4	800

Table 3.21: Result $\xi_i = S[\![Q']\!](\rho_{t_i}, \rho_a)$ for i=1, 2, 3

(a) ξ_1

AVG(Sal)
1600

(b) ξ_2

AVG(Sal)
2000

(c) ξ_3

AVG(Sal)
800

Step (2d): Execute Q_i over the environment $(\rho_{t_i^{out}}, \rho_a)$ for i=1, 2, 3:

The execution of Q_1, Q_2 and Q_3 over $(\rho_{t_1^{out}}, \rho_a)$, $(\rho_{t_2^{out}}, \rho_a)$ and $(\rho_{t_3^{out}}, \rho_a)$ gives the results shown in Tables 3.22(a), 3.22(b) and 3.22(c) respectively. Observe that the execution in the third case results into an empty table.

Step (2e): Get the final result of the SQL statement $Q = \bigcup_i S[\![Q_i]\!](\rho_{t_i^{out}}, \rho_a)$ for i=1, 2, 3:

In the example, $S[\![Q]\!](\rho_{t^{out}} \cup \rho_{t^{in}}, \rho_a) = S[\![Q]\!](\rho_{t_{dept}} \cup \rho_{t_{emp}}, \rho_a) = S[\![Q_1]\!](\rho_{t_1^{out}}, \rho_a)$ $\cup\ S[\![Q_2]\!](\rho_{t_2^{out}}, \rho_a) \cup S[\![Q_3]\!](\rho_{t_3^{out}}, \rho_a)$. The result is shown in Table 3.23.

Table 3.22: $S[\![Q_i]\!](\rho_{t_i^{out}}, \rho_a)$: Evaluation of Q_i for $i = 1, 2, 3$

(a)

Dname	*Loc*
Math	Turin

(b)

Dname	*Loc*
Computer	Venice

(c)

Dname	*Loc*
empty	

Table 3.23: $S[\![Q]\!](\rho_{t^{out}} \cup \rho_{t_{in}}, \rho_a)$: Evaluation of Q

Dname	*Loc*
Math	Turin
Computer	Venice

3.10.2 SELECT statement with non co-related subquery

In this section, we describe the semantics of SELECT statements nested with non co-related subqueries with an example.

Formal semantics of SELECT statement with non co-related subqueries:

$S[\![\langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2(Q_{select_2}),\ \vec{g}(\vec{e})),\ \phi_1(Q_{select_1})\rangle]\!]_\varsigma(\rho_d, \rho_a)$
$= S[\![\langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2(Q_{select_2}),\ \vec{g}(\vec{e})),\ \phi_1(Q_{select_1})\rangle]\!]_\varsigma(\rho_{t^{out}} \cup \rho_{t_1^{in}} \cup \rho_{t_2^{in}}, \rho_a)$
where $target(Q_{select_1}) = \{t_1^{in}\}$ and $target(Q_{select_2}) = \{t_2^{in}\}$.
$= S[\![\langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2',\ \vec{g}(\vec{e})),\ \phi_1'\rangle]\!]_\varsigma(\rho_{t^{out}}, \rho_a)$, where

$S[\![Q_{select_1}]\!](\rho_{t_1^{in}}, \rho_a) = \xi_1$ and $S[\![Q_{select_2}]\!](\rho_{t_2^{in}}, \rho_a) = \xi_2$ and $\phi_1' = \phi_1[\xi_1/Q_{select_1}]$ and $\phi_2' = \phi_2[\xi_2/Q_{select_2}]$

Illustration of the semantics of SELECT statement with non co-related subquery using an example:

Consider the following SELECT statement with non co-related subquery and the database instance $dB2$ depicted in Table 3.18:

SELECT *Dname*, *Loc* FROM t_{dept} WHERE *Deptno* = SOME(SELECT *Dno* FROM t_{emp} WHERE *Sal* $\geq$ 1500)

From the above query, we get the following information:

- $t^{out} = t_{dept}$
- $t^{in} = t_{emp}$

- Q = SELECT *Dname*, *Loc* FROM t_{dept} WHERE *Deptno*=SOME (Q')
  where, Q' = SELECT *Dno* FROM t_{emp} WHERE $Sal \geq 1500$

Now we illustrate the operations step by step:

Step 1: Execute the inner query Q' on the environment $(\rho_{t^{in}}, \rho_a)$ and get the result ξ':

In the example, $t^{in} = t_{emp}$ and the semantic execution of Q' on $(\rho_{t_{emp}}, \rho_a)$ yields to the result ξ', depicted in Table 3.24.

Table 3.24: Table $\xi' = S[\![Q']\!](\rho_{t_{emp}}, \rho_a)$

Dno
2
1
1

Step 2: Substitute the result ξ' in place of the subquery Q', and get the corresponding outer SQL statement Q:

As ξ' is $\langle 2, 1, 1\rangle$, by substituting it in place of Q', we get:

$$Q = \texttt{SELECT}\ \textit{Dname}, \textit{Loc}\ \texttt{FROM}\ t_{dept}\ \texttt{WHERE}\ \textit{Deptno} = \texttt{SOME}(2,\ 1,\ 1)$$

Step 3: Execute Q over the environment $(\rho_{t^{out}}, \rho_a)$:

The execution of Q over $(\rho_{t^{out}}, \rho_a)$, *i.e.*, $S[\![Q]\!](\rho_{t_{dept}}, \rho_a)$ yields to the result shown in Table 3.25.

Table 3.25: $S[\![Q]\!](\rho_{t_{dept}}, \rho_a)$: Evaluation of Q

Dname	*Loc*
Math	Turin
Computer	Venice

3.10.3 Formal semantics of UPDATE/INSERT/DELETE statement with co-related subquery

$S[\![\langle A_{sql},\ \phi(Q_{select})\rangle]\!]_\varsigma(\rho_d, \rho_a)$

$= S[\![\langle A_{sql},\ \phi(Q_{select})\rangle]\!]_\varsigma(\rho_{t^{out}} \cup \rho_{t^{in}}, \rho_a)$ where $target(Q_{select}) = \{t^{in}\}$.

$= \bigcup_i S[\![\langle A_{sql},\ \phi_i\rangle]\!]_\varsigma(\rho_{t_i^{out}}, \rho_a)$ where,

$$Let\ t_i = t_i^{out} \times t^{in} \text{ and } S[\![Q_{select}]\!](\rho_{t_i}, \rho_a) = \xi_i \text{ and } \phi_i = \phi[\xi_i/Q_{select}]$$

3.10.4 Formal semantics of UPDATE/INSERT/DELETE statement with non co-related subquery

$S[\![\langle A_{sql},\ \phi(Q_{select})\rangle]\!](\rho_d, \rho_a)$

$= S[\![\langle A_{sql},\ \phi(Q_{select})\rangle]\!](\rho_{t^{out}} \cup \rho_{t^{in}}, \rho_a)$

$= S[\![\langle A_{sql},\ \phi'\rangle]\!](\rho_{t^{out}}, \rho_a)$, where $S[\![Q_{select}]\!](\rho_{t^{in}}, \rho_a) = \xi$ and $\phi' = \phi[\xi/Q_{select}]$

Chapter 4

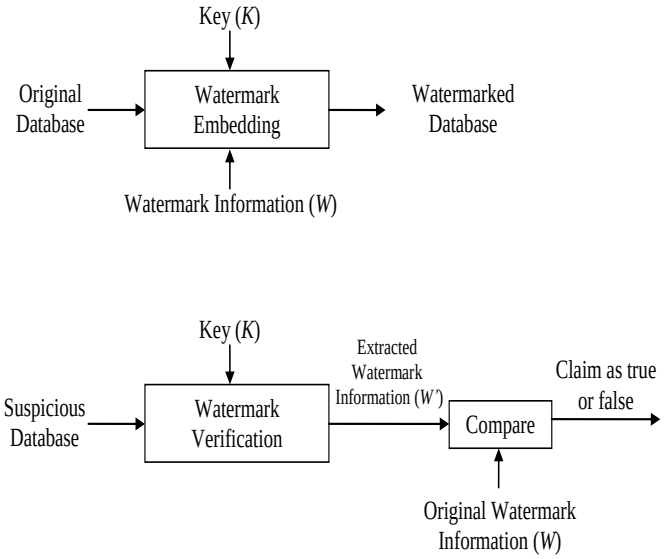
Persistent Watermarking of Relational Databases

[Part of this chapter is already published in (82, 83, 91)]

The recent surge in the growth of the Internet technology and Information Systems results in offering of a wide range of web-based services, such as database as a service, digital repositories and libraries, e-commerce, online decision support system etc. These applications make the digital assets, such as digital images, video, audio, database content etc, easily accessible by the ordinary people around the world for sharing, purchasing, distributing, or many other purposes. As the information usage proliferates among more and more users, the database content faces serious challenges like illegal redistribution, ownership claims, forgery, theft, etc. Although the encryption is one way to prevent attacks to the database content, this approach is too restrictive and does not constitute a general solution to the challenges mentioned above. A more effective approach is using digital watermarking technologies where a watermark is some kind of information that is embedded into the underlying data for tamper detection, localization, ownership proof, traitor tracing etc.

Initially, most of the work on watermarking was concentrated on watermarking of still images, video, audio, VLSI design etc (1, 129, 161). However, in the recent years watermarking of database systems started to receive attention because of the increasing use of it in many real-life applications. Examples where database watermarking might be of a crucial importance include protecting rights and ensuring the integrity

Figure 4.1: Basic Database Watermarking Technique



of outsourced relational databases in service provider model (77), in data mining technology where data are sold in pieces to parties specialized in mining it (7), online B2B interactions (98) etc. The idea to secure a database of map information (represented as a graph) by digital watermarking technique was first coined by Khanna and Zane in 2000 (116). In 2002, Agrawal et al. proposed the idea of digital watermarking for relational database (6).

In general, the database watermarking techniques consist of two phases: *Watermark Embedding* and *Watermark Verification*. During watermark embedding phase, a private key K (known only to the owner) is used to embed the watermark W into the original database. The watermarked database is then made publicly available. To verify the ownership of a suspicious database, the verification process is performed where the suspicious database is taken as input and by using the private key K (the same which is used during the embedding phase) the embedded watermark (if present) is extracted and compared with the original watermark information. Figure 4.1 depicts the basic database watermarking technique.

The relational data defers from multimedia data in many respects: (*i*) Few Redundant Data: Multimedia objects consists of large number of bits providing large cover to hide watermark, whereas the database object is a collection of independent objects,

called tuples. The watermark has to be embedded into these tuples, (*ii*) Out-of-Order Relational Data: The relative spatial/temporal positions of different parts or components in multimedia objects do not change, whereas there is no ordering among the tuples in database relations as the collection of tuples is considered as set, (*iii*) Frequent Updating: Any portion of multimedia objects is not dropped or replaced normally, whereas tuples may be inserted, deleted, or updated during normal database operations, (*iv*) There are many psycho-physical phenomena based on human visual system and human auditory system which can be exploited for mark embedding. However, one can not exploit such phenomena in case of relational databases. Due to these differences between relational and multimedia data, there exist no image or audio watermarking method which is suitable for watermarking of relational databases. These differences give rise to many technical challenges in database watermarking as well.

Most of the existing watermarking techniques (5, 20, 73, 133, 202) in the literature are private, meaning that they are based on some private parameters (*e.g.* secret key). Only the authorized people (*e.g.* database owners) who know these private parameters are able to verify the watermark to prove their ownership of the database in case of illegal redistribution, false ownership claim, theft etc. However, private watermarking techniques suffer from disclosure of the private parameters to dishonest people once the watermark is verified in presence of the public. With access to the private parameters, attackers can easily invalidate watermark detection by either removing watermarks from protected data or adding a false watermark to non-watermarked data. In contrast, in public watermarking techniques (132, 187), any end-user can verify the embedded watermark as many times as necessary without having any prior knowledge about any of the private information to ensure that they are using the correct (not tampered) data coming from the original source. For instance, when customers are using sensitive information such as currency exchange rates and stock prices, it is very important for them to ensure that data is correct and coming from the original sources. Observe that since the location of the public watermark in the host data is public, the robustness (57) of it is a prime concern. In addition, fragileness (57) of the public watermark must be maintained when any end-user wishes to verify the correctness of the data through it.

The watermark verification phase in the existing techniques (73, 133, 187, 202) completely relies on the content of the database. In other words, the success of the

watermark detection is content dependent. *Benign Updates* or any other authorized processing of the database content may damage or distort the embedded watermark, resulting the watermark verification unsuccessful. For instance, suppose a publisher is offering 20% discount to the price of all articles. The modification of the price information may yield to the watermark detection phase almost infeasible, if the price values are marked at bit-level or if any information (*viz*, hash value) is extracted based on this price information and is used in the embedding phase. Therefore, most of the previous techniques are designed to face *Value Modification Attacks*, but are unable to resolve the persistency of the watermark under intentional operations.

All the issues above motivate us to propose a novel fragile and robust persistent watermarking scheme that embeds both private and public watermarks where the former allows owners to prove their ownership, while the latter allows end-users to verify the correctness and originality of the data in the database without loosing its strength and security. In this setting, the public watermark is based on a part of the database state which remains invariant under processing by the SQL statements associated with the database, whereas private watermarking is based on an appropriate form of the original database state, called *abstract database*, and the *semantics-based properties* of the data which remain invariant under processing by the associated SQL statements.

The structure of this chapter is as follows: in section 4.1, we survey the current state-of-the-art of relational database watermarking schemes in the literature. In the survey, we discuss different types of possible attacks on watermarked databases, and we classify and compare all the existing techniques along various dimensions. In section 4.2, we discuss our proposal on fragile and robust persistent watermarking, by illustrating with suitable example.

4.1 Literature Survey

In this section, we survey the current state-of-the-art of relational database watermarking schemes in the literature, and we classify them according to their intent, the way they express the watermark, the cover type, the granularity level, and their verifiability.

4.1.1 Applications of Digital Watermark for Relational Databases

Digital Watermarks for relational databases are potentially useful in many applications, including:

1. Ownership Assertion: Watermarks can be used for ownership assertion. To assert ownership of a relational database, Alice can embed a watermark into her database R using some private parameters (*e.g.* secret key) which is known only to her. Then she can make the watermarked database publicly available. Later, suppose Alice suspects that the relation S published by Mallory[1] has been pirated from her relation R. The set of tuples and attributes in S can be a subset of R. To defeat Mallory's ownership claiming, Alice can demonstrate the presence of her watermark in Mallory's relation. For such a scheme to work, the watermark has to survive intentional or unintentional data processing operations which may remove or distort the watermark.

2. Fingerprinting: Fingerprinting aims to identify a traitor. In the applications where database content is publicly available over a network, the content owner would like to discourage unauthorized duplication and distribution by embedding a distinct watermark (or fingerprint) in each copy of the database content. If, at a later point in time, unauthorized copies of the database are found, then the origin of the copy can be determined by retrieving the fingerprint.

3. Fraud and Tamper Detection: When database content is used for very critical applications such as commercial transactions or medical applications, it is important to ensure that the content was originated from a specific source and that it had not been changed, manipulated or falsified. This can be achieved by embedding a watermark in the underlying data of the database. Subsequently, when the database is checked, the watermark is extracted using a unique key associated with the source, and the integrity of the data is verified through the integrity of the extracted watermark.

[1]Conventionally, in cryptography literature, Mallory represents the malicious active attacker.

4.1.2 Different Types of Attacks

Generally, the digital watermarking for integrity verification is called fragile watermarking as compared to robust watermarking for copyright protection. In a robust watermarking scheme, the embedded watermark should be robust against various attacks which aim at removing or distorting the watermark. While in a fragile watermarking scheme, the embedded watermark should be fragile to modifications so as to detect and localize any modification in presence of different attacks.

The watermarked database may suffer from various types of intentional and unintentional attacks which may damage or erase the watermark, as described below:

1. Benign Update: In this case, the tuples or data of any watermarked relation are processed as usual. As a result, the marked tuples may be added, deleted or updated which may remove the embedded watermark or may cause the embedded watermark undetectable (for instance, during update operation some marked bits of marked data can be erroneously flipped). This type of processing are performed unintensionally.

2. Value Modification Attack:

   - Bit Attack: This attack attempts to destroy the watermark by altering one or more bits in the watermarked data. More information about the marked bit position makes attack more successful. However, in this case usefulness of data is crucial: more alternation may result the data completely useless.

     Bit attack may be performed randomly which is known as *Randomization Attack* by assigning random values to certain bit positions; or by *Zero Out Attack* where the values in the bit positions are set to zero; or may be performed by inverting the values of the bit positions, known as *Bit Flipping Attack*.

   - Rounding Attack: Mallory may try to lose the marks contained in a numeric attribute by rounding all the values of the attribute. Success of this attack depends on the estimation of how many bit positions are involved in the watermarking. Underestimation of it may cause the attack unsuccessful, whereas overestimation may cause the data useless.

- Transformation: An attack related to the rounding attack is one in which the numeric values are linearly transformed. For example, Mallory may convert the data to a different unit of measurement (e.g., Fahrenheit to Celsius). The unnecessary conversion by Mallory would raise suspicion among users.

3. Subset Attack: Mallory may consider a subset of the tuples or attributes of a watermarked relation and by attacking (deleting or updating) on them he may hope that the watermark has been lost.

4. Superset Attack: Some new tuples or attributes are added to a watermarked database which can affect the correct detection of the watermark.

5. Collusion Attack: This attack requires the attacker to have access to multiple fingerprinted copies of the same relation.

   - Mix-and-Match Attack: Mallory may create his relation by taking disjoint tuples from multiple relations containing similar information.
   - Majority Attack: This attack creates a new relation with the same schema as the copies but with each bit value computed as the majority function of the corresponding bit values in all copies so that the owner can not detect the watermark.

6. False Claim of Ownership: This type of attack seeks to provide a traitor or pirate with evidence that raises doubts about merchant's claim.

   - Additive Attack: Mallory may simply add his watermark to Alice's watermarked relation and try to claim his ownership.
   - Invertibility Attack: Mallory may launch an invertibility attack to claim his ownership if he can successfully discover a fictitious watermark which is in fact a random occurrence from a watermarked database.

7. Subset Reverse Order Attack: Attacker enjoys this attack by exchanging the order or positions of the tuples or attributes in relation which may erase or disturb the watermark.

8. Brute Force Attack: In this case, Mallory tries to guess about the private parameters (*e.g.* secret key) by traversing the possible search spaces of the parameters. This attack can be thwarted by assuming that the private parameters are long enough in size.

4.1.3 Watermarking Issues

The important issues that arise in the study of digital watermarking techniques for relational databases are:

- Capacity: It determines the optimum amount of data that can be embedded in a cover and the optimum way to embed and extract this information.
- Usability: The changes in the data of the database during watermarking process should not degrade the usability of the data. The amount of allowable change differs from one database to another, depending on the nature of stored records.
- Robustness: Watermarks embedded in databases should be robust against malicious or accidental attempts at removal without destroying the usability of the database.
- Security: The security of the watermarking process relies on some private parameters (*e.g.* secret key) which should be kept completely secret. Owner of the database should be the only one who has knowledge about them.
- Blindness: Watermark extraction should require neither the knowledge of the original unwatermarked database nor the watermark information. This property is critical as it allows the watermark to be detected in a copy of the database relation, irrespective of later updates to the original relation.
- Incremental Watermarking: After a database has been watermarked, the watermarking algorithm should compute the watermark values only for the added or modified tuples, keeping the unaltered watermarked tuples untouched.
- Non-interference: If multiple marks are inserted into a single relational database, then they should not interfere with each other.

- Public System: Following Kerckhoffs (114), the watermarking system should assume that the method used for inserting a watermark is public. Defense must lie only in the choice of the private parameters (*e.g.* secret key).

- False Positiveness and False Negativeness: The false hit is the probability of a valid watermark being detected from unwatermark data, whereas false miss is the probability of not detecting a valid watermark from watermarked data that has been modified in typical attacks. The false hit and false miss should be negligible.

4.1.4 Classification of Watermarking Techniques

The watermarking techniques proposed so far can be classified along various dimensions as follows:

- Watermark Information: Different watermarking schemes embed different types of watermark information (*e.g.* image, text etc.) into the underlying data of the database.

- Distortion: Watermarking schemes may be distortion-based or distortion-free depending on whether the marking introduces any distortion to the underlying data.

- Cover Type: Watermarking schemes can be classified based on the type of the cover (*e.g.* type of attributes) into which marks are embedded.

- Granularity Level: The watermarking can be performed by modifying or inserting information at bit level or higher level (*e.g.* character level or attribute level or tuple level).

- Verifiability/Detectability: The detection/verification process may be deterministic or probabilistic in nature, it can be performed blindly or non-blindly, it can be performed publicly (by anyone) or privately (by the owner only).

- Intent of Marking: Different watermarking schemes are designed to serve different purposes, namely, integrity and tamper detection, localization, ownership proof, traitor detection etc.

4.1.5 Watermarking Techniques

In this Section, we try to cover the details of various watermarking techniques proposed so far. We categorize the proposed techniques based on (*i*) whether marking introduces any distortion, (*ii*) the type of the underlying data (cover) in which watermark information is embedded, and (*iii*) the type of the watermark information to be embedded.

Based on whether the marking introduces any changes in the underlying data of the database, the watermarking techniques can be categorized into two: *Distortion-based* and *Distortion-free*.

(1) Distortion-based Watermarking

The watermarking techniques in this category introduce small changes in the underlying data of the database during the embedding phase. The degree of changes should be such that any changes made in the data are tolerable and should not make the data useless. The watermarking can be performed at bit level, or character level, or higher such as attribute or tuple level, over the attribute values of types numeric, string, categorical, or any.

(a) Watermarking Based on Numerical Data Type Attribute:

Different types of information are embedded into the numeric data type attributes of the database as watermark information. Below we categorize the existing schemes based on the watermark information they are embedding into the numeric data type attributes of the database.

Arbitrary meaningless bit pattern as watermark information. The watermarking schemes proposed by Agrawal et al. (4, 5, 6) (also known as AHK algorithm) is based on numeric data type attribute and marking is done at bit-level. The basic idea of these schemes is to ensure that some bit positions for some of the attributes of some of the tuples in the relation contain specific values. This bit pattern constitutes the *watermark*. The tuples, attributes within a tuple, bit positions in an attribute, and specific bit values at those positions are algorithmically determined under the control of the private parameters γ, ν, ξ and K known only to the owner of the relation. The

parameters γ, ν, ξ and K represent number of tuples to mark, number of attributes available to mark, number of least significant bits available for marking in an attribute, and secret key respectively. In (6), the cryptographic MAC function $H(K||H(K||r.P))$ where $r.P$ is the primary key of the tuple r and || represents concatenation operation, is used to determine the candidate bit positions. The HASH function $H(K||r.P)$ is used to determine the bit values to be embedded at those positions. The choice of MAC and HASH is due to the one-way functional characteristics and less collision probability. In (4, 5), the authors used pseudorandom sequence generator (*e.g.*, Linear Feedback Shift Register (89, 90)) instead of HASH and MAC to identify the marking bits and mark positions. The security and robustness of this scheme relies on these parameters which are completely private to the owner. The watermark detection algorithm is blind and probabilistic in nature. A relation is considered as pirated if the matching pattern is present in at least τ tuples, where τ depends on the actual number of tuples marked and a preselected value α, called the significance level of the test. Observe that the success of watermark detection phase depends on the fixed order of attributes. Re-sort of attributes' order may yield to the detection phase almost infeasible. Although the main assumption of this scheme is that the relation has primary key whose value does not change, they suggested an alternative to treat a relation without primary key. Li et al. (134) also suggested three different schemes to obtain virtual primary key for a relation without primary key.

Lafaye (126) described the security properties for the AHK algorithm by analyzing the security and robustness in two situations: (*i*) Multiple Keys Single Database (MKSD): When a single database is watermarked several times using different secret keys and sold to different users, and (*ii*) Single Key Multiple Databases (SKMD): When several different databases are watermarked using a single secret key. An attempt of random attack on a watermarked content obtained by the AHK algorithm, may be successful when randomize the ξ^{th} least significant bits of all tuples of the relation. However, this attack is highly invasive since most values of the relation are impacted by the attack. The locations guessed based on MKSD and SKMD can be used to build a better focussed attack.

Qin et al. (164) suggested an improvement over the Agrawal and Kiernan's scheme (6). Instead of using hash function, they use chaotic random series based on the Logistic chaos equation which has two properties: the non-repetitive iterative operation and

the sensitiveness to initial value. It avoids the inherent weakness of collision of Hash function. The selection of bits of LSB for embedding watermark meets the requirements of both data range and data precision of each attribute, rather than simply to use a same ξ for all attributes. So the error caused by watermark is decreased significantly, hardly affects the availability of the database.

Among the most recent works, Gupta et al. (74) proposed a reversible watermarking scheme which is the modified version of Agrawal and Kiernan's one (6). In this scheme, during the detection phase, the original unwatermarked version of the database can be recovered along with the ownership proof. The operation involved in the embedding phase extracts a bit *OldBit* from the integer portion of the attribute value before replacing it by the watermark bit and inserts it in the fraction portion of the attribute value. Thus, the watermark bit can be recovered during detection and the attribute can be restored to its unmarked value by replacing the watermark bit with the original bit *OldBit* extracted from the fraction part. They also proposed an another algorithm to defeat any attempt of additive or secondary attack which relies on the obvious fact that the database relation must be watermarked by the actual owner before Mallory.

The watermarking method in (197) embeds random digits (between 0 to 9) at LSB positions of the candidate attributes for some algorithmically chosen tuples. During the embedding phase, the tuples are securely partitioned into groups using the cryptographic hash function and only the first m (which is equal to the length of the owner's watermark) groups are considered. The decision whether to mark i^{th} ($1 \leq i \leq m$) group depends on the i^{th} bit of the owner's watermark, whereas the selection of the tuples in a group is based on a secret key (which is different from that used during partitioning) as well as the information at second LSB positions of the numeric candidate attributes. Finally, random numbers (between 0 and 9) are embedded at LSB positions in the attribute values of the selected tuples. Observe that although the owner has a watermark of length m, it is not actually embedded. Rather, it is used to identify some valid groups to embed the random values which acts as embedded watermark information. The detection phase determines the presence of mark in a group if the maximum occurrence frequency for a value between 0 and 9 for that group exceeds a threshold.

Image as watermark information. Wang et al. (190) described an image-based watermarking scheme where instead of embedding original image as watermark, an scrambled image based on Arnold transform with scrambling number d is used. Since Arnold transform of an image has the periodicity P, the result which is obtained in the extraction phase can be recovered from the scrambled form to the original after $(P-d)$ iterations. In the embedding phase, the original image of size $N \times N$ is first converted into scrambled image which is then represented by a binary string b_s of length $L = N \times N$. Secondly, all tuples in the relation are grouped into L groups. The hash value which is computed using tuple's primary key, secret key and order of the image, determines the group in which each tuple belongs. Finally, the i^{th} bit of b_s is embedded into the algorithmically chosen bit position of the attribute value for those tuples in i^{th} group that satisfy a particular criterion. The detection phase follows majority voting technique. However, the security of this scheme improves as it relies not only on the secret key but also the scrambling number d and the order of the image N.

Rather than embedding scrambled image, the watermarking technique in (100) embeds the original image by first converting it into a bit flow (EMC, Encrypted Mark Code) of certain length, and then by following similar algorithmic steps as in (190). The only two differences are that (*i*) the watermark insertion technique in (190) assumes single fixed attribute to mark for all tuples whereas (100) does not, and (*ii*) during selection of bit positions, the order of the image is not considered in (100). Finally, after marking, (100) checks the usability of the data with respect to the intended use. If acceptable, the change is committed, otherwise rolled back.

Another watermarking scheme to embed image in BMP format is presented in (203). In watermark insertion phase, the BMP image is divided into two parts: *header* and *image data*. An error correction approach of BCH (Bose-Chaudhuri-Hocquenhem) coding is used to encode the image data part into watermark. Based on the tuple's ID value which is computed by performing hashing function parameterized with tuple's primary key and BMP header, all the tuples are assigned to k distinct subsets, where k is the length of the watermark. Finally, each of the k bits of the watermark is used to mark each of the k subsets of tuples. During the marking, the selected least significant bit positions of selected attributes of some specific tuples satisfying a particular criterion are altered. The selection of bit positions depends on HASH or MAC function parameterized with BMP header, tuple's primary key and other

parameters like number of least significant bits in the attribute value. Observe that the selected bit positions are not set to the watermark bits directly, but rather, are set to mask bits which are computed from both the hash value and the watermark bit together.

The image-based fragile watermarking scheme in (186) aims at maintaining integrity of the database and uses support vector regression (SVR) to train high correlation attributes to generate the SVR predicting function for embedding watermark into particular numeric attributes. This scheme consists of three phases: (*i*) Training Phase: select training tuples and obtain trained SVR predicting function; (*ii*) Embedding Phase: all tuples in the relation are used to embed image watermark where the number of watermark bits is designed to be equal to the number of tuples. Each numeric attribute value C_i of i^{th} tuple t_i is predicted using the SVR prediction function f resulting $\bar{C}_i = f(t_i)$. Based on the i^{th} watermark bit b_i (obtained after converting the image into bit flow), the value of C_i is modified by $\bar{C}_i + 1$ or $\bar{C}_i - 1$; (*iii*) Tamper Detection: the trained SVR predicting function is used to generate the predicted value for each tuple and compared with the value contained in the database. The difference of these two values determines the watermark information and can ensure whether database is tampered or not. However, the limitation of this scheme is that it can identify the modification which takes place in the objective attribute set only. This scheme works good in the case where the tuples in the table are independent but highly correlated between the attributes.

Speech as watermark information. Wang et al. (191) proposed the use the owner's speech to generate unique watermark. The preparation of watermark from the speech consists of several stages: compression of speech signal to shorten the watermark, speech signal enhancement to remove noise in frequency domain, speech signal conversion into bit stream, and finally, watermark generation by using the copyright message of the holder and the result of the converted speech signal. The bit-level marking is performed during watermark embedding phase by following the same algorithmic steps as in image-based technique of (100).

Genetic Algorithm based watermark signal. The authors in (144) proposed a Genetic Algorithm-based technique to generate watermark signal, focusing on the optimization

issue. They follow the same algorithmic framework as of (100).

Content characteristics as watermark information. The watermarking schemes in (73, 202) are performed based on the content of the database itself.

In (73), the authors proposed a fragile watermarking scheme that can verify the integrity of database relation. In the proposed scheme, all tuples in a database relation are first securely divided into groups and sorted. In each group, there are two kinds of watermarks to be embedded: attribute watermark W_1 which consists of γ watermarks of length ν and tuple watermark W_2 which consists of ν watermarks of length γ, where γ and ν are the number of attributes in a tuple and average number of tuples in each group respectively. W_1 and W_2 are created by extracting bit sequence from the hash value. For attribute watermark W_1, the hash value is generated according to the message authentication code and the same attribute of all tuples in the same group, while for tuple watermark W_2, it is formed from the same message authentication code and all attributes of the same tuple. Observe that, in both the embedding and detection phases, they ignore the least two significant bits of all attributes of numeric type except the primary key when computing hash values. The attribute watermark is embedded in LSB level, whereas tuple watermark is embedded at next to the LSB level. In this way, the embedded watermarks actually form a watermark grid, which helps to detect, localize and characterize modifications.

In (202), the watermark insertion phase extracts some bits, called local characteristic, from the characteristic attribute A_1 of tuple t and embeds those bits into the watermark attribute A_2 of the same tuple. The selection of tuples depends on whether the generated random value (between 0 and 1) is less than the embedded proportion α of the relational databases and the non-NULL requirement of characteristic attribute value. In the watermark detection phase, by following similar procedure, the local characteristic of the characteristic attribute are extracted and compared against the last bits of watermark attribute.

Cloud model as watermark information. The cloud watermarking scheme in (201) is based on the Cloud Model with three digital characteristics: Expected value (*Ex*), Entropy (*En*) and Hyper Entropy (*He*). In watermark creation phase, it uses the forward cloud generation algorithm to generate cloud drops from cloud and embed those

cloud drops into the relation as watermark. The detection algorithm uses backward cloud generation algorithm to extract the cloud with parameters *Ex*, *En* and *He* from the embedded cloud drops, and finally, a similar cloud algorithm is used to verify whether both clouds (one used during watermark embedding and other extracted during watermark verification phase) are similar or not. This scheme is not blind as it requires original relational database during verification phase.

Other meaningful watermark information. The watermarking scheme in (102) embeds a meaningful watermark information by first converting it into a bit flow. The scheme computes unique ID for all tuples in the relation and sort them in ascending order according to their ID values. The tuples are then partitioned into p groups each containing m tuples. The i^{th} bit of the bit flow is embedded into the selected tuples in i^{th} group by following same selection criterion as in AHK algorithm with exception that it considers only single attribute to mark. Before committing the change, a constraint function is used to check whether the change exceeds the data usability bounds. The constraint function includes the basic data statistical measurement constraints, semantics constraints and structural constraints. Mean and standard deviation of the data set are very common aspects in basic data statistical measurement constraints. Semantics constraints and structural constraints are defined by user's input as SQL statements according to relational table. If during embedding phase, any tuple is selected but rolled back, it is recorded and avoided during extraction phase to reduce false positive. Watermark extraction phase is blind, probabilistic and follows the majority voting technique.

The partitioning of tuples in most of the techniques is based on hashing. Huang et al. (101), instead, proposed the use of well-known techniques (*e.g. k-means* algorithm) to cluster the tuples into some equivalent classes. The embedding of the watermark bit is based on the comparison of the parity of watermark bit and the LSB of candidate attribute. The *k-means* method assures the location of the embedded watermark irregular.

The watermark insertion phase in (99) works in three phases: (*i*) select a group of candidates in all attributes of the relation, and record it as the watermark schema; (*ii*) append the error correction code (ECC Code) to the watermark; (*iii*) executes the

watermark insertion algorithm. The insertion algorithm creates a pseudo random sequence using primary key and secret key. This sequence is used to identify the attribute to mark based on the significance of the attributes and the watermark bit to be embedded. During marking, the local constraint and a bidirectional mapping (to reduce watermarking data of various types into numeric data) are used. The local constraints can be defined as the upper bounds of "the distance" of attributes after/before watermarking. Finally global constraints which is a series of SQL statements are evaluated to decide whether to commit the changes. Observe that watermark schema selection in embedding phase and watermark schema detection in verification phase exploit the non-blindness property.

The schemes in (50, 51, 72) adopt watermarking scheme which follows same algorithmic steps as of (100) but embeds other meaningful watermark information rather than image by first converting it into a bit flow of certain length. In (50), the selection of the candidate attribute is based on the weights of all numeric attributes with a different hashing function. Observe that in watermark insertion phase of (72) the mark position is determined using the mark bit itself.

(b) Watermarking Based on Categorical Data Type Attribute:

Unlike the aforementioned watermarking schemes where the marking is based on numeric attribute, the right protection scheme in (178, 179) proposed by Sion et al. is based on categorical type data. The watermark embedding process starts with a relation with at least a categorical type attribute A (to be watermarked), a watermark wm and a set of secret keys (k_1, k_2), and other parameters (e.g., e which determines the percentage of tuples to mark). Using the primary key K and secret key k_1 and parameter e, it discovers a set of "fit" tuples, used to encode the mark. The fit tuple selection process is same as AHK algorithm. Suppose the database relation has η tuples, then fit tuples set contains roughly η/e tuples. The shorter watermark wm is converted into wm_data of length equal to η/e by deploying Error Correcting Code (ECC). The marking algorithm generates a secret value of required number of bits to represents all possible categorical values for attribute A depending on the primary key and k_1, and then, forcing its least significant bit to a value according to a corresponding (random, depending on the primary key and k_2) position in wm_data data. The pseudorandom nature of hash function $H(T_i(K), k_2)$) guarantees, on average, that a large majority of

the bits in *wm_data* data are going to be embedded at least once. The use of different key k_1 and k_2 ensures that there is no correlation between the selected tuples for embedding (selected by k_1) and the corresponding bit value positions in *wm_data* (selected by k_2). Moreover, they suggested to perform embedding based on multiple categorical attributes by considering not only the association between the primary key and single categorical attribute A but all association between primary key and categorical attributes to increase robustness of the scheme. Although this scheme is claimed to be robust against serious attacks (*e.g.* random attacks), however, the scheme is not suitable for database relations that need frequent updates, since it is very expensive to re-watermark the updated database relations. Though only a small part of selected tuples are affected by watermark embedding, the modifications of categorical attributes (*e.g.* change from "red" to "blue") in certain applications may be too significant to be acceptable. This watermarking technique is applied to binned medical data in a hierarchical manner (17).

(c) Watermarking Based on Non-Numeric Multi-Word Attributes:

Ali and Ashraf (8) proposed a watermarking scheme which is based on hiding binary image in spaces of non-numeric multi-word attributes of subsets of tuples, instead of numeric attribute at bit-level. The watermark is divided into m string each containing n bits. On the other hand, the database is also divided into non-intersecting subsets each containing m tuples. The m short strings of the watermark image are embedded into each m-tuple subset. The embedding is done as follows: suppose the integer representation of the i^{th}, $i \in [1 \ldots m]$, short string is d_i. A double space is created after d_i words of the pre-selected nonnumeric, multi-word attribute of i^{th} tuple in the subset. The extraction phase counts the number single spaces appearing before double space which indicates the decimal equivalent of the embedded short binary string. Since the proposed algorithm embeds the same watermark for all non-intersecting subsets of the database, it is robust against subset deletion, subset addition, subset alteration and subset selection attacks. Another advantage for space-based watermarking is that large bit-capacity available for hiding the watermark which may also facilitate embedding of multiple small watermarks. However, it may suffer from watermark removal attack if Mallory replaces all double spaces between two words (if exist) by single space for all tuples in the relation.

(d) Watermarking Based on Tuple or Attribute Insertion:

All the techniques discussed so far introduce distortion at attribute level. We now discuss two techniques where distortion is introduced at table level by inserting some fake tuples or additional attributes as watermarks.

Fake tuples as watermark information. The approach in (162) aims to generate fake tuples and insert them erroneously into the database. The fake tuple creation algorithm take care of candidate key attributes and sensitivity level of non candidate attributes. He uses Bernoulli sampling probability p_i for the i^{th} non-candidate attribute A_i to decide its fake value which may be chosen uniformly or as the value with higher occurrence frequency in the existing set of values of A_i in the relation. Unlike other algorithms, the detection algorithm is not an inverse algorithm to the watermark generating algorithm and insertion algorithm is probabilistic in nature. Detection algorithm checks to see whether the fake tuples inserted during watermark insertion phase, exist or has been changed. It checks it via primary key. As soon as it finds one match (i.e. identical or similar tuples), detection is done. The detection will fail for the watermarked database when all of the fake tuples are deleted by benign deletions. The number of fake tuples to be inserted is decided by the database owner. However, the watermark insertion phase must take into account the fact that the values of the fake tuples marks should not by any means degrade the quality of the data in the database and should not impact the query results. One advantage of this scheme is that the ownership can be publicly verified more than once until all the fake tuples are revealed and the scheme does not suffer from incremental updatability.

Virtual attribute as watermark information. Rather than inserting fake tuples, the author in (163) proposed another watermarking technique by inserting a virtual attribute in the relation which will serve as watermark containing parity checksum of all other attributes and an aggregate value obtained from any one of the numeric attribute of all tuples. The process of virtual attribute insertion is performed independently for each non-overlapping partitions obtained from the original relation. This scheme is designed to authenticate the tamper-proof receipt of the database over an insecure communication channel. Although this approach is fragile and can easily detect any

of the deletion or insertion or alter attacks, it suffers from the watermark removal attack.

(2) Distortion-free Watermarking

Most of the distortion free watermarking techniques are fragile in the sense that in addition to the ownership claiming, they aim at maintaining the integrity of the information in the database. The watermark insertion phase does not depend on any specific type of attribute and does not introduce any distortion in the underlying data of the database.

(a) Extracting Hash Value as Watermark Information:

In order to achieve the purpose of fragile watermark, authors in (19, 133) proposed watermarking schemes which are able to detect any modifications made to a database relation. These schemes are designed for categorical data that cannot tolerate distortion, hence, the watermark embedding is distortion free. In (133), partitioning of tuples is based on the hash value parameterized with primary key and secret key, whereas in (19), partitioning is based on categorical attribute values. After partitioning, the tuple level and group level hash values for each group are computed. In (133), a watermark of length equal to the number of tuple pairs in the group, is extracted from the group level hash value and for each tuple pair, the order of the two tuples are changed or unchanged according to their tuple hash values and the corresponding watermark bit. Moreover, Li (131) suggests to perform the exchange of tuples' positions based on Myrvold and Ruskeys linear permutation unranking algorithm to increase the embedding capacity. In these schemes, any modification of an attribute value will affect the watermarks in two groups as the modified tuple may be removed from one group and be added to the other group.

(b) Combining Owner's Mark and Database Features as Watermark Information:

The scheme proposed by Tsai et al. (187) aims at maintaining the integrity of the information in the database and is based on public authentication mechanism. The idea behind this scheme is that, first an watermark W is created which is a $\sqrt{n} \times \sqrt{n}$ white image, where n is the no. of tuples in the relation, besides four corners having

mark of the owner. It creates a value C_i ($0 \leq C_i \leq 255$) for each tuple t_i in the database using hash function MD5 and XOR operation. If there are n tuples in the database, it produces a feature C of length n by combining all C_i in order. Finally, a certification code R is produced by XOR-ing C and W. The encrypted form of R using private key is made available publicly. During verification the integrity of the relation T', in similar way, it generates feature C' from T'. After decryption using public key the certification code R is XOR-ed with C' that yield the watermark W'. The integrity of this extracted watermark proves the integrity of the database.

(c) Converting Database Relation into Binary Form used as Watermark Information:

The public watermarking scheme by Li and Deng (132) is applicable for marking any type of data including integer numeric, real numeric, character, and Boolean, without fear of any error constraints. The interesting features of this scheme is that it does not use any secret key and can be verified publicly as many times as necessary. The unique watermark key, used in both creation and verification phase, is public and obtained by one-way hashing from various information like the Identity of the owner(s) and characteristics of the database (*e.g.* DB Name, Version etc.). Observe that the public watermark key is different from the public-private key pair of asymmetric cryptography. This watermark key is used to generate a watermark W from the relation R. The watermark W is a database relation whose schema is $W(P, W_0, ..., W_{\gamma-1})$, where $W_0, \ldots, W_{\gamma-1} \in \{0, 1\}$. Compared to database relation R, the watermark W has the same number η of tuples and the same primary key attribute P . The number γ of binary attributes in W is a control parameter that determines the number ω of bits in W , where $\omega = \eta \times \gamma$ and $\gamma < \textit{number of attributes}$ in R. In the algorithm, a cryptographic pseudorandom sequence generator (*e.g.*, Linear Feedback Shift Register (89, 90)) to randomize the order of the attributes and the MSBs of the attribute values are used for generating the watermark W. The use of MSBs is for thwarting potential attacks that modify the data. Since the watermark key K, the watermark W, and the algorithm are publicly known, anyone can locate those MSBs. Any modification to these MSBs introduces intolerable errors to the underlying data and can easily be captured during verification phase. However, alteration of other bits in the data can not be detected by this scheme.

The watermarking techniques in (20, 21) follow the same algorithmic steps as of (132). The basic difference is that the former considers a private key instead of public, and thus, can not be publicly verifiable. In addition, the former is partition-based and considers the extracted binary watermark as an image which is used to prove the ownership. This image is treated as the abstract counterpart of the concrete relation R, and the abstraction is sound in the sense that concretization of the abstract image must cover R. However, the disadvantage of this scheme is that the extracted image may not have any meaningful pattern. As the alteration of other bits except MSB can not be captured during the detection phase, the soundness conditions holds for only those alteration functions that modify MSB. Therefore, the scheme fails to detect any tamper made to the bits except MSB.

(d) R-tree Based Permutation as Watermark:

In contrast to the traditional watermarking schemes, an R-tree data structure-based watermarking technique has been proposed in (113). The proposed technique takes advantage of the fact that R-trees do not put conditions on the order of entries inside the node. In the proposed scheme, entries inside R-tree nodes are rearranged, relative to a secret initial order (a secret key), in a way that corresponds to the value of the watermark. To achieve that, they proposed a one-to-one mapping between all possible permutations of entries in the R-tree node and all possible values of the watermark. Without loss of generality, watermarks are assumed to be numeric values. The proposed mapping employs a numbering system that uses variable base with factorial value. The detection rate of the malicious attacks depends on the nature of the attack, distribution of the data, and the size of the R-tree node. The proposed watermarking technique has the following desirable features: (*i*) It does not change the values of the data in the R-tree node but rather hides the watermark in the relative order of entries inside the R-tree node; (*ii*) It does not increase the size of the R-tree; (*iii*) The proposed technique does not interfere with R-tree operations; (*iv*) The performance overhead is minimal; (*v*) The integrity check does not require the knowledge of unwatermarked data (blind watermark).

4.1.6 Fingerprinting Techniques

The fingerprinting techniques proposed in the literature are based on numerical data type attributes.

Li et al. (134, 135) proposed an extension of agrawal and Kiernan's scheme (6) to embed fingerprint into the database. The basic difference is that instead of embedding meaningless bit pattern, they embed meaningful fingerprint where the fingerprint of length L (where $L > logN$, N=number of buyers) is computed from cryptographic hash function whose input is the concatenation of a secret key K (known by the merchant only) and user identifier n. The index of the fingerprint bit to be embedded is computed using hash function which is different from the hash used to select the bit positions, to ensure that the fingerprint bits are not correlated with the locations in which they are embedded. However, since the method is applicable for the relations with primary key only, (134) mentioned three different approaches to perform fingerprinting for a relation which has no primary key: (*i*) S-Scheme: the bits other than the least significant bits available for marking in the single numeric attribute of each selected tuples are considered as virtual primary key. But it suffers from both duplicate and deletion problem; (*ii*) E-Scheme: all numeric attributes of each selected tuples are examined independently by computing a virtual primary key from the attribute values. However, E-Scheme suffers from duplicate problem; (*iii*) M-Scheme: dynamically selects the bit positions used to construct a virtual primary key.

Liu et al. (137) proposed a block oriented finger printing scheme. For each buyer, the fingerprint is obtained using hash function based on the private key and buyer's ID. By combining the least significant bits of the table attributes, a two dimensional image is obtained and is divided into sub-images. Using Pseudo-random generator a bit is chosen in each sub-image and is XOR-ed with a fingerprint bit.

In (71), the authors proposed two level fingerprinting scheme to identify both the owner and the traitor. In the first embedding process, it embeds a unique fingerprint to identify each recipient to whom the relational data is distributed. In doing so, first the tuples are partitioned into m group, where m is the number of bits in the binary representation of the fingerprint. Next, the i^{th} fingerprint bit is embedded to the candidate bit position for selected tuples in the i^{th} group. The second embedding process is designed for verifying the extracted fingerprint and giving a numerical confidence

level. It uses the fingerprint itself as a secret key. The selected positions will be set to "1" or "0" depending on whether the hash value (seeded with secret key concatenating with Primary Key) is odd or even. To avoid conflict between the two embedding, only tuples not selected in the first embedding process are allowed to be marked in the second. The fingerprint extracting algorithm is the converse process to this first embedding process. It extracts a bit of the fingerprint from each group, and a numerical confidence level of each bit could be calculated. The bits those do not meet the pre-set confidence level are unreliable but could be localized. These bits could either be "1" or "0". Thus, a candidate set of suspect fingerprints can be obtained. In the fingerprint verification algorithm, they used each suspect fingerprint as the secret key to detect the pattern embedded in the second embedding process. Once the pattern is detected, the fingerprint is proved to be the exact originally embedded fingerprint at a high numerical confidence level.

4.1.7 Comparison

The classification and comparison of different schemes are depicted in Table 4.1 and 4.2. Table 4.1 depicts distortion-based watermarking and fingerprinting schemes, whereas distortion-free watermarking schemes are listed in Table 4.2. As the watermark detection phase for most of the schemes is probabilistic in nature, we explicitly mention when the detection phase is deterministic.

4.1.8 Probabilistic Issues

Consider n Bernoulli trials of an event, with probability p of success and $q = 1 - p$ of failure in any trial. Let $b(k; n, p)$ be the probability of obtaining exactly k successes out of n Bernoulli trials. Then,

$$b(k; n, p) = \binom{n}{k} p^k q^{n-k}$$

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \quad 0 \leq k \leq n$$

Let $B(k; n, p) = \sum_{i=k+1}^{n} b(i; n, p)$ which is the probability that more than k successes take place in n Bernoulli trials. Consider the robust watermarking scheme AHK algorithm

Table 4.1: Comparison of Distortion-based Watermarking and Fingerprinting Schemes

Proposed Schemes	Watermark Information	Cover Type	Granularity Level	Verifiability	Intent
AHK algorithms (4, 5, 6)	Meaningless Bit Pattern	Numeric	Bit-level	Blind, Private	Ownership Proof
Gupta et al. (74)	Meaningless Bit Pattern	Numeric	Multi-Bit level	Blind, Reversible, Private	Ownership Proof
Image-based (8, 186, 190, 203)	Image	Numeric or Non-Numeric Multi-Word	Bit-level or Whole Attribute Value or Character-level	Blind, Private	Ownership Proof and/or Tamper Detection
Speech-based (191)	Owner's Speech	Numeric	Bit-level	Blind, Private	Ownership Proof
Content-based (73, 202)	Database Content	Numeric	Multi-Bit level	Blind, Private	Ownership Proof and/or Tamper Detection and Localization
Cloud Model-based (201)	Cloud model with three characteristics: Expected value, Entropy and Hyper Entropy	Numeric	Whole Attribute Value	Non-Blind, Private	Ownership Proof
Categorical Attribute-based (178, 179)	Meaningful Binary String	Categorical	Bit-level	Blind, Private	Ownership Proof
Fake Tuple-based (162)	Fake Information obtained from database content	Database Table	Tuple-level	Blind, Private	Ownership Proof
Virtual Attribute-based (163)	Database content	Database Table	Attribute-level	Blind, Deterministic, Private	Tamper Detection
Others (72, 99, 102)	Meaningful Information of any type	Numeric	Bit-level	Blind or Non-Blind, Private	Ownership Proof
Fingerprinting Techniques (71, 134, 135, 137)	Meaningful Fingerprint Identifying Buyers uniquely	Numeric	Bit-level	Blind, Private	Traitor Detection

Table 4.2: Comparison of Distortion-free Watermarking Schemes

Proposed Schemes	Watermark Information	Cover Type	Granularity Level	Verifiability	Intent
Permutation-based (19, 133)	A Part of Group-level Hash Value	Tuples' Positions	Tuple-level	Blind, Private	Tamper Detection
Characteristic-based (187)	White Image with Owner's Mark at Four Corners	Nil	Nil	Blind, Public	Tamper Detection
Binary Form Relation (20, 21, 132)	Relation in Binary form	Nil	Nil	Blind, Public or Private	Ownership Proof
R tree-based Scheme (113)	Numeric Value Identifying Owner	Order of Entries in R-tree Nodes	R-tree Nodes	Blind, Private	Tamper Detection

(5, 6). In AHK algorithm, a watermark is successfully detected if number of match is more than the threshold τ which is just a percentage of the total number of embedded bits. Suppose ω is the total number of embedded bits. If the detection algorithm scans ω number of bits and observes the number of bits whose values match those assigned by the marking algorithm, the probability that at least τ out of ω random bits matches the assigned value is $B(\tau; \omega, 0.5)$. Thus, Alice should choose τ such that $B(\tau; \omega, 0.5) < \alpha$ where α is the false hit *i.e.* probability that Alice will discover her watermark in a database relation not marked by her. By choosing lower values of α, Alice can increase her confidence that if the detection algorithm finds her watermark in a suspected relation, it probably is a pirated copy. Suppose, Mallory knows the private parameters ν and ξ used by AHK algorithm. Since Mallory does not know the exact marked positions, he randomly chooses ζ tuples out of η tuples and flips all of the bits in all of ξ bit positions in all of ν attributes. The attack would be successful, if he flips at least $\bar{\tau} = \omega - \tau + 1$ marks. The probability that this attack will succeed is represented as $\sum_{\bar{\tau}}^{\omega} \frac{\binom{\omega}{i}\binom{\eta-\omega}{\zeta-i}}{\binom{\eta}{\zeta}}$.

Consider now the categorical attribute based schemes in (179). Suppose an attacker randomly alters q number of tuples and succeeds in each case to flip the embedded watermark bit with a success rate p, then the probability of success of altering at least r ($r < q$) watermark bits in the result is:

$$P(r, q) = \sum_{i=r}^{q} \binom{q}{i} \times p^q \times (1-p)^{(q-i)}$$

This metric illustrates the relationship between attack vulnerability and embedding bandwidth. Since only e tuple (on average) is watermarked, thus, Mallory effectively attacks only an average of a q/e tuples actually watermarked. If $r > q/e$, then $P(r, q) = 0$. For $r < q/e$, we have

$$P(r, q) = \sum_{i=r}^{q/e} \binom{q/e}{i} \times p^{q/e} \times (1-p)^{(q/e-i)}$$

Now consider a fragile watermarking scheme (133). Assume that each group consists of exactly ν tuples; thus, the length of each embedded watermark W is $\nu/2$. Let the j^{th} attribute of the i^{th} tuple is modified. This modification will affect the tuple hash value, group hash value and thus yield to watermark W'. The probability that this

modification can be detected *i.e.* $W \neq W'$ is, thus, $P = 1 - \frac{1}{2^{v/2}}$. If the value of the primary key has been modified. The probability that the tuple will be in the same partition is $1/g$ and probability of shifting to another group is $1 - 1/g$. Shifting of a tuple between groups affect the hash value of both groups. The probability that the modification can be correctly detected is $P = \frac{1}{g}(1 - \frac{1}{2^{v/2}}) + \frac{g-1}{g}(1 - \frac{1}{2^{(v-1)/2}})(1 - \frac{1}{2^{(v+1)/2}})$. Clearly, the probability in this case is less than that in the case of modifying non-primary key value.

Observe that most of the distortion-based watermarking techniques mainly aim at protecting the ownership, whereas distortion-free watermarking techniques mostly are fragile and aim at maintaining integrity of the database information. Although we classify the schemes based on different watermark information, most of the numerical distortion-based schemes follow almost similar steps to identify the candidate bit positions for the watermark. Finally, we observe that the usability of the watermarked database and queries still remains an open issue for future research.

4.2 Proposed Persistent Public/Private Watermarking Schemes

In our proposal, we address the notion of persistent watermarking of relational databases that serves as a way to recognize the integrity and ownership proof of the database while allowing the evaluation of its content by a set of SQL statements Q associated with it. We propose a novel *fragile* and *robust persistent* watermarking scheme that embeds both *private* and *public* watermarks where the former allows the owner to prove his ownership, while the latter allows any end-user to verify the correctness and originality of the data in the database without loss of strength and security. The persistency of the watermarks is preserved by exploiting the invariants of the database content: the public watermark is based on a part of the database state which remains invariant under processing by the SQL statements associated with the database, whereas private watermarking is based on an appropriate form of the original database state, called *abstract database*, and the *semantics-based properties* of the data which remain invariant under the processing by the associated SQL operations.

Before describing our proposal, let us define some basic notions.

(1) Persistent Watermark. Given a database dB and the set of applications interacting with the dB. Let Q be the set of SQL statements issued by the applications. We denote the database model by a tuple $\langle dB, Q \rangle$. It is worthwhile to mention that by Q we assume the set of all data manipulation operations SELECT, UPDATE, DELETE, INSERT.

Let the initial state of the database dB be d_0. When statements in Q are executed, its initial state d_0 changes and goes through a number of valid states $d_1, d_2 \ldots, d_{n-1}$. Let W be the watermark that is embedded in state d_0. The watermark W is persistent if we can extract and verify it blindly from any of the following $n-1$ states successfully.

Definition 8 (Persistent Watermark) *Let $\langle dB, Q \rangle$ be a database model where Q represents the set of SQL statements associated with the database dB. Suppose the initial state of dB is d_0. The processing of the SQL statements in Q over d_0 yield to a set of valid states $d_1, \ldots, d_{n-1}$. A watermark W embedded in state d_0 of dB is called persistent if*

$$\forall i \in [1..(n-1)], \;\; verify(d_0, W) = verify(d_i, W)$$

where $verify(d, W)$ is a boolean function such that the probability of "$verify(d, W) = true$" is negligible when W is not the watermark embedded in d.

(2) Static versus Non-static Database States. Consider a database model $\langle dB, Q \rangle$ where Q is the set of SQL statements associated with dB. Let dB goes through n valid states $d_0, d_1, \ldots, d_{n-1}$ under processing by the statements in Q. For any state d_i, $i \in [0..(n-1)]$, we can partition the data cells in d_i into two parts: *static* and *non-static*. Static part contains those data cells of d_i that are not affected by Q at all, whereas data cells in non-static part of d_i may change under processing by the statements in Q.

Let $CELL_{d_i}$ be the set of cells in state d_i of dB. We denote the set of static cells of d_i *w.r.t.* Q by $STC^Q_{d_i} \subseteq CELL_{d_i}$. For each tuple $t \in d_i$ we denote the static part of it by $STC^Q_t \subseteq STC^Q_{d_i}$. Thus, $STC^Q_{d_i} = \bigcup_{t_j \in d_i} STC^Q_{t_j}$.

Now we discuss how to determine the static and non-static parts of database states bounded with Q. As SELECT and INSERT statements in Q do not affect the existing data cells in a state at all, they do not take part in determining static/non-static part of the database state. However, DELETE statement may delete some cells form the static or non-static part, resulting in a subset of it. Thus, if $STC^Q_{d_i}$ and $(CELL_{d_i} - STC^Q_{d_i})$ represent the static and non-static part of a database state d_i *w.r.t.* Q respectively, a subset of it remains invariant over all the n valid states $d_0, d_1, \ldots, d_{n-1}$ under the

processing of DELETE statements in Q. The UPDATE statements modify values of the data cells in non-static part only. Let ATT^{update} be the set of attributes of dB that are targeted by the UPDATE statements in Q. Thus we can identify the set of cells $STC^{Q}_{d_i}$, $i \in [0..(n-1)]$ in state d_i *w.r.t.* Q corresponding to the attributes not in ATT^{update}, which remains invariant over all the n valid states.

(3) Semantics-based Properties of Data in Database States. Given a database state d_i, $i \in [0..(n-1)]$ of dB associated with a set of SQL statements Q, we can identify some semantics-based properties of the data in d_i *w.r.t.* Q. These properties include *intra-cell* (*IC*), *intra-tuple* (*IT*) or *Intra-attribute among-tuples* (*IA*) properties.

Intra-cell (IC) Property: In this case, individual cells of the relational databases represent some specific properties of interests. Let the possible values for a cell corresponding to the attribute Z be $a \leq Z \leq b$ over all the valid states, where a and b represent integer values. The *Intra-cell* (*IC*) property can be represented by $[a, b]$ from the domain of intervals. For instance, suppose a publisher is offering at most 50% discount to all the articles published by them. Thus, the article with price \$100 euro can be at least \$50 euro. The *intra-cell* property for this cell can be represented by the interval [50, 100]. Observe that, although the actual value of any cell may change under the processing by the SQL statements, the *IC* properties represented by the elements from non-relational abstract domains remain unchanged.

Intra-tuple (IT) Property: An *intra-tuple* property is a property which is extracted based on the inter-relation between two or more attribute values in the same tuple. As an example, we may consider inter-relation between the attributes *basic_price* and *total_price* in a database containing commodity information where *total_price* includes *basic_price* plus a percentage of *VAT* on the *basic_price*. This can be abstracted by the relational abstract domain, *viz* the domain of octagons (146). Examples of other relational abstract domain that can be used to represent *IT* properties are convex polyhedra, difference-bound matrices, linear equalities etc (31, 32, 145). Observe that although the update operation may modify the values of the inter-related attributes but their relational property should remain intact.

Intra-attribute among-tuples (IA) property: The *Intra-attribute among-tuples* (*IA*) property is obtained from the set of independent tuples in a relation. Examples of such property are: (*i*) in an employee database $\#male\ employee = \#female\ employee \pm 1$, where

# denotes cardinality of a set, (*ii*) average salary of male employees is greater than average salary of female employees, (*iii*) total number of female employees is greater than 3, etc. The first two can be abstracted by relational abstract domain, whereas last one can be represented by interval $[3, +\infty]$.

We denote the set of semantics-based properties obtained this way from state d_i *w.r.t.* Q by $P^Q_{d_i}$. For each tuple $t \in d_i$ we denote the set of *IC*, *IT* properties by $P^Q_t = IC^Q_t \cup IT^Q_t \subseteq P^Q_{d_i}$. Note that *IA* properties can not be determined at tuple level. Thus, $P^Q_{d_i} = \{\bigcup_{t_j \in d_i}(IC^Q_{t_j} \cup IT^Q_{t_j})\} \cup IA^Q_{d_i}$, where $IA^Q_{d_i}$ is the *Intra-attribute among-tuples* (*IA*) property in state d_i *w.r.t.* Q. Observe that the set of *IC*, *IT* and *IA* properties $P^Q_{d_i}$ remains invariant over all the n valid states $d_0, d_1, \ldots, d_{n-1}$ obtained by processing the statements in Q.

(4) Persistency in Abstract Database. In chapter 3, we proposed a sound approximation technique for database query languages based on the Abstract Interpretation framework where the values of the concrete database are replaced by abstract values from abstract domains representing some specific properties of interests, resulting into an abstract database. The abstract database provides a partial view of the data by disclosing the properties rather than their exact content. Consider an employee database that consists of a single table *emp* depicted in Table 4.3(a). Table 4.3(b) depicts an partial abstract database consisting of $emp^\sharp$ obtained by abstracting the basic and gross salaries of the employees in *emp* by the elements from domain of intervals.

Watermarking based on partial abstract databases which are obtained by abstracting only the data cells in non-static part ($CELL_d - STC^Q_d$) of the state d *w.r.t.* Q, results into a content-independent persistent watermark. This is because although the exact values in ($CELL_d - STC^Q_d$) may change under processing by the statements in Q, their properties represented by the abstract values remain invariant.

We are now in position to describe our proposed public and private watermarking scheme. In the rest of the chapter, we do not restrict ourself to any particular data type of the attributes. Attributes of any type including numeric, boolean, character, or any other can play roles in the public as well as private watermarking phase. Consider a database $dB(PK, A_0, A_1, A_2, \ldots, A_{\beta-1})$ in state d associated with a set of SQL statements Q, where PK is primary key. We divide the attribute set $\{A_0, A_1, A_2, \ldots, A_{\beta-1}\}$ into two

Table 4.3: A concrete and corresponding partial abstract employee database

(a) Concrete table *emp*

eID	Name	Basic Sal (euro)	Gross Sal (euro)	Age	DNo
E001	Bob	1000	1900	48	2
E002	Alice	900	1685	29	1
E003	Matteo	1200	2270	58	2
E004	Tom	600	1190	30	2
E005	Marry	1350	2542.5	55	1

(b) Partial abstract table $emp^{\sharp}$

$eID^{\sharp}$	$Name^{\sharp}$	$Basic\ Sal^{\sharp}$ (euro)	$Gross\ Sal^{\sharp}$ (euro)	$Age^{\sharp}$	$DNo^{\sharp}$
E001	Bob	[1000, 1300]	[1900, 2470]	48	2
E002	Alice	[900, 1170]	[1685, 2190.5]	29	1
E003	Matteo	[1200, 1560]	[2270, 2951]	58	2
E004	Tom	[600, 780]	[1190, 1547]	30	2
E005	Marry	[1350, 1755]	[2542.5, 3305.25]	55	1

parts *w.r.t.* Q: *Static attribute set* $A^{Q}_{static} = \{A^{s}_{0}, A^{s}_{1}, \ldots, A^{s}_{p-1}\}$ and *Non-static attribute set* $A^{Q}_{var} = \{A^{v}_{0}, A^{v}_{1}, \ldots, A^{v}_{q-1}\}$, where $p + q = \beta$. The set of static data cells STC^{Q}_{d} in state d *w.r.t.* Q corresponds to static attribute set A^{Q}_{static}, whereas the set of non-static data cells $(CELL_d - STC^{Q}_{d})$ corresponds to non-static attribute set A^{Q}_{var}. Although the primary key PK may be static in nature, we exclude it from the set A^{Q}_{static} and mention it separately in the rest of the chapter for better understanding.

The **Public watermark** is embedded into known locations of the host data with known methods to guarantee its public detectability. We identify most significant bit (MSB) positions of the data cells in STC^{Q}_{d} as the location for public watermark. We avoid non-static data cells because their values keep changing under processing by the statements in Q. This ensures the persistency of the public watermark. Since the public watermark in the host data is visible to all end-users, it is highly possible that attackers try to remove or distort it. We achieve the robustness of the public watermark by choosing only most significant bit positions of the host data as the location for public watermark so that any malicious change of the MSB makes the data useless and also reflects to the hash value, resulting the verification unsuccessful. Moreover, our scheme is designed to be fragile by using cryptographic hash value of each tuple so as to detect and locate any modifications when attackers try to modify the data in the database while keeping the watermark untouched.

The **private watermarking** is based on two invariants of the database states: *Semantics-based properties of the data* and *Partial abstract database*, so as to maintain the persistency of the watermark under processing by the SQL statements associated with the database. The security of the private watermarking relies on the secret key as well as the level of abstraction used. Attackers do not know which properties are used to abstract the database. In addition, private watermarking is also based on MSBs of the attribute values. We assume the secret key to be large enough to thwart Brute force attack.

It is worthwhile to mention that, unlike existing techniques (5, 20, 132, 202), the verification phase of the proposed scheme is deterministic. Moreover, the watermarking does not introduce any distortion to the underlying data, hence it is distortion-free.

4.2.1 Public Watermarking

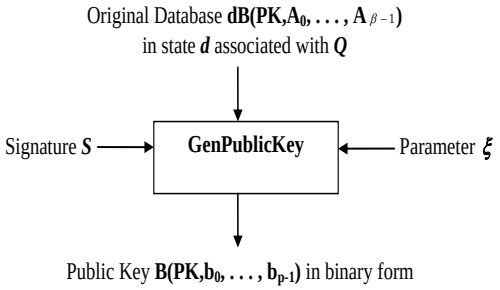
The overall architecture of the public watermarking phase is depicted in Figure 4.2. It consists of a single procedure, called **GenPublicKey**. The inputs of **GenPublicKey** are the database $dB(PK, A_0, A_1, A_2, \ldots, A_{\beta-1})$ in state d associated with a set of SQL statements Q, the signature S of the database owner which is known to all end-users, and a parameter ξ representing the number of most significant bits (MSBs) available in attributes. The procedure generates a table $B(PK, b_0, \ldots, b_{p-1})$ where PK is primary key, p is the number of attributes in A^Q_{static} and $\forall j \in [0..(p-1)]$: b_j contains either 1 or 0. The binary table B is treated as public key and made available to all end-users. Later, when any end-user wants to verify the source of a suspicious database, she uses B as the public key to generate and verify the embedded signature.

The algorithm of **GenPublicKey** is depicted in Figure 4.3. Let us describe it in details.

Let $|S|$ be the length of the signature S in binary form. We divide S into m blocks $\{S_0, S_1, \ldots, S_{m-1}\}$ each of length p, where p is the number of attributes in A^Q_{static} and $m = \lceil \frac{|S|}{p} \rceil$. If the length of the last block is less than p, we append 0s to make it of length p.

For each tuple $t \in d$, the algorithm generates an hash value h in binary form of length p from its primary key and its static part $STC^Q_t = \{t.A^s_0, \ldots, t.A^s_{p-1}\}$. We exclude the dynamic part of the tuples in computing hash because it keeps changing under the processing of SQL operations. While computing hash values, we assume the

Figure 4.2: Overall architecture of Public Watermarking Phase



Performed by: Database Owner

following so that any minor tampering of the data can be captured through the hash value: the hash function is one-way function; it is almost infeasible to generate the same hash value from two different messages; all bits in the message provide the same contribution to the hash value.

The HASH function we might use takes a parameter p and generates a binary hash value of length p: we can use Merkle-Damgård's Meta method (143) where the length of initial hash value and the length of each block of the binary string obtained from "$t.PK||t.A_0^s|| \ldots ||t.A_{p-1}^s$" (where || stands for concatenation operation) is considered to be p.

Using pseudorandom sequence generator *PRSG* (*e.g.* Linear FeedBack Shift Register) seeded by tuple's primary key $t.PK$, we identify which group the tuple belongs to. If the tuple t belongs to i^{th} group, we compute $w = h \otimes S_i$ where h is the binary hash value of length p and S_i is the i^{th} block of the binary signature S. In other words, we embed i^{th} block S_i of the signature S into all tuples that belong to the i^{th} group. This ensures the existence of the signature during verification phase if there exists at least one marked tuple in each group after the processing of *DELETE* operations over the database state. Observe that w is of length p.

Corresponding to tuple t, we now create a binary tuple r in $B(PK, b_0, \ldots, b_{p-1})$ whose primary key PK is the same as that of t, *i.e.* $r.PK = t.PK$. For each static attribute $A_j^s \in A_{static}^Q$ where $j = 0, \ldots, (p-1)$, we obtain a MSB bit position in the corresponding data cell $t.A_j^s$ by computing $k = HASH(t.PK||t.A_j^s)\%\xi$ where ξ is the number of MSBs available in A_j^s. The value of the j^{th} attribute b_j of r is, thus, r.b_j= k^{th} MSB of $t.A_j^s \otimes w[j]$.

Figure 4.3: Algorithm for Signature Embedding and Public Key Generation

Algorithm 1: GenPublicKey

Input: Database $dB(PK, A_0, A_1, A_2, \ldots, A_{\beta-1})$ in state d associated with a set of SQL statements Q, Owner's signature S, Parameter ξ representing the no. of MSBs available in attributes
Output: A publicly available binary table $B(PK, b_0, \ldots, b_{p-1})$

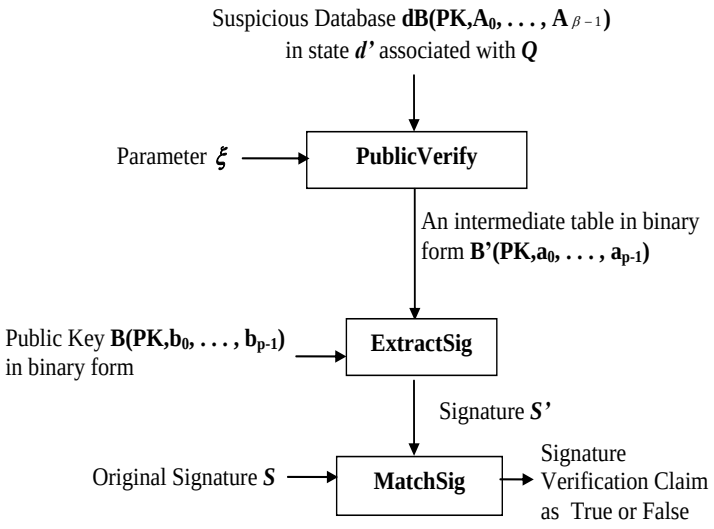
1. Identify $A^Q_{static} = \{A^s_0, A^s_1, \ldots, A^s_{p-1}\}$
2. Compute $m = \lceil \frac{|S|}{p} \rceil$, where $|S|$ denotes length of signature S in binary form and p=no. of attributes in A^Q_{static}.
3. Split the signature S into m blocks $\{S_0, S_2, \ldots, S_{m-1}\}$ where $|S_i| = p$.
4. FOR each tuples $t \in d$ DO
5. $h = HASH(t.PK||t.A^s_0|| \ldots ||t.A^s_{p-1}, p)$
6. $i = PRSG(t.PK)\%m$
7. $w = h \otimes S_i$
8. Generate a binary tuple r in $B(PK, b_0, \ldots, b_{p-1})$ with $r.PK = t.PK$
9. FOR $j = 0 \ldots p-1$ DO
10. $k = HASH(t.PK||t.A^s_j)\%\xi$
11. $r.b_j = k^{th}$ MSB of $t.A^s_j \otimes w[j]$
12. END FOR
13. END FOR
14. Return B

We perform similar operations for all tuples in state d of dB, and finally we get a binary table $B(PK, b_0, \ldots, b_{p-1})$ consisting of a set of binary tuples generated this way. This binary table B is then made publicly available and treated as public key which is later used by end-users to verify the embedded signature S. We assume that the database contains primary key. However, for the databases that do not contain primary key we can follow the virtual primary key schemes in (134).

Signature Verification

Figure 4.4 depicts an overall architecture of the signature verification phase performed by end-users. The procedure **PublicVerify** takes a suspicious database $dB(PK, A_0, \ldots, A_{\beta-1})$ in different state d' as input, and generates an intermediate binary table $B'(PK, a_0, \ldots, a_{p-1})$. Based on this intermediate binary table $B'(PK, a_0, \ldots, a_{p-1})$ and the public key $B(PK, b_0, \ldots, b_{p-1})$ which is generated by the database owner in watermarking

Figure 4.4: Overall architecture of publicly Signature Verification phase



Performed by: End-Users

phase, the procedure **ExtractSig** extracts a signature S'. Finally, **MatchSig** compares S' with the original signature S. If it matches, the verification claim is true, otherwise false.

The algorithms of the procedures **PublicVerify** and **ExtractSig** are depicted in Figure 4.5 and 4.6 respectively. For each tuple $t \in d'$, Algorithm **PublicVerify** generates a binary tuple r' in $B'(PK, a_1, \ldots, a_{p-1})$ whose primary key is equal to the primary key of t, *i.e.* $r'.PK = t.PK$. The binary values of the attributes a_j, $j \in [0..(p-1)]$ in r' are obtained as follows: (*i*) compute a binary hash value h of length p from the primary key $t.PK$ and static part $STC_t^Q = \{t.A_0^s, \ldots, t.A_{p-1}^s\}$ in similar way as in Algorithm **GenPublicKey**, (*ii*) extract the k^{th} MSB from $t.A_j^s$ in similar way as in Algorithm **GenPublicKey**, (*iii*) compute $a_j = k^{th}$ MSB of $t.A_j^s \otimes h[j]$, where $h[j]$ represents j^{th} bit of h. In this way, the algorithm generates a set of binary tuples from the tuples in database state d', and collection of these binary tuples forms the table B'.

Procedure **PublicVerify** then calls another procedure **ExtractSig**, and passes the binary table B' and the public key B (generated by the owner in watermarking phase). **ExtractSig** finds the pairs of tuples (r, r′) where $r \in B$ and $r' \in B'$ such that their

Figure 4.5: Algorithm to Verify Signature

Algorithm 2: PublicVerify

Input: Database $dB(PK, A_0, A_1, A_2, \ldots, A_{\beta-1})$ in state d' associated with Q, Parameter ξ, Public key $B(PK, b_0, \ldots, b_{p-1})$, Owner's Signature S
Output: Signature Verification Claim as True or False

1. Identify $A^Q_{static} = \{A^s_0, A^s_1, \ldots, A^s_{p-1}\}$
2. FOR each tuples $t \in d'$ Do
3. $h = HASH(t.PK||t.A^s_0|| \ldots ||t.A^s_{p-1}, p)$
4. Construct a tuple r' in $B'(PK, a_0, \ldots, a_{p-1})$ such that $r'.PK = t.PK$
5. FOR $j = 0 \ldots p-1$ DO
6. k=HASH(t.PK, $t.A^s_j$) % ξ.
7. $r'.a_j = k^{th}$ MSB in $t.A^s_j \otimes h[j]$.
8. END FOR
9. END FOR
10. S'=**ExtractSig**(B,B')
11. Return **MatchSig**(S, S')

primary keys are same *i.e.* $r.PK = r'.PK$. It then performs attribute-wise XOR *i.e.* $r.b_j \otimes r'.a_j$ for all $j \in [0..(p-1)]$, excluding the primary key attribute, and concatenate them to obtain a binary string *str*. If the tuple r and r' belongs to the i^{th} group which is determined from the pseudo random sequence generator *PRSG* seeded by *r.PK* or $r'.PK$, the corresponding *str* denotes the i^{th} block S'_i of signature S'. This way we can collect all strings *str* from the tuples belonging to the i^{th} group and put them into the buffer $buff[i]$. If no tampering occurred, all strings in $buff[i]$ will be same and represent S'_i. However, when data is tampered, some strings *str* in $buff[i]$ may be different from the others. In such case, function $MajorityVote()$ returns the string with maximum match. In this way, we can determine $S'_1, \ldots, S'_m$ by extracting *str* from tuples belonging to m different groups. By concatenating them, finally we get the signature S'. The procedure **MatchSig** returns true when S' matches with the original signature S, otherwise it returns False.

Example 7 *Consider the employee database of Table 4.3(a) where eID is the primary key. Suppose the set of SQL statements Q associated with the database are only able to increase the basic and gross salary of the employees by at most 30%. As only the basic and gross*

Figure 4.6: Algorithm to Extract Signature

Algorithm 3: ExtractSig

Input: Public key $B(PK, b_0, \ldots, b_{p-1})$ and Binary table $B'(PK, a_0, \ldots, a_{p-1})$
Output: Signature S'

1. Find binary tuple $r \in B$ and $r' \in B'$ such that $r.PK == r'.PK$.
2. FOR all pair (r, r') DO
3. str=NULL
4. FOR $j = 0 \ldots p-1$ DO
5. Perform $str = str||r.b_j \otimes r'.a_j$
6. END FOR
7. $i = PRSG(r.PK)\%m$
8. $buff[i] \leftarrow str$
9. END FOR
10. FOR $i = 0 \ldots m-1$ DO
11. S'_i=$MajorityVote(buff[i])$
12. END FOR
13. Return $S' = S'_0||S'_1||\ldots||S'_i||\ldots||S'_{m-1}$

salary can possibly be modified by the statements, we get $A^Q_{static} = \{Name, Age, Dno\}$ *and* $A^Q_{var} = \{Basic\ Sal, Gross\ Sal\}$.

Let the signature of the database owner be S = "RAJU" which is public and known to all end-users. Suppose the binary representation of S is 01010010010000010100101001010101, obtained by concatenating the ASCII codes of the characters in S. Since $|S| = 32$ *and the number of static attributes in* A^Q_{static} *is p=3, we divide S into* $m = \lceil \frac{|S|}{p} \rceil = \lceil \frac{32}{3} \rceil = 11$ *blocks i.e. 010 100 100 100 000 101 001 010 010 101 010. Observe that each block has length* $p = 3$. *Since last block contains only 2 bit, we append 0 to make it of length 3.*

Consider the tuple t=$\langle E001,\ Bob,\ 1000,\ 1900,\ 48,\ 2\rangle$. *The static part of t is* $STC^Q_t = \langle t.Name, t.Age, t.Dno\rangle = \langle Bob, 48, 2\rangle$. *Let the binary hash value of length* $p = 3$ *obtained from its primary key* $t.eID = E001$ *and its static part* STC^Q_t *be* $h = HASH(E001||Bob||48||2, 3) = 001$. *Based on the random value generated from the pseudorandom sequence generator PRSG seeded by* $t.PK = E001$, *suppose we determine that t belongs to third partition i.e.* $i = 2$. *Therefore, we compute* $w = h \otimes S_2 = 001 \otimes 100 = 101$.

Corresponding to tuple t, we now create a binary tuple r in $B(PK, b_0, b_1, b_2)$ *with* $r.PK = t.PK = E001$.

We now describe how to compute $r.b_0$, $r.b_1$ *and* $r.b_2$: *let the value of the parameter* ξ *be 4. For each of the three static attribute values t.Name = "Bob", t.Age = "48" and t.Dno = "2"*

in STC_t^Q, we compute most significant bit position k and extract the k^{th} most significant bit from them. Suppose we get the value of k as 2, 3, 1 for "Bob", "48" and "2" respectively. We extract the 2^{nd} MSB of "Bob", the 3^{rd} MSB of "48", and the 1^{st} MSB of "2". Let the extracted bits be 0, 1, 1 respectively. Thus, $r.b_0 = 0 \otimes 1 = 1$, $r.b_1 = 1 \otimes 0 = 1$ and $r.b_2 = 1 \otimes 1 = 0$, and the corresponding binary tuple r in $B(PK, b_0, b_1, b_2)$ is $\langle E001, 1, 1, 0\rangle$. We compute the same for other tuples in state d, and finally we obtain a binary table B which is then made publicly available.

Now we illustrate the verification phase. Consider the tuple t=$\langle E001, Bob, 1000, 1900, 48, 2\rangle$ in a pirated table emp′. The static part of t is STC_t^Q=$\langle Bob, 48, 2\rangle$. We compute the binary hash value h of length $p = 3$ in similar way, and we obtain $h = 001$. We now construct a binary tuple r′ in $B'(PK, a_0, a_1, a_2)$ as follows: (i) $r'.PK = t.PK = E001$, (ii) for attribute values "Bob", "48" and "2", we get MSB position k as 2, 3 and 1 respectively. Thus, a_0=$0 \otimes 0 = 0$, a_1=$1 \otimes 0 = 1$ and a_2=$1 \otimes 1 = 0$, and the binary tuple r′ in $B'(PK, a_0, a_1, a_2)$ is $\langle E001, 0, 1, 0\rangle$.

When we call the procedure **ExtractSig**, *it finds two binary tuples $r = \langle E001, 1, 1, 0\rangle \in B$ and $r' = \langle E001, 0, 1, 0\rangle \in B'$, and it generates the string $str = 1 \otimes 0 || 1 \otimes 1 || 0 \otimes 0 = 100$. Since the tuples r and r′ belong to the 2^{nd} group which is determined from the pseudorandom sequence generator seeded by $r.PK = E001$, we get that the string $str = 100$ represents the 2^{nd} block S'_2 of the signature S′. In similar way we can extract all 11 blocks $S'_0, \ldots, S'_{11}$ of the signature S′ from the tuples in d′ belonging to 11 different groups, and by concatenating them we get* 01010010010000010100101001010101 *which is same as the original signature S = "RAJU".*

4.2.2 Private Watermarking

The private watermarking algorithm **PrivateWatermark** is depicted in Figure 4.7. The inputs of the algorithm are the original database $dB(PK, A_0, A_1, \ldots, A_{\beta-1})$ in state d bounded with a set of SQL statements Q and a secret key K. It generates a private binary watermark PW whose schema is $PW(PK, c_0, \ldots, c_{\beta-1}, p_0, p_1, p_2)$.

The algorithm generates a partially abstract database state $d^\sharp$ from the original state d by abstracting the data cells belonging to non-static part ($CELL_d - STC_d^Q$) only. For each tuple $t^\sharp$ in $d^\sharp$, the algorithm generates a tuple r in $PW(PK, c_0, \ldots, c_{\beta-1}, p_0, p_1, p_2)$ whose primary key is equal to the primary key of $t^\sharp$ just to identify the tuples in PW uniquely and to perform matching in the verification phase. Note that as the primary key attribute is static in nature we never abstract its values. The algorithm, then, adds three values for the attributes p_0, p_1 and p_2 in r that correspond to the encoded values of

Figure 4.7: Private Watermarking Algorithm

Algorithm 4: PrivateWatermark

Input: Database $dB(PK, A_0, \ldots, A_{\beta-1})$ in state d bounded with a set of SQL statements Q, Secret key K
Output: A private binary watermark $PW(PK, c_0, \ldots, c_{\beta-1}, p_0, p_1, p_2)$

1. Obtain Partially Abstract Database $dB^\sharp(PK^\sharp, A_0^\sharp, \ldots, A_{\beta-1}^\sharp)$ in state $d^\sharp$ by abstracting non-static part ($CELL_d - STC_d^Q$) only;
2. Determine *intra − attribute among − tuples* property $IA_{d^\sharp}^Q$;
3. FOR each tuple $t^\sharp \in d^\sharp$ DO
4. Construct tuple r in PW with primary key $r.PK = t^\sharp.PK^\sharp$;
5. Determine $IC_{t^\sharp}^Q$, $IT_{t^\sharp}^Q$;
6. $r.p_0 = g_{encode}^p(IC_{t^\sharp}^Q)$;
7. $r.p_1 = g_{encode}^p(IT_{t^\sharp}^Q)$;
8. $r.p_2 = g_{encode}^p(IA_{d^\sharp}^Q)$;
9. FOR (i=0; i< β; i=i+1) DO
10. $val = G_i(K \circ t^\sharp.PK^\sharp \circ t^\sharp.A_0^\sharp \circ \cdots \circ t^\sharp.A_{\beta-1}^\sharp)$;
11. $j = val\%(no.\ of\ attributes\ in\ t^\sharp)$;
12. $r.c_i$= (MSB of j^{th} attribute in $t^\sharp$);
13. delete the j^{th} attribute from $t^\sharp$;
14. END FOR
15. END FOR
16. Return PW;

intra-cell, *intra-tuple* properties for $t^\sharp$ and encoded value of *intra-attribute among-tuples* property for the whole database state $d^\sharp$, where g_{encode}^p represents an encoding function. G_i represents a pseudorandom sequence generator that returns i^{th} random value *val* when it is seeded by the attribute values of $t^\sharp$ including its primary key, and the secret key K. For all i from 0 to $\beta - 1$, *val* chooses an attribute randomly in $t^\sharp$ excluding the primary key and consider its MSB as the binary value for c_i in r. While computing the seed value for G_i or extracting MSBs, if there is any problem with abstract form of the values we can use its encoded form too. For instance, we can encode any interval by using the Chinese Remainder Theorem (166).

Observe that since the binary tuples in PW are constructed from the semantics-based properties and partially abstract database information, the private watermark PW is invariant under processing by the statements in Q. The inputs of the verification

algorithm are the database in state d' bounded with Q, secret Key K, and the output is a binary table PW'. We use a boolean function $match(PW, PW')$ to compare PW' with the original private watermark PW which is obtained in the private watermarking phase. Note that the function $match(PW, PW')$ compares tuple by tuple taking into account the the primary key of the tuples in PW and PW'. As tuples may be deleted from or added to the initial state d and yield to a different state d', only those tuples whose primary keys are common in both PW and PW' are compared. If $match(PW, PW') = True$, then the claim of the ownership is true, and false otherwise. Observe that the verification phase is deterministic rather than probabilistic (132), as we compare and verify tuples in PW' against the tuples in PW with the same primary key only, and the binary values of the attributes of tuples in PW are invariant.

Example 8 *Consider the employee database that consists of a table emp with eID as the primary key as shown in Table 4.3(a) where we determine that* $A^Q_{static} = \{Name, Age, Dno\}$ *and* $A^Q_{var} = \{Basic\ Sal, Gross\ Sal\}$ *w.r.t. the SQL statements that are only able to increase the basic and gross salary of the employees by at most 30%.*

The partially abstract table $emp^\sharp$ *is shown in Table 4.3(b) where data cells corresponding to the non-static attribute set* A^Q_{var} *are abstracted by elements from the domain of intervals. Consider an abstract tuple* $t^\sharp$*, say,* $\langle E002, Alice, [900, 1170], [1685, 2190.5], 29, 1\rangle$ *in* $emp^\sharp$*. Corresponding to* $t^\sharp$ *we create a tuple r in watermark table* $PW(PK, c_0, \ldots, c_{\beta-1}, p_0, p_1, p_2)$ *with* $r.PK = E002$*.*

In $t^\sharp$*, the abstract values of the basic and gross salary are* $[900, 1170]$ *and* $[1685, 2190.5]$ *respectively. These abstract values represent the intra-cell (IC) properties for* $t^\sharp$*. The relation between the two attributes* $Basic\ Sal^\sharp$ *and* $Gross\ Sal^\sharp$ *can be represented, for instance, by the following equation:* $Gross\ Sal^\sharp \geq \frac{(165 \times Basic\ Sal^\sharp)}{100} + 200$*, assuming that* $Gross\ Sal^\sharp$ *includes Basic* $Sal^\sharp$*, 65% of the Basic* $Sal^\sharp$ *as PF, HRA etc and minimum of 200 euro as incentive. Thus, the intra-tuple (IT) property can be obtained by abstracting the above relation by the elements from the domain of polyhedra (31) i.e. by the linear equation just mentioned. The intra-attribute among-tuples property may be: "The number of employees in every department is more than 2". This can also be represented by* $[3, +\infty]$ *in the domain of intervals. Suppose after encoding these three properties, we obtain the encoded values* k_1*,* k_2*,* k_3*. Therefore, the values of the attributes* p_0*,* p_1*,* p_2 *in r will be* k_1*,* k_2*,* k_3 *respectively.*

Suppose the random selection of the attributes in $t^\sharp$ *based on the random value generated by the pseudorandom sequence generator yield to the selection order as follows:* $\langle [1685, 2190.5], 1, 29, [900, 1170], Alice\rangle$*. We choose MSB from these attribute values in this order. For instance, for the abstract values (represented by intervals), we may extract the MSB from the encoded*

values which are obtained by using Chinese Remainder Theorem (166). Let the extracted MSBs be 0, 1, 1, 0, 1 respectively. Thus the tuple r in PW would be $\langle E002, 0, 1, 1, 0, 1, k_1, k_2, k_3\rangle$.

After performing similar operations for all the tuples in all partitions, the watermark PW is generated.

4.2.3 Time Complexity

The time complexity to generate public key B (in case of public watermarking phase) and to generate private watermark PW (in case of private watermarking phase) depends on the number of tuples η in the original database state d linearly. That is, time complexity of the algorithms **GenPublicKey** and **PrivateWatermark** is $O(\eta)$ where η is the number of tuples in the original database state. Similarly, time complexity of the public and private watermark verification phases is also $O(\eta)$.

Given a database $dB(PK, A_0, A_1, A_2, \ldots, A_{\beta-1})$, the number of attributes in public watermark B is $p + 1$, where p is the number of static attributes in A^{Q}_{static}. Suppose η is the number of tuples in the original database state. The total number of cells in public watermark B is, thus, $(p + 1) \times \eta$. If δ is the number of bits required to represent the primary key, the total number of bits in B is $(\delta + p) \times \eta$. Thus, the space complexity can be represented by $O(\eta)$.

Similarly we can show that, for a relation with η tuples, the total number of bits in private watermark PW is $(\nu + \beta) \times \eta$ where β is the number of attributes except the primary key, and ν is the total number of bits required to represent the primary key and three semantics-based properties p_0, p_1, p_2. Therefore, the space complexity in case of private watermarking can also be represented by $O(\eta)$.

4.2.4 Discussions

Before concluding, let us briefly discuss the properties of our proposal and relate it with the existing techniques in the literature.

Our proposed public and private watermarking scheme has the following properties: (*i*) it is blind, (*ii*) it does not introduce any distortions to the underlying data, and thus never degrades the usability of the data in the database, (*iii*) it preserves the persistency of both public and private watermarks, (*iv*) public watermarking is robust as well as fragile, (*v*) there is no need of recomputation when tuples are modified by the

UPDATE statements. (*vi*) verification phase is deterministic rather than probabilistic and can, thus, reduce the false positive and false negative.

Although the public watermarking algorithm of (132) is robust, it is not fragile: attackers can easily tamper the data by keeping the MSBs unchanged. Observe that our scheme uses cryptographic hash value obtained from the static part of each tuple. Thus, any modification of the attribute values of a tuple reflects to the hash value and results the signature extraction from that tuple unsuccessful. In other words, any modification is narrowed down to each tuple.

The watermark embedding phase in (73, 133, 187, 202) is content-dependent. Any intentional processing of the database content may damage or distort the existing watermark, resulting the persistency of it into a risk. Our scheme is designed in such a way to preserving the persistency of the watermark by exploiting the invariants of the database state.

The watermark detection algorithm of (5, 20, 132, 202) is parameterized with a threshold value. The lower the value of the threshold, the higher is the probability of a successful verification. We strictly improve on these techniques by exploiting the invariants of the database state and by keeping the identity of the binary tuples in the public key B and in the private watermark PW. This makes the verification phase in both cases deterministic.

We can randomize the selection of MSB in private watermarking by introducing a parameter ξ where ξ represents the number of most significant bits available. We choose j^{th} bit from the ξ available MSBs of the attribute A using the following equation: $j = hash(PK \circ A) \ mod \ \xi$.

As suggested in (132), we can also use the *watermark generation* parameter $x \leq \beta$, that controls the number of attributes in private watermark PW. In such case, the schema of PW will be $W(PK, c_1, \ldots, c_x, p_0, p_1, p_2)$. Thus the number of attributes in watermark would be $(\nu + x) \times \eta$, where η is the number of tuples and ν is the total number of bits required to represent primary key and properties p_0, p_1, p_2.

Although the notion of persistency is a crucial issue in the context of database watermarking, to our best knowledge this is the first attempt to address this issue. There are very few works on public watermarking of relational databases and all are distortion-free. In order to avoid the transmission of the public watermarks separately on demand to the end-users for verification purpose, there is a scope to embed the

generated persistent public watermark information into the less significant part of the underlying data, resulting into a distortion-based public watermarking scheme.

Chapter 5

Observation-based Fine Grained Access Control (OFGAC)

[Part of this chapter is already published in (81, 85, 86)]

Due to emerging trend of Internet and database technology, the information systems are shared by many people all around the world. With more and more information being shared, exchanged, distributed or published through the web, it is important to ensure that sensitive information is being accessed by the authorized users only. Disclosure of sensitive information to unauthorized users may cause a huge loss to the enterprizes or organizations. Access control mechanism (15, 70, 110) emerged as a most effective solution to ensure the safety of the information in Relational Database Management System (RDBMS) or eXtensible Markup Language (XML) documents. The granularity of the traditional access control mechanism is coarse-grained and can be applied only at database/table level for RDBMS or file/document level for XML. Therefore, an XML file containing data with both public and private protection requirements, for instance, will have to be split into two files before applying the traditional access control. The need of more flexible business requirements and security policies mandate the use of Fine Grained Access Control (FGAC) mechanisms (54, 112, 121, 138, 167, 174, 192, 204, 205) that provide the safety of the database information even at lower level such as individual tuple/cell level for RDBMS or element/attribute level for XML without changing their original structure.

The traditional fine grained disclosure policy p determines the part of the database

state to be allowed to disclose while answering the queries Q. It splits any database state into two distinct parts: a public one (insensitive data) and a private one (sensitive data). Generic users are able to see the information from public part only, while the private part remains undisclosed. We denote a database state σ under a disclosure policy p by a tuple $\sigma_p = \langle \sigma_h, \sigma_l \rangle$, where σ_h and σ_l represent the states corresponding to the private and public part respectively. Given a database state σ_p under a policy p and a query Q, the execution of Q on σ_p returns the result $\xi = S[\![Q]\!](\sigma_p)$. In reality, p depends on the context in which queries are issued, for instance, the identity of the issuer, the purpose of the query, the data provider's policy etc.

As far as the security of the system is concerned, the disclosure policy should comply with the *non-interference* policy (168), *i.e.* the results of admissible queries should not depend on the confidential data in the database. The *non-interference* says that a variation of private (sensitive) database values does not cause any variation of the public (insensitive) view (see Definition 9).

Definition 9 (Non-interference) *Let $\sigma_p = \langle \sigma_h, \sigma_l \rangle$ and $\sigma'_p = \langle \sigma'_h, \sigma'_l \rangle$ be two database states under the disclosure policy p. The non-interference policy says that*

$$\forall Q, \forall \sigma_p, \sigma'_p : \ \ \sigma_l = \sigma'_l \implies S[\![Q]\!](\sigma_p) = S[\![Q]\!](\sigma'_p)$$

That is, if the public (insensitive) part of any two database states under disclosure policy p are the same, the execution of any admissible query Q over σ_p and σ'_p return the same results.

In the context of information flow security, the notion of non-interference is too restrictive and impractical in some real systems where intensional leakage of the information to some extent is allowed with the assumption that the power of the external observer is bounded. Thus, we need to weaken or downgrading the sensitivity level of the database information, hence, the notion of non-interference which considers weaker attacker model. The weaker attacker model characterizes the observational characteristics of the attacker and can be able to observe specific properties of the private data. Example 9 illustrates this situation with a suitable example.

Example 9 *Consider an XML document that stores customers' information of a bank. Figure 5.1(a) and 5.1(b) represent the Document Type Definition (DTD) and its instance respectively. According to the DTD, the document consists of zero or more "customer" elements with three different child elements: "PersInfo", "AccountInfo", "CreditCardInfo" for each customer. The*

"CreditCardInfo" for a customer is optional, whereas each customer must have at least one bank account represented by "AccountInfo". The element "PersInfo" keeps the record of personal information for the customers.

*Suppose, according to the access control policy, that employees in the bank's customer-care section are not permitted to view the exact content of IBAN and credit-card numbers of the customers, while they are allowed to view only the first two digits of IBAN numbers and the last four digits of credit card numbers, keeping other sensitive digits hidden. For instance, in case of the 12 digits credit card number "4023 4581 8419 7835" and the IBAN number "IT10G 02006 02003 000011115996", a customer-care personnel is allowed to see them as "**** **** **** 7835" and "IT*** ***** ***** ***********" respectively, just to facilitate the searching of credit card number and to redirect the account related issues to the corresponding country (viz, "IT" stands for "Italy"). In addition, suppose the policy specifies that the expiry dates and secret numbers of credit cards and the deposited amounts in the accounts are fully-sensitive and completely hidden to them. The traditional FGAC mechanisms are unable to implement this scenario as the IBAN numbers or credit card numbers are neither private nor public as a whole. To implement traditional FGAC, the only possibility is to split the partial sensitive element into two sub-elements: one with private privilege and other with public. For example, the credit-card numbers can be split into two sub-elements: one with first 12 digits which is made private and the other with last 4 digits which is made public. However, practically this is not feasible in all cases, as the sensitivity level and the access-privilege of the same element might be different for different users, and the integrity of data is compromised. For instance, when an integer data (say, 10) is partially viewed as an interval (say, [5, 25]), we can not split it.*

To cope with this situation, we propose an Observation-based Fine Grained Access Control (OFGAC) mechanism where data are made accessible at various level of abstraction according to their sensitivity level, based on the Abstract Interpretation framework. In this setting, unauthorized users are not able to infer the exact content of the data cell containing partial sensitive information, while they are allowed to get a relaxed view of it, according to their access rights.

The structure of this chapter is as follows: in section 5.1, we discuss the related work on FGAC framework for relational databases and XML documents in the literature. In sections 5.2 and 5.3, we describe the proposed OFGAC framework to the context of RDBMS and XML documents respectively.

Figure 5.1: A Document Type Definition (DTD) and its instance

(a) DTD

```
<?xml version="1.0"? >
<! DOCTYPE BankCusomers [ >
<! ELEMENT BankCusomers(Customer*) >
<! ELEMENT Customer(PersInfo, AccountInfo+, CreditCardInfo?) >
<! ELEMENT PersInfo(Cid, Name, Address, PhoneNo) >
<! ELEMENT Cid (# PCDATA) >
<! ELEMENT Name (# PCDATA) >
<! ELEMENT Address (street, city, country, pin) >
<! ELEMENT street (# PCDATA) >
<! ELEMENT city (# PCDATA) >
<! ELEMENT country (# PCDATA) >
<! ELEMENT pin (# PCDATA) >
<! ELEMENT PhoneNo (# PCDATA) >
<! ELEMENT AccountInfo (IBAN, type, amount) >
<! ELEMENT IBAN (# PCDATA) >
<! ELEMENT type (# PCDATA) >
<! ELEMENT amount (# PCDATA) >
<! ELEMENT CreditCardInfo (CardNo, ExpiryDate, SecretNo) >
<! ELEMENT CardNo (# PCDATA) >
<! ELEMENT ExpiryDate (# PCDATA) >
<! ELEMENT SecretNo (# PCDATA) >
<! ATTLIST Cid IBAN CDATA #REQUIRED ]>
```

(b) XML document

```
<?xml version="1.0"? >
<BankCusomers>
<Customer>
<PersInfo>
<Cid> 140062 </Cid>
<Name> John Smith </Name>
<Address>
<street> Via Pasini 62 </street>
<city> Venezia </city>
<country> Italy </country>
<pin> 30175 </pin>
</Address>
<PhoneNo> +39 3897745774 </PhoneNo>
</PersInfo>
<AccountInfo>
<IBAN> IT10G 02006 02003 000011115996 </IBAN>
<type> Savings </type>
<amount> 50000 </amount>
</AccountInfo >
<CreditCardInfo>
<CardNo> 4023 4581 8419 7835 </CardNo>
<ExpiryDate> 12/15 </ExpiryDate>
<SecretNo> 165 </SecretNo>
</CreditCardInfo>
</Customer>
</BankCusomers>
```

5.1 Related Works

The existing schemes on FGAC for RDBMS suggest to mask the confidential information by special symbols like NULL (130) or *Type*-1/*Type*-2 variables (192), or to execute the queries over the operational relations (175, 205) or authorized views (112, 167), etc.

Wang et al. (192) proposed a formal notion of correctness in fine-grained database access control. They showed why the existing approaches (130) fall short in some circumstances with respect to soundness and security requirements, like when queries contain negation operations. Moreover, they proposed a labeling approach for masking unauthorized information by using two types of special variables (Type-1 or Type-2) as well as a secure and sound query evaluation algorithm in case of cell-level disclosure policies.

In (175, 205), the authors observed that the proposed algorithm in (192) is unable to satisfy the soundness property for the queries containing the negation operations NOT IN or NOT EXISTS. They proposed an enforcing rule to control the information leakage where the query is executed on an operational relation rather than the original relation. However, although the algorithm for Enforcing Rule satisfies the soundness and security properties for all SQL queries, it would not reach the maximum property (192).

The authors in (23) expressed the secret information by an existentially quantified boolean query. They presented a formal model of secret information disclosure that defines a query to be suspicious if and only if the disclosed secret could be inferred from its answer.

Agrawal et al. (3) introduced the syntax of a fine grained restriction command at column level, row level, or cell level. The enforcement algorithm automatically combines the restrictions relevant to individual queries annotated with purpose and recipient information, and transforms the users' queries into equivalent queries over a dynamic view that implements the restriction.

In (204), the authors extended the SQL language to express the FGAC security policies. They provide syntax to create a new policy type or replace the old policy with a new one. Many policy instances of a policy type can be created when needed. Moreover it specifies the operations on the objects that the security policy restricts and the filter list that specifies the data to be accessed in the specific objects. Finally

it has constraint expressions whose truth value determine whether the policy will be executed or not.

Rizvi et al. in (167) described two models for fine-grained access control: Truman and Non-Truman models. Both models support authorization-transparent querying. Unlike the Truman model, the Non-Truman model avoids the pitfalls of the query modification approach and allows a great deal of flexibility in authorization, such as authorization of aggregate results. They defined the notions of unconditional and conditional validity of the query, and presented several inference rules for validity.

Kabra et al. (112) defined the circumstances when a query plan is safe with respect to user defined functions (UDFs) and other unsafe functions (USFs). They proposed techniques to generate safe query plans. However, these safe query plans may yield to un-optimized plans.

The authors in (118) presented two models to solve the information leakage problem occurring during query aggregation. The first model is the base model that uses a single inference dispersion value (Δ) for each user where as the second model uses multiple inference dispersion values for each user with a view to provide more accessibility. Whenever a user queries the database, the query is passed through the inference interpreter. Based on the data items already sent to the user and the data items currently requested the interpreter determines if there is a possibility of inference. The interpreter rejects the query if it finds that inference is possible; otherwise, the query is processed. The interpreter determines inference mathematically by using a mechanism called aggregation graphs and setting up a threshold called inference dispersion.

In (97), authors presented a quantitative model for privacy protection. In the model, a formal representation of the user's information states is given, and they estimate the value of information for the user by considering a specific user model. Under the user model, the privacy protection task is to ensure that the user cannot profit from obtaining the private information.

Various proposals in support of fine-grained XML access control have been introduced in the literature. These include View-based (14, 53, 54), Non-Deterministic Finite Automata (NFA)-based (24, 138, 151), RDBMS-based (121, 128, 184) XML Access Control Enforcement Techniques, etc.

The idea of view-based access control is to create and maintain separate view for each user based on the authorization rules. During run-time, users' queries are exe-

cuted on the corresponding view, without worrying about the security enforcement. However, for a system with large number of users, the view based approach suffers from high maintenance and storage cost, although views are prepared offline. Yu et al. (199) strictly improved the time and space complexity of view-based approaches by taking advantage of structural locality of accessibility where data items grouped together with similar accessibility properties and a Compressed Accessibility Map (CAM) is built that acts as an accessibility index. In addition to the existing authorization privileges ("allow" or "forbid"), Wu and Raun (196) included one more access privilege, called delegatable administrative privilege, where authorization of an element can be propagated to its parent/child elements if the element has delegatable administrative privilege and there is no authorization for its parent/child elements at all. Moreover, any conflict in the access authorization, if occurs, are resolved by assigning priority to each authorization privileges.

The proposals in (24, 138, 151) are based on rewriting the users queries conforming the access control rules by using Non-Deterministic Finite Automata (NFA). The static analysis in (151) compares the schema automata, access-control automata, and query regular expressions, and classifies queries to be either entirely authorized or entirely prohibited before submitting it to the XML engine. For partially authorized XML queries it relies on the XML engine to filter out the data nodes that users do not have authorization to access, *i.e.*, when static analysis cannot provide determinate answers, the scheme relies on run-time checking. QFilter (24) first generates NFA for the set of access control rules. The input query is then processed against the NFA to determine whether the query satisfies the access control rules completely or partially, and accordingly, it is rewritten into a filtered query by removing the conflicting portions from the input query. The proposal in (138) uses NFA to process streaming XML data for access control.

Although, Relational Database Management System, due to its structured nature, becomes inappropriate in the context of World Wide Web, most data for XML documents still reside in relational databases behind the scene. The proposals in (121, 128, 184) takes advantage of relational model, by mapping all the XML data and access controls rules for XML data (in XML format) into the equivalent relational database and structured query language representation. Finally, the accessibility of

the data are checked at the level of relational database representation using the SQL representation of the rules.

5.2 OFGAC for RDBMS

In this section, we describe the observation-based access control framework for RDBMS, in order to provide various levels of accessibility of database information to the users.

5.2.1 Observation-based Access Control Policy Specification for RDBMS

Before describing the OFGAC framework for RDBMS, let us define observation-based access control policy under OFGAC framework, in contrast to traditional binary-based access control policy, and the abstract database representation under these observation-based access policies.

Definition 10 (Observation-based Disclosure Policy) *Given a domain of observable properties D, and an abstraction function $\alpha_D : \wp(val) \rightarrow D$, an observation-based disclosure policy op assigned to the observer O is a tagging that assigns each value v in the database state σ a tag $\alpha_D(v) \in D$, meaning that O is allowed to access the value $\alpha_D(v)$ instead of its actual value v.*

Unlike traditional FGAC, the database information which are unauthorized under an observation-based disclosure policy *op*, are abstracted by the information at various levels of abstractions representing some properties of interest. In other words, the confidential database information are abstracted by the abstract values, rather than NULL or special symbols (130, 192). The unauthorized users, thus, could not be able to infer the exact content of the sensitive cells. Different levels of sensitivity of the information correspond to different levels of abstractions. Less sensitive values are abstracted by the abstract values at lower level of abstraction, while more sensitive values are abstracted by the abstract values at higher level of abstraction. This way, we can tune different parts of the same database content according to different levels of abstractions at the same time, giving rise to various observational access control for various parts of the database state. The query issued by the external users will be directed to and executed over the abstract databases, yielding to a sound approximation of the query results. The traditional fine grained access control can be seen as a special case of our OFGAC framework.

Example 10 *Consider the database of Table 5.1 where the cells containing sensitive information are marked with 'N' within parenthesis. In OFGAC framework, we abstract these sensitive*

Table 5.1: Concrete Database

(a) "*emp*"

eID	*Name*	*Age*	*Dno*	*Sal*
1	Matteo (N)	30	2	2800 (N)
2	Pallab (N)	22	1	1500
3	Sarbani (N)	56 (N)	2	2300
4	Luca (N)	35	1	6700 (N)
5	Tanushree (N)	40 (N)	3	4900
6	Andrea (N)	52 (N)	1	7000 (N)
7	Alberto (N)	48	3	800
8	Mita (N)	29 (N)	2	4700 (N)

(b) "*dept*"

Dno	*Name*	*Loc*	*Phone*	*DmngrID*
1	Financial	Venice	111-1111 (N)	6
2	Research	Rome	222-2222 (N)	8
3	Admin	Treviso	333-3333 (N)	3

Table 5.2: Partial Abstract Database

(a) "$emp^\sharp$"

$eID^\sharp$	$Name^\sharp$	$Age^\sharp$	$Dno^\sharp$	$Sal^\sharp$
1	Male	30	2	Medium
2	Male	22	1	1500
3	Female	[50, 59]	2	2300
4	Male	35	1	Very high
5	Female	[40, 49]	3	4900
6	Male	[50, 59]	1	Very high
7	Male	48	3	800
8	Female	[20, 29]	2	High

(b) "$dept^\sharp$"

$Dno^\sharp$	$Name^\sharp$	$Loc^\sharp$	$Phone^\sharp$	$DmngrID^\sharp$
1	Financial	Venice	$\top$	6
2	Research	Rome	$\top$	8
3	Admin	Treviso	$\top$	3

information by the abstract values representing specific properties of interests as depicted in Table 5.2. Observe that in $emp^\sharp$, the ages are abstracted by the elements from the domain of intervals, the salaries are abstracted by their relative measures: Low, Medium, High, Very High, and the names are abstracted by their sex properties. It is worthwhile to mention here that we assume employees' salaries to be more sensitive than their ages, and thus, we abstract salary values with a higher level of abstraction than the age values, although both are numeric. Most importantly, in the abstract table $dept^\sharp$, since the phone numbers of all departments are strictly confidential, they are abstracted by the top element $\top$ of their

corresponding abstract lattice. We call the resulting database as "Partial Abstract Database", in contrast to "Full Abstract Database", since only a subset of the database information is abstracted. The correspondence between the concrete values of salaries and the abstract values that partially hide sensitive salary values can formally be expressed by the abstraction and concretization functions α_{sal} and γ_{sal} respectively as follows:

$$\alpha_{sal}(X) \triangleq \begin{cases} \bot \ \textit{if}\ X = \emptyset \\ \textit{Low} \ \textit{if}\ \forall x \in X:\ 500 \leq x \leq 1999 \\ \textit{Medium} \ \textit{if}\ \forall x \in X:\ 2000 \leq x \leq 3999 \\ \textit{High} \ \textit{if}\ \forall x \in X:\ 4000 \leq x \leq 5999 \\ \textit{Very High} \ \textit{if}\ \forall x \in X:\ 6000 \leq x \leq 10000 \top \ \textit{otherwise} \end{cases}$$

$$\gamma_{sal}(d) \triangleq \begin{cases} \emptyset \ \textit{if}\ d = \bot \\ \{x\ :\ 500 \leq x \leq 1999\}\ \textit{if}\ d = \textit{Low} \\ \{x\ :\ 2000 \leq x \leq 3999\}\ \textit{if}\ d = \textit{Medium} \\ \{x\ :\ 4000 \leq x \leq 5999\}\ \textit{if}\ d = \textit{High} \\ \{x\ :\ 6000 \leq x \leq 10000\}\ \textit{if}\ d = \textit{Very High} \\ \{x\ :\ 500 \leq x \leq 10000\}\ \textit{if}\ d = \top \end{cases}$$

Formally, the sensitive values of the data cells belonging to an attribute x are abstracted by using the Galois Connection $(\wp(D_x^{con}), \alpha_x, \gamma_x, D_x^{abs})$, where $\wp(D_x^{con})$ and D_x^{abs} represent the powerset of concrete domain of x and the abstract domain of x respectively, whereas α_x and γ_x represent the corresponding abstraction and concretization functions (denoted $\alpha_x : \wp(D_x^{con}) \rightarrow D_x^{abs}$ and $\gamma_x : D_x^{abs} \rightarrow \wp(D_x^{con})$) respectively. In case of insensitive information, the abstraction and concretization functions represent the identity function *id*.

Given a concrete database state σ_{op} under an observation-based disclosure policy *op*, the abstract state is obtained by performing $\sigma_{op}^{\sharp} = \alpha(\sigma_{op})$ where the abstraction function α can be expressed as collection of abstraction functions for all attributes in the database.

We assume that for each type of values in a database there exists a hierarchy of abstractions such that Galois Connections combine consistently, *i.e.* if $(X, \alpha_1, \gamma_1, Y)$ and $(Y, \alpha_2, \gamma_2, Z)$ represent two Galois Connection, then we have the following:

$$\textit{if}\ (X, \alpha_1, \gamma_1, Y)\ \textit{and}\ (Y, \alpha_2, \gamma_2, Z)\ \textit{then}\ (X, \alpha_2 \circ \alpha_1, \gamma_1 \circ \gamma_2, Z)$$

Observe that the traditional FGAC (130, 192) is a special case of our OFGAC framework where each unauthorized cell is abstracted by the top element $\top$ of its corresponding abstract lattice.

5.2.2 Preserving Referential Integrity Constraints under OFGAC

We know that the uniqueness of the values in primary or foreign key attributes is a crucial requirement in order to maintain the secure linking or referential integrity among database relations. When sensitive cells in primary or foreign key attributes are abstracted in OFGAC framework, the uniqueness criterion can not be maintained due to loss of precision, and the referential integrity constraints among database relations put under threat. In (192), Wang et al. used Type-2 variable in order to keep these referential integrity constraints intact while masking operation is performed. The attribute values which are sensitive and act as primary key or foreign key are masked by the Type-2 variables.

We extend the same approach of Wang et al. (192) in our OFGAC framework. We denote the Type-2 variable by a pair (v, A) where A is an abstract value and $\gamma(A)$ is the domain of the variable v, depicted in definition 11.

Definition 11 (Type-2 Variable) *A Type-2 variable is represented by a pair (v, A) where A is an abstract value and $\gamma(A)$ is the domain of the variable v. Given v_1, v_2, A_1, A_2 where v_1 is assumed to be different from v_2, then we have that (i) "$(v_1, A_1) = (v_1, A_1)$" and "$(v_1, A_1) \neq (v_2, A_1)$" are always true, (ii) "$(v_1, A_1) \neq (v_2, A_2)$" is true if $\gamma(A_1) \cap \gamma(A_2) = \emptyset$, (iii) "$(v_1, A_1) = (v_2, A_2)$" is $\top$ if $\gamma(A_1) \cap \gamma(A_2) \neq \emptyset$, and (iv) "$(v_1, A_1)$=$c$" is $\top$ where $c \in \gamma(A_1)$.*

Given two sensitive values x_1 and x_2 under the same primary/foreign key attribute. According to Definition 11, we abstract them as follows: (*i*) if $x_1 = x_2$ and $\alpha(x_1) = \alpha(x_2) = A$, then both x_1 and x_2 are abstracted by the Type-2 variable (v, A), (*ii*) if $x_1 \neq x_2$ and $\alpha(x_1) = \alpha(x_2) = A$, then x_1 and x_2 are abstracted by (v_1, A) and (v_2, A) respectively, (*iii*) if $x_1 \neq x_2$, $\alpha(x_1) = A_1$, $\alpha(x_2) = A_2$ and $\gamma(A_1) \cap \gamma(A_2) = \emptyset$, then x_1 and x_2 are abstracted by (v_1, A_1) and (v_2, A_2) respectively.

Example 11 *Consider the supplier-parts database and its abstract version under an observation-based access control policy, depicted in Table 5.3. The attributes S-id and P-id are the primary keys of the tables "Supplier" and "Part" respectively, whereas the composite attribute {S-id, P-id} is used as the primary key of the table "Supp-Part". Observe that S-id and P-id in*

"Supp-Part" are used as the foreign keys that link to the primary keys of "Supplier" and "Part" respectively, and relate the suppliers with the parts sold by them. Suppose according to the disclosure policy that all values of the attributes S-id, P-id and some values of QTY in "Supp-Part" are confidential (marked with 'N' in the parenthesis). If we abstract these values by the abstract values from the domain of intervals, we may loose the ability to identify the tuples uniquely and the secure linking between "Supplier" and "Part" might be disturbed. To preserve the uniqueness of the values in abstract domain, we use Type-2 variable, as depicted in the abstract tables "Supplier$^\sharp$", "Part$^\sharp$" and "Supp-Part$^\sharp$". Observe that since the attribute QTY is not primary key or foreign key, we abstract them only by the abstract values from the domain of intervals.

Table 5.3: Preserving Referential Integrity Constraint by using Type-2 variable

(a) "*Supplier*"

S-id	*Name*	*Age*
S230 (N)	Alice	24
S201 (N)	Bob	21
S368 (N)	Tea	22

(b) "*Supp − Part*"

S-id	*P-id*	*QTY*
S230 (N)	P140 (N)	120
S201 (N)	P329 (N)	260 (N)
S230 (N)	P563 (N)	200
S368 (N)	P329 (N)	450 (N)
S368 (N)	P140 (N)	430 (N)

(c) "*Part*"

P-id	*Pname*
P140 (N)	Screw
P329 (N)	Bolt
P563 (N)	Nut

(d) "*Supplier*$^\sharp$"

S-id$^\sharp$	*Name*$^\sharp$	*Age*$^\sharp$
$(v_1, [S200, S249])$	Alice	24
$(v_2, [S200, S249])$	Bob	21
$(v_3, [S350, S399])$	Tea	22

(e) "*Supp − Part*$^\sharp$"

S-id$^\sharp$	*P-id*$^\sharp$	*QTY*$^\sharp$
$(v_1, [S200, S249])$	$(v_4, [P100, P149])$	120
$(v_2, [S200, S249])$	$(v_5, [P300, P349])$	[250,299]
$(v_1, [S200, S249])$	$(v_6, [P550, P599])$	200
$(v_3, [S350, S399])$	$(v_5, [P300, P349])$	[450, 499]
$(v_3, [S350, S399])$	$(v_4, [P100, P149])$	[400, 449]

(f) "*Part*$^\sharp$"

P-id$^\sharp$	*Pname*$^\sharp$
$(v_4, [P100, P149])$	Screw
$(v_5, [P300, P349])$	Bolt
$(v_6, [P550, P599])$	Nut

5.2.3 Query Evaluation under OFGAC

A general framework for Abstract Interpretation of Relational Databases has been introduced in the chapter 3. Here, we briefly recall some notions on query abstraction, and we extend them by considering queries on multiple abstractions as well.

Example 12 *Consider the concrete database of Table 5.1 and the corresponding partial abstract database depicted in Table 5.2 under an observational disclosure policy op. Suppose an external user issues a query Q_1 under op as below:*

$$Q_1 = \texttt{SELECT * FROM } emp \texttt{ WHERE } Sal > 4800;$$

The system transforms Q_1 into the corresponding abstract version of the query (denoted $Q_1^\sharp$ as shown below:

$$Q_1^\sharp = \texttt{SELECT * FROM } emp^\sharp \texttt{ WHERE } Sal^\sharp > 4800 \texttt{ OR } Sal^\sharp >^\sharp High;$$

The result of $Q_1^\sharp$ on $emp^\sharp$ is depicted in Table 5.4. Observe that the pre-condition $\phi^\sharp$ (represented

Table 5.4: $\xi_1^\sharp$: Result of $Q_1^\sharp$

$eID^\sharp$	$Name^\sharp$	$Age^\sharp$	$Dno^\sharp$	$Sal^\sharp$
4	Male	35	1	Very high
5	Female	[40, 49]	3	4900
6	Male	[50, 59]	1	Very high
8	Female	[20, 29]	2	High

by WHERE clause in $Q_1^\sharp$) evaluates to true for the first three tuples in the result $\xi_1^\sharp$, whereas it evaluates to $\top$ (may be true or may be false) for the last tuple. The result of $Q_1^\sharp$ is sound as it over-approximate the result of the query Q_1. Observe in fact that $Q_1^\sharp$ includes also the "false positive" corresponding to the concrete information about Mita.

Query Evaluation in presence of Aggregate Functions and Set Operations under OFGAC

In chapter 3, we defined the abstract aggregate functions and abstract set operations in an abstract domain of interest. In OFGAC framework, we apply them in the same way as depicted in Example 13 and 14 respectively.

Example 13 *Consider the abstract database of Table 5.2 and an abstract query $Q_2^\sharp$ containing aggregate functions.*

$$Q_2^\sharp = \texttt{SELECT COUNT}^\sharp(*), \texttt{ AVG}^\sharp(Age^\sharp) \texttt{ FROM } emp^\sharp$$
$$\texttt{WHERE } (Age^\sharp \texttt{ BETWEEN 32 AND 55}) \texttt{ OR } (Age^\sharp \texttt{ BETWEEN}^\sharp \texttt{ [30, 39] AND [50, 59]});$$

The result of $Q_2^\sharp$ on $emp^\sharp$ is depicted in Table 5.5. In the example, the evaluation of the abstract WHERE clause extracts five tuples in total where three tuples with $eID^\sharp$ equal to 4, 5, 7 belong to $G_{yes}^\sharp$ and two tuples with $eID^\sharp$ equal to 3, 6 belong to $G_{may}^\sharp$. Thus, in case of $\texttt{AVG}^\sharp(Age^\sharp)$, we get $a^\sharp = fn^\sharp(\{35, [40, 49], 48\}) = average^\sharp(\{35, [40, 49], 48\})$=[41, 44] and $b^\sharp = fn^\sharp(\{[50,$

Table 5.5: $\xi_2^\sharp$: Result of $Q_2^\sharp$

$\texttt{COUNT}^\sharp(*)$	$\texttt{AVG}^\sharp(Age^\sharp)$
[3, 5]	[41, 50]

59], 35, [40, 49], [50, 59], 48}) = average$^\sharp$({[50, 59], 35, [40, 49], [50, 59], 48}) = [44, 50]. Hence, $\texttt{AVG}^\sharp(Age^\sharp)$ = [min$^\sharp(a^\sharp)$, max$^\sharp(b^\sharp)$]=[41, 50]. Similarly, in case of $\texttt{COUNT}^\sharp(*)$, we get $a^\sharp$ = count$^\sharp(G^\sharp_{yes})$=[3, 3] and $b^\sharp$ = count$^\sharp(G^\sharp)$ = [5, 5]. Thus, $\texttt{COUNT}^\sharp(*)$ =[3, 5]. Observe that the result is sound, i.e., $\xi_2 \in \gamma(\xi_2^\sharp)$ where ξ_2 is the result of a concrete query $Q_2 \in \gamma(Q_2^\sharp)$.

Example 14 *Consider the abstract database of Table 5.2 and an abstract query $Q_3^\sharp = Q_l^\sharp$ $\texttt{MINUS}^\sharp$ $Q_r^\sharp$, where*

$Q_l^\sharp$ = *SELECT * FROM emp$^\sharp$ WHERE Sal$^\sharp$ > 2500 OR Sal$^\sharp$ >$^\sharp$ Medium; and*

$Q_r^\sharp$ = *SELECT * FROM emp$^\sharp$ WHERE Sal$^\sharp$ > 5500 OR Sal$^\sharp$ >$^\sharp$ High;*

The execution of $Q_l^\sharp$ and $Q_r^\sharp$ on emp$^\sharp$ are depicted in Table 5.6(a) and 5.6(b) respectively. In

Table 5.6: Abstract Computation of $Q_3^\sharp$

(a) $\xi_l^\sharp$: Result of $Q_l^\sharp$

$eID^\sharp$	$Name^\sharp$	$Age^\sharp$	$Dno^\sharp$	$Sal^\sharp$
1	Male	30	2	Medium
4	Male	35	1	Very high
5	Female	[40, 49]	3	4900
6	Male	[50, 59]	1	Very high
8	Female	[20, 29]	2	High

(b) $\xi_r^\sharp$: Result of $Q_r^\sharp$

$eID^\sharp$	$Name^\sharp$	$Age^\sharp$	$Dno^\sharp$	$Sal^\sharp$
4	Male	35	1	Very high
6	Male	[50, 59]	1	Very high
8	Female	[20, 29]	2	High

(c) $\xi_3^\sharp$: Performing $\texttt{MINUS}^\sharp$ between $\xi_l^\sharp$ & $\xi_r^\sharp$

$eID^\sharp$	$Name^\sharp$	$Age^\sharp$	$Dno^\sharp$	$Sal^\sharp$
1	Male	30	2	Medium
5	Female	[40, 49]	3	4900
8	Female	[20, 29]	2	High

Table 5.6(a), for the first tuple the pre-condition of $Q_l^\sharp$ evaluates to $\top$ (thus, belongs to $\xi^\sharp_{may_l}$), whereas for the remaining four tuples it evaluates to true (thus, belongs to $\xi^\sharp_{yes_l}$). Similarly, in Table 5.6(b), for the first two tuples the pre-condition of $Q_r^\sharp$ evaluates to true (hence, belongs to $\xi^\sharp_{yes_r}$), whereas for the last one it evaluates to $\top$ (hence, belongs to $\xi^\sharp_{may_r}$). Thus, the first component $(\xi^\sharp_{yes_l} - (\xi^\sharp_{yes_r} \cup \xi^\sharp_{may_r})) \in \xi_3^\sharp$ contains the tuple with $eID^\sharp$ equal to 5, and the second component $((\xi^\sharp_{may_l} \cup \xi^\sharp_{may_r}) - \xi^\sharp_{yes_r}) \in \xi_3^\sharp$ contains the tuples with $eID^\sharp$ equal to 1 and 8, as shown in Table 5.6(c). The result is sound, i.e., $\xi_3 \in \gamma(\xi_3^\sharp)$ where ξ_3 is the result of a concrete query $Q_3 \in \gamma(Q_3^\sharp)$.

5.2.4 Collusion Attacks

Wang et al. in (192) illustrate the security of the FGAC system in case of collusion and multi-query attacks. They define the security aspect in the context of *one-party single-query/weak security* and *multi-party multi-query/strong security*, and prove that the system with weak-security is also secure under strong-security.

In OFGAC, transforming the system into an abstract domain means transforming the attackers, and the attackers are modeled by abstractions. The robustness of the database under OFGAC policies depends on the ability of the external observers to distinguish the database states based on the observable properties of the query results.

Here we consider three different scenarios: Figure 5.2(a), 5.2(b) and 5.2(c) illustrates these three cases where the shaded portions indicate the sensitive information and α ($\alpha_i \neq \alpha_j$ if $i \neq j$) is the abstraction function used to abstract those sensitive information.

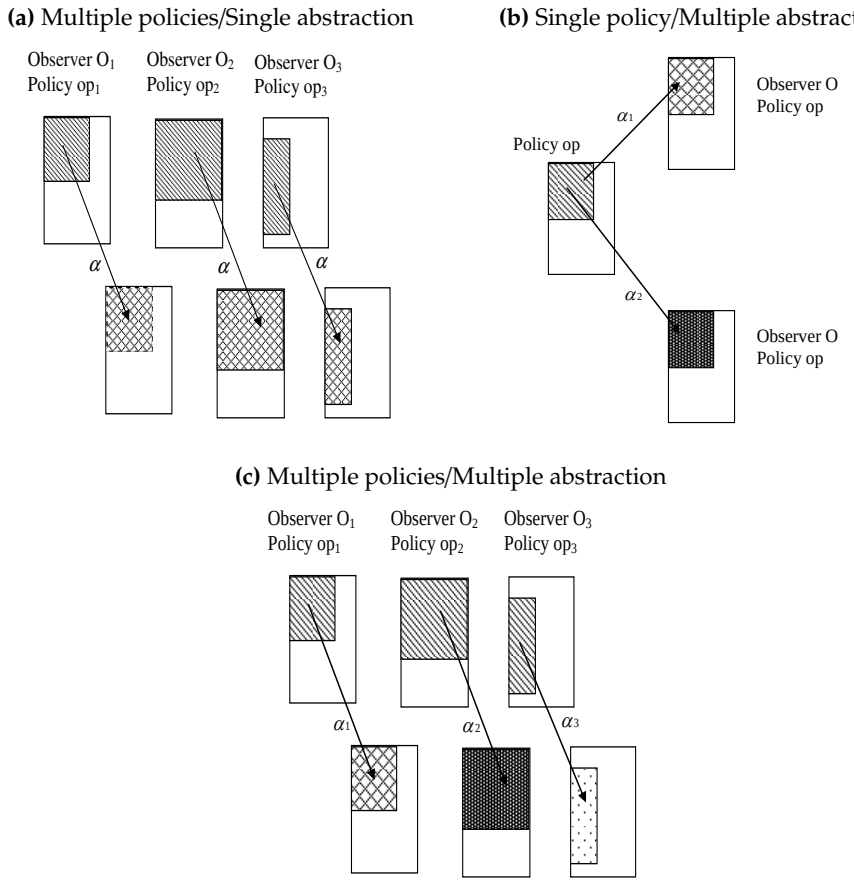
Case 1: Multiple Policies/Single Level Abstraction: Suppose each of the n observers under $op_1, op_2, \ldots, op_n$ respectively issues a query Q. Let σ be a database state without any policy. Under op_i, $i = 1, \ldots, n$, the database state σ is represented by σ_{op_i} and is abstracted into $\sigma^{\sharp}_{op_i} = \alpha(\sigma_{op_i})$. Therefore, observer O_i under op_i will get the query result $\xi^{\sharp}_i = S^{\sharp}[\![Q^{\sharp}]\!](\sigma^{\sharp}_{op_i})$, where $Q^{\sharp}$ is the abstract version of Q. When these n users collude, they feed the query results $\xi^{\sharp}_i$, $i = \ldots, n$, to a function f which can perform some comparison or computation (*viz*, difference operation) among the results and infer about some sensitive information for some observers.

For instance, suppose a portion of database information is sensitive under policy op_j, while it is insensitive under another policy op_k, $j \neq k$. In the former case, this part of information will be abstracted, while in the latter case it will not. Thus, if this portion of information appears in both of the query results $\xi^{\sharp}_j$ and $\xi^{\sharp}_k$, then it is possible for the j^{th} observer to infer the exact content of that portion of information as it is not abstracted in ξ_k.

Let $\sigma_{op} = \langle \sigma_l, \sigma_h \rangle$ and $\sigma_{op'} = \langle \sigma'_l, \sigma'_h \rangle$ be the database states under two different policies op and op'. The database state $\sigma_{op \bullet op'}$ obtained by combining two policies op and op' are defined as follows:

$$\sigma_{op \bullet op'} = \langle (((\sigma_l \cup \sigma_h) - (\sigma_h \cap \sigma'_h)), (\sigma_h \cap \sigma'_h) \rangle$$

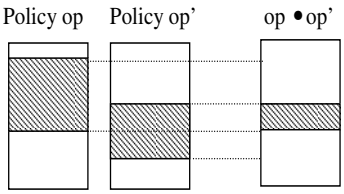
Figure 5.2: Policies and Observations



This fact is depicted in Figure 5.3. So, when the observers under op and op' collude and share the query results, both will act as equivalent to the observer under the policy $op \bullet op'$ and thus they can infer the values belonging to the public part of $op \bullet op'$, *i.e.*, $((\sigma_l \cup \sigma_h) - (\sigma_h \cap \sigma'_h))$ by issuing a sequence of queries individually and by comparing the results together.

Case 2: Single Policy/Multiple Level Abstraction: Consider n different observers O_1, O_2, ..., O_n under the same policy op and the sensitive information part is abstracted to different level of abstraction to different observers. Higher levels of abstraction make the database information less precise, whereas lower levels of abstraction represent them with more precision. Thus, the result of a query for the one with higher abstraction contains less precise information than that with lower abstraction.

Figure 5.3: Combination of policies



Consider two different observers O_1 and O_2 under *op* where the sensitive database information of σ_{op} are abstracted by the domains of abstraction D_1^{abs} and D_2^{abs}, yielding to $\sigma_{op}^{1\sharp}$ and $\sigma_{op}^{2\sharp}$ respectively.

First consider the case where D_2^{abs} is a more abstract domain than D_1^{abs}, *i.e.*, D_2^{abs} is an abstraction of D_1^{abs}. Since both observers are under the same policy, the query results over $\sigma_{op}^{1\sharp}$ and $\sigma_{op}^{2\sharp}$ may contain some common abstract information - one from D_1^{abs} and other from D_2^{abs}. Thus when O_1 and O_2 collude, it is possible for O_2 to obtain sensitive information with lower level of abstraction from the result obtained by O_1 as it is abstracted with lower level of abstraction for O_1. But no real collusion may arise in this case, as the overall information available to O_1 and O_2 together is at most as precise as the one already available to O_1.

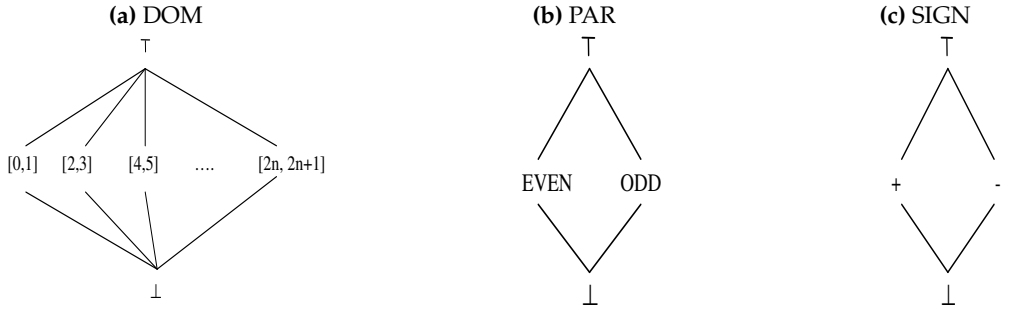
The other case is where the two domains are not one the abstraction of the other. For example, let in a particular database state an attribute of a table has the sensitive values represented by an ordered list $\langle 5, 0, 2, 3, 1\rangle$. Suppose the observer O_1 is limited by the property DOM represented by domain of intervals as shown in Figure 5.4(a), while O_2 is limited by parity property represented by the abstract domain PAR=$\{\bot$, EVEN, ODD, $\top\}$ as depicted in Figure 5.4(b). Thus O_1 sees $\langle$[4, 5], [0, 1], [2, 3], [2, 3], [0, 1]$\rangle$, while O_2 sees $\langle$ODD, EVEN, EVEN, ODD, ODD$\rangle$. When O_1 and O_2 collude, they can infer the exact values for the attribute, *i.e.*, $\langle 5, 0, 2, 3, 1\rangle$ by combining the query results. The corresponding combined lattice obtained by reduced product (47) of the above two abstract lattices DOM and PAR is shown in Figure 5.5(a).

Given an OFGAC under *Single policy/Multiple level abstraction* scenario where same information under the same policy is abstracted by n different level of abstraction to n different observers. Such OFGAC is *collusion-prone* when intersection of the sets (not singletons) obtained by concretizing abstract values

of any common sensitive cell appearing in different query results for different observers, yield to a singleton. This is depicted in Definition 12.

We now show an example where no collusion takes place in practice. Consider two observers O_1 and O_2, where O_1 is limited by the sign property depicted in Figure 5.4(c), whereas O_2 is limited by the parity property depicted in Figure 5.4(b). Let ⟨-2, 0, 2, -1, 1⟩ be a list of sensitive values appearing in the results of the queries issued by both O_1 and O_2. Thus, O_1 sees ⟨-, +, +, -, +⟩, while O_2 sees ⟨EVEN, EVEN, EVEN, ODD, ODD⟩. When O_1 and O_2 collude, they can infer the values as ⟨EVEN$^-$, EVEN$^+$, EVEN$^+$, ODD$^-$, ODD$^+$⟩ by combining the query results. However, although these combined abstract values represent more precise information than that of the individual result, the observer still could not be able to infer the exact content. Figure 5.5(b) shows the combined abstract lattice obtained by reduced product (47) of two abstract lattices SIGN and PAR.

Figure 5.4: Abstract Lattices of DOM, PAR and SIGN



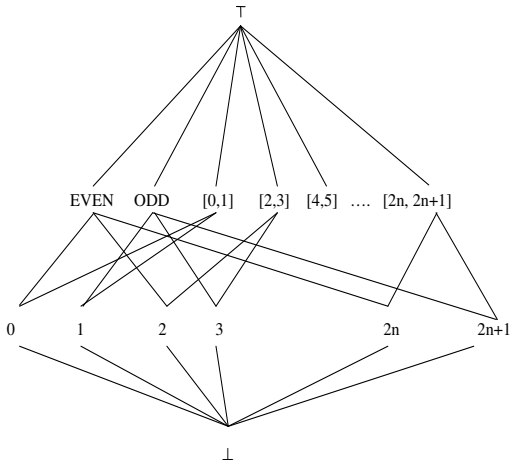
Definition 12 *An OFGAC under Single policy/Multiple level abstraction scenario is collusion-prone, if the OFGAC uses n different abstract domains $D_1^{abs}, \ldots, D_n^{abs}$ for n different observers and $\exists\{d_i, \ldots, d_j\} \in D_i^{abs} \times \cdots \times D_j^{abs}$ for $\{i, \ldots, j\} \subseteq \{1, \ldots, n\}$ such that $\bigcap_{k\in\{i,\ldots,j\}} \gamma(d_k) = \{e\}$ while $\forall k \in \{i, \ldots, j\}$, $\gamma(d_k) \neq \{e\}$.*

Theorem 1 *An OFGAC using n different abstract domains $D_1^{abs}, \ldots, D_n^{abs}$ for n different observers under the same policy is collusion-prone if the reduced product (47) of $\{D_i^{abs}, \ldots, D_j^{abs}\} \subseteq \{D_1^{abs}, \ldots, D_n^{abs}\}$ is isomorphic to a concrete domain D.*

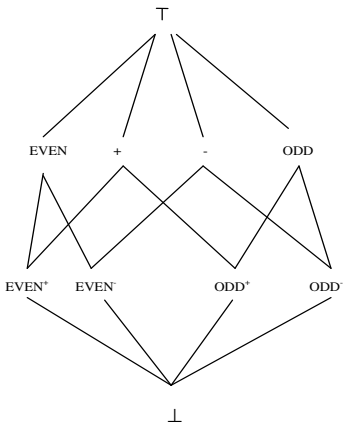
Case 3: Multiple Policies /Multiple Level Abstractions: This is the combination of the previous two cases. Observers may collude to act as the observer under the

Figure 5.5: Combination of lattices

(a) Combined lattice of DOM and PAR



(b) Combined lattice of SIGN and PAR



combination of their individual policies, or may try to infer about the confidential information appearing in the query results by combining (*e.g.*, intersecting) their domain of abstract values.

5.3 OFGAC for XML

We now extend the notion of OFGAC to the context of XML documents. We first introduce the notion of access control policy specification for XML under OFGAC framework. Then, we apply the OFGAC approach in two directions: view-based and RDBMS-based.

5.3.1 Observation-based Access Control Policy Specification for XML

Most of the existing approaches on fine grained access control for XML are based on the basic policy specification introduced by Damiani et al. (54) that specifies the access authorization by a 5-tuple of the form $\langle Subject, Object, Action, Sign, Type \rangle$. The "*Subject*" represents the identifiers or locations of the access requests to be granted or rejected. It is denoted by a 3-tuple $\langle UG, IP, SN \rangle$ where UG, IP and SN are the set of user-groups/user-identifiers, the set of completely-specified/patterns-of IP addresses and the set of completely-specified/patterns-of symbolic names respectively. For instance, ⟨ Physicians, 159.101.*.*, *.hospital.com ⟩ represents a subject belonging to the group

physicians, issuing queries from the IP address matching with the pattern 159.101.*.* in the domain matching with symbolic name pattern *.hospital.com. The "*Object*" represents the Uniform Resource Identifier (URI) of the elements or attributes in the documents. The URI is specified by the conditional or unconditional path expressions. The "*Action*" is either "read" or "write" or both being authorized or forbidden. The "*Sign*" $\in \{+, -\}$ is the sign of authorization. Sign "+" indicates "allow access", whereas sign "-" indicates "forbid access". The "*Type*" of the access represents the level of access (DTD level or instance level), whether access is applicable only to the local element or applicable recursively to all sub-elements, hard or soft etc. The priority of the type of accesses from highest to lowest are: LDH (Local Hard Authorization), RDH (Recursive Hard Authorization), L (Local Authorization), R (Recursive Authorization), LD (Local Authorization specified at DTD level), RD (Recursive Authorization specified at DTD level), LS (Local Soft Authorization), RS (Recursive Soft Authorization). Since this specification provides users only two choices in accessing the information: either "allow" or "forbid", we call it *Binary-based FGAC Policy* for XML.

In contrast to binary-based FGAC Policy, we specify the Observation-based Access Control Policy for XML under OFGAC framework by a 5-tuple of the form $\langle Subject, Object, Action, Abstraction, Type\rangle$. The components "*Subject*", "*Object*", "*Action*" and "*Type*" are defined exactly in the same way as in case of FGAC policy specification. The component "*Abstraction*" is defined by the Galois Connection $(\wp(D_x^{con}), \alpha_x, \gamma_x, D_x^{abs})$, where $\wp(D_x^{con})$ and D_x^{abs} represent the powerset of concrete domain of x and the abstract domain of x respectively, and α_x and γ_x represent the corresponding abstraction and concretization functions.

Since the "*Object*" represents either XML element or attribute, the following two cases may arise when "*Abstraction*" is applied on them:

- The "*Object*" represents an intermediate element and "*Type*" is "Recursive" (denoted by "R"). In this case, the abstraction defined in the rule for an element is propagated downwards and applied to all its sub-elements and attributes recursively.

- The "*Object*" represents an attribute and "*Type*" is "Local" (denoted by "L"). In this case, only the attribute value is abstracted by following the Galois Connection specified in the rule.

Example 15 illustrates the observation-based access control policy specification for the XML document of Figure 5.1.

Table 5.7: Observation-based Access Control Policy Specification for XML code

Rule	*Subject*	*Object*	*Action*	*Abstraction*	*Type*
R1	customer-care, 159.56.*.*, *.Unicredit.it	/BankCustomers/ Customer/ PersInfo	read	$(\wp(D_x^{con}), id, id, \wp(D_x^{con}))$	R
R2	customer-care, 159.56.*.*, *.Unicredit.it	/BankCustomers/ Customer/ AccountInfo/ IBAN	read	$(\wp(D_{iban}^{con}), \alpha_{iban}, \gamma_{iban}, D_{iban}^{abs})$	L
R3	customer-care, 159.56.*.*, *.Unicredit.it	/BankCustomers/ Customer/ AccountInfo/ type	read	$(\wp(D_{type}^{con}), id, id, \wp(D_{type}^{con}))$	L
R4	customer-care, 159.56.*.*, *.Unicredit.it	/BankCustomers/ Customer/ AccountInfo/ amount	read	$(\wp(D_{amount}^{con}), \alpha_\top, \gamma_\top, \{\top\})$	L
R5	customer-care, 159.56.*.*, *.Unicredit.it	/BankCustomers/ Customer/ CreditCardInfo/ CardNo	read	$(\wp(D_{CardNo}^{con}), \alpha_{CardNo}, \gamma_{CardNo}, D_{CardNo}^{abs})$	L
R6	customer-care, 159.56.*.*, *.Unicredit.it	/BankCustomers/ Customer/ CreditCardInfo/ ExpiryDate	read	$(\wp(D_{ExDate}^{con}), \alpha_\top, \gamma_\top, \{\top\})$	L
R7	customer-care, 159.56.*.*, *.Unicredit.it	/BankCustomers/ Customer/ CreditCardInfo/ SecretNo	read	$(\wp(D_{SecNo}^{con}), \alpha_\top, \gamma_\top, \{\top\})$	L

Example 15 *Consider the XML code in Figure 5.1. The observation-based access control policy under OFGAC framework can be specified as shown in Table 5.7, where the abstraction functions are defined as follows:*

$$\alpha_{CardNo}(\{d_i \ : \ i \in [1 \ldots 16]\}) = \texttt{**** **** **** **}\ d_{13}d_{14}d_{15}d_{16}$$

$$\alpha_\top(X) = \top$$

where X is a set of concrete values and $\top$ is the top most element of the corresponding abstract lattice. The functions α_{iban}, γ_{iban}, γ_{CardNo}, $\gamma_\top$ are also defined in this way depending on the corresponding domains. Observe that the identity function id is used to provide the public accessibility of non-sensitive information, whereas the functions $\alpha_\top$ and $\gamma_\top$ are used to provide private accessibility of highly sensitive information by abstracting them with top most element $\top$ of the corresponding abstract lattice.

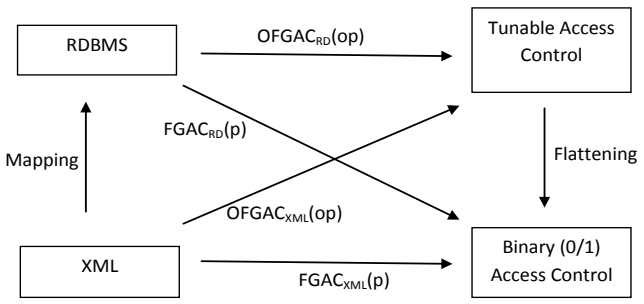
5.3.2 OFGAC Approaches

Given a binary-based access control policy p or an observation-based access control policy op for XML documents, the FGAC and OFGAC can be implemented in two ways:

- Non-deterministic Finite Automata (NFA)-based: By applying p or op directly to the XML documents (view-based) or by rewriting users' XML queries by pruning the unauthorized part in it.
- RDBMS-based: By taking the support of RDBMS, where the XML documents and the XML policies (p or op) are first mapped into the underlying relational databases and the policy SQL respectively, and then the users' XML queries are mapped into equivalent SQL queries and evaluated on those relational databases by satisfying the policy SQL.

Figure 5.6 depicts a pictorial representation of these approaches. Observe that the application of FGAC *w.r.t.* p results into a binary-based access control system that yields two extreme views to the information: either "allow" or "forbid", whereas the application of OFGAC *w.r.t.* op, on the other hand, results into a tunable access control system where partial view of the information at various levels of abstractions is provided. We now discuss the OFGAC approach for XML documents in two directions:

Figure 5.6: Pictorial Representation of FGAC Vs. OFGAC



view-based and RDBMS-based.

View-based OFGAC for XML. Consider the XML code in Figure 5.1 and the associated observation-based access control policy specification depicted in Table 5.7. We

know that in view-based approaches for each subject interacting with the system, separate views are generated with respect to the access rules associated with the subject (54). Therefore, in our example, the XML view corresponding to the users belonging to "customer-care" section of the bank is depicted in Figure 5.7. The queries issued by a user are then executed on the corresponding secure view without worrying about the security constraint. Consider the following XML query Q_{xml} issued by a personnel

Figure 5.7: View generated for the employees in bank's customer-care section

```
<?xml version="1.0"? >
<BankCusomers>
<Customer>
<PersInfo>
<Cid> 140062 </Cid>
<Name> John Smith </Name>
<Address>
<street> Via Pasini 62 </street>
<city> Venezia </city>
<country> Italy </country>
<pin> 30175 </pin>
</Address>
<PhoneNo> +39 3897745774 </PhoneNo>
</PersInfo>
<AccountInfo>
<IBAN> IT*** ***** ***** ********** </IBAN>
<type> Savings </type>
<amount> ⊤ </amount>
</AccountInfo >
<CreditCardInfo>
<CardNo> **** **** **** 7835 </CardNo>
<ExpiryDate> ⊤ </ExpiryDate>
<SecretNo> ⊤ </SecretNo>
</CreditCardInfo>
</Customer>
</BankCusomers>
```

in the customer-care section:

$$Q_{xml} = /BankCusomers/Customer/AccountInfo[@type = \text{"}Savings\text{"}]$$

The execution of Q_{xml} on the view of Figure 5.7 returns the following results:

```
<AccountInfo>
<IBAN> IT*** ***** ***** ********** </IBAN>
<type> Savings </type>
<amount> ⊤ </amount>
</AccountInfo>
```

RDBMS-based OFGAC for XML. Consider the XML document in Figure 5.1 and the observation-based policy specification in Table 5.7. By following (121), we first map the XML document into relational database representation, partially shown in Table 5.8. Observe that we do not translate the XML policies into the equivalent SQL statements, rather we put the rules into the relational database itself by associating them with the corresponding elements or attributes. The empty rule in a row specifies that the corresponding element (and its sub-elements and child-attributes) or attribute has public authorization. If any access-conflict occurs for any sub-element, it is resolved simply by adopting *abstraction-take-precedence* policy according to which authorization

corresponding to more abstract view overrides the authorization corresponding to less abstract view. The users' XML queries are then mapped into SQL representation and are evaluated on this relational database under OFGAC framework as described before. Suppose the following XML query Q_{xml} is issued by an employee from customer-care

Table 5.8: The equivalent relational database representation of the XML code

(a) "*BankCustomers*"

id	*pid*	*rule*
BC1	null	-

(b) "*Customer*"

id	*pid*	*rule*
C1	BC1	-

(c) "*PersInfo*"

id	*pid*	*rule*
PI1	C1	R1

(d) "*AccountInfo*"

id	*pid*	*rule*
AI1	C1	-

(e) "*CreditCardInfo*"

id	*pid*	*rule*
CI1	C1	-

(f) "*IBAN*"

id	*pid*	*val*	*rule*
IB1	AI1	IT10G 02006 02003 000011115996	R2

(g) "*type*"

id	*pid*	*val*	*rule*
TP1	AI1	Savings	R3

(h) "*amount*"

id	*pid*	*val*	*rule*
AM1	AI1	5000	R4

(i) "*CardNo*"

id	*pid*	*val*	*rule*
CN1	CI1	4023 4581 8419 7835	R5

(j) "*ExpiryDate*"

id	*pid*	*val*	*rule*
EX1	CI1	12/15	R6

section of the bank:

$$Q_{xml} = /BankCusomers/Customer/AccountInfo[@type = \text{“}Savings\text{''}]/IBAN$$

Since the OFGAC Policies and XML documents are now in the form of relational database, the system translates Q_{xml} into an equivalent SQL query Q_{rdb} as follows:

Q_{rdb} =`SELECT` *Ch_No.val* `FROM` *IBAN Ch_No, type Ch_Tp, AccountInfo P_AccInfo, Customer P_Cust, BankCustomers P_BCust* `WHERE` (*Ch_No.pid = P_AccInfo.id* `AND` *Ch_Tp.pid = P_AccInfo.id* `AND` *Ch_Tp.val* = "*Savings*") `AND` *P_AccInfo.pid = P_Cust.id* `AND` *P_Cust.pid = P_BCust.id*

The execution of Q_{rdb} on the database of Table 5.8, by following the OFGAC technique for RDBMS, yields the following result:

val
IT*** ***** ***** **********

Observe that RDBMS-based approaches suffer from time-inefficiency, whereas view-based approaches, on the other hand, suffer from space-inefficiency. The possibility of collusion attacks for XML documents under OFGAC framework is same as described before in case of RDBMS.

Chapter 6

SQL Injection Attacks

[Part of this chapter is already published in (80)]

The increasing popularity of web-based services, such as online stores, e-commerce, social network services etc, make them an ideal target for different attacks. One of the most serious type of attacks against web applications is the family of the so called SQL injection attacks (SQLIA). In an SQL injection attack, data provided by the user during run-time are included in an SQL statements in such a way that part of the user's inputs is treated as SQL code. The web applications which receive input from users and incorporate it into SQL statements to an underlying database possibly suffers from SQLIA.

For example, suppose a database contains user names and passwords, and the underlying application contains the following code:

```
query ="SELECT * FROM emp WHERE username='" +
       request.getParameter("username") + "' AND password='" +
       request.getParameter("password") + "';"
```

This query is used to authenticate the user who tries to login to the web site, by checking username and password against data stored in the database. However, if a malicious user enters the input: "' OR '1'='1'--" into the username field, the query

string submitted to the database would be:

```
query =SELECT * FROM emp WHERE
        username=‘ ’ OR ‘1’=‘1’--’ AND password=‘ ’;
```

Since the atomic formula `‘1’=‘1’` is a tautology, the user will bypass the check, and authentication will be successful.

In (93), the authors classified the SQL injection attacks into different types: (*i*) Injection through user input; (*ii*) Injection through cookies; (*iii*) Injection through server variables; (*iv*) Second-order injection. The characterization of attacks is based on goals or intents of the attacker. The different types of attack intents are: Identifying injectable parameters, Performing database finger-printing, Determining database schema, Extracting data, Adding or modifying data, Performing denial of service, Evading detection, Bypassing authentication, Executing remote commands, Performing privilege escalation.

In this chapter, we propose a novel scheme to detect the presence of SQLIA, combining static and dynamic analysis whose main features are: (*i*) it is based on obfuscation and deobfuscation of SQL statements, (*ii*) SQLIA can easily be detected and has a negligible run-time overhead because of dynamic verification is carried out on the obfuscated statements at atomic formula level and the number of verifications for possible SQLIA has been reduced by introducing the notion of secure and vulnerable terms and formulas, (*iii*) the obfuscation and deobfuscation techniques are application-independent and developers need not to be aware about this.

The structure of this chapter is as follows: in section 6.1, we discuss the related work in the literature. Section 6.2 defines the notion of secure and vulnerable terms and formulas involved in pre-condition ϕ. Section 6.3 describes the proposed SQLIA prevention technique and compares the proposal *w.r.t.* the literature.

6.1 Related Works

In (27, 92, 152, 181), the authors followed a model-based approach and introduced the tools AMNESIA, SQLGuard and SQLCheck respectively. All these schemata have two phases: static and dynamic. In AMNESIA, the static phase builds models corresponding to all legitimate SQL statements present in the application by analysing the program. In its run-time phase, before submitting the request to database, all dynamically generated SQL statements are checked against the corresponding model, and those violating the models are identified as SQLIA. Observe that, the accuracy of

the statically-built model is the measure of the success of AMNESIA. In SQLGuard and SQLCheck, the model is based on a set of grammar-rules against which the dynamically generated SQL statements are checked to detect the possibility of SQLIA. SQLGuard compares the parse tree of the SQL statement which is generated using grammar-rules, before and after the user input. Comments are included as a token in the parse tree to detect when attacker tries to comment out some active portion of the SQL statement. SQLCheck generates a set of augmented grammar-rules. The SQL statements are augmented and parsed by the augmented grammar. If parsing fails, it claims the presence of SQLIA. In both cases, the user inputs are augmented by some delimiter generated from a private key. Thus, the success of this two schemata are completely dependent on the fact that the attacker is not being able to discover the private key. In (152) the static model is based on graph representation of the SQL statements.

McClure and Krüger (142) proposed a completely different query development platform by changing the so-called unregulated query generation process that uses string concatenation, to a new systematic one. This solution consists of two parts. The first is an abstract object model. The second is an executable which is executed against a database and the output is a Dynamic Link Library (DLL) containing classes that are strongly-typed to the database schema. This DLL is used as a concrete instantiation of the abstract object model. However, as they provided a completely new paradigm for query development process which is not as easy as previous one, the developer need to learn before its use.

SQLrand (25) is based on the concept of Instruction-Set Randomization. The SQL standard keywords are manipulated by appending a random integer to them. The attacker is not aware about that random integer. Thus, if any malicious user attempting to SQL injection attack would immediately be thwarted as the injective codes in the randomized SQL statements are treated as non-keywords. In practice, it suffers from many aspects. First, a modified database would require all applications submitting SQL statements to conform to its new language. Second, the proxy server which intercepts randomize statements and de-randomizes the keywords imposes a significant infrastructure overhead. Third, this technique completely relies on the fact that the attacker is unable to discover the random secret number used to randomize.

In (188), Valeur et al. proposed a learning-based Intrusion Detection System (IDS) to detect SQLIA. The IDS is trained using a set of typical application queries. The technique builds statistical models of the typical queries, and then monitors the application at run-time to identify the queries that do not match the model. However, the

fundamental limitation is that the success of such system completely depends on the quality of the training set used. Poor training set would result large number of false positive and false negative.

Scott and Sharp, in their work (171), proposed a solution to provide an application level security including SQLIA for web-based applications. They use a security policy description language (SPDL) to specify a set of validation constraints and transformation rules to be applied to application parameters as they flow from the web page to the application server. The compiled SPDL codes are kept on a security gateway which acts as application level firewall. However, the developers are completely responsible for this. They have to know not only which data needs to be filtered, but also what patterns and filters need to apply to data.

Gould et al. (69) described the static type checking of dynamically generated queries. Although this technique was not developed with the intent to detect and prevent SQLIA, it can be used to prevent the attacks that take advantage of type mismatches in a dynamically generated query.

Many testing techniques (103, 122, 173) have been proposed to test whether an web application is vulnerable to SQLIA. The proposal in (103) is based on black-box approach, whereas the proposals in (122, 173) are based on white-box approach.

In (78, 104, 155, 158), the authors provided taint-based approaches to SQLIA: static analysis is used to check taint flows against preconditions for sensitive functions. The analysis detects the points in which preconditions have not been met and can suggest filters and sanitization functions that can be automatically added to the application to satisfy these preconditions.

Among the most recent works, (136) is a defensive coding practices to prevent SQLIA. It describes input validation (whitelist, blacklist) and encoding techniques (sanitize input) to ensure the safety of input. It also introduces a hybrid strategy combining them. The defensive approach suffers from the false positive or false negative problems. Defensive coding is prone to human error and is not as rigorously and completely applied as automated techniques.

Finally, Bertino et al. proposed a scheme in (16) that detects the presence of SQLIA by creating a fingerprint of the application based on SQL queries submitted by it to the database. Association rule mining techniques are then used to extract useful rules from this fingerprint.

6.2 Secure and Vulnerable Terms and Formulas

In this section, we define the terms and formulas in first-order logic, and we introduce the notion of secure and vulnerable terms with regards to different attacks.

Definition 13 (Terms) *The set of terms of a first-order language L is the set of strings of symbols formed according to the following rules:*

- *All the variable symbols $x_1, x_2, x_3, ...$ and all the constant symbols c_i in L are terms.*
- *If f_n is an n-ary function symbol in L and $t_1, t_2, ..., t_n$ are terms, then $f_n(t_1, t_2, ..., t_n)$ is a term.*

Definition 14 (Atomic Formula) *An atomic formula is a string of symbols of the form $R_n(t_1, t_2, ..., t_n) \in \{true, false\}$, where R_n is a relation symbol of arity n of the language and $t_1, t_2, ..., t_n$ are terms.*

If the language contains the equality relation, then any string of the form $t_1 = t_2$, where t_1, t_2 are terms, is also an atomic formula.

Definition 15 (Well-formed Formula) *The set of formulas of a language L is the set of strings of symbols formed according to the following rules:*

- *All atomic formulas are Well-formed Formulas (wffs).*
- *If ϕ and ψ are formulas and x_i is a variable symbol, then so are $\neg\phi, \phi \wedge \psi, \phi \vee \psi, \phi \rightarrow \psi, \phi \leftrightarrow \phi, \forall x_i \phi, \exists x_i \phi$.*

Now we define the notion of secure and vulnerable terms, atomic formulas and well-formed formulas. Let C_ϕ and V_ϕ denote the set of constants and variables, respectively, appearing in the pre-condition ϕ. Let C_{sec} and V_{sec} are set of secure constants and variables respectively whereas, V_{vul} stands for the set of public variables which are vulnerable to different attacks. Observe that, since constants and database variables are provided by the developers, we assume them as secure. Application variables may or may not be secure depending on whether they are used as public variables directly or influenced by other vulnerable variables indirectly. Therefore,

$$C_\phi \subseteq C_{sec} \text{ and } V_\phi \subseteq V_{sec} \cup V_{vul}$$

Let T_{sec}, AF_{sec} and WFF_{sec} represent the set of secure terms, atomic formulas and well-formed formulas, whereas T_{vul}, AF_{vul} and WFF_{vul} represent the set of vulnerable terms, atomic formulas and well-formed formulas respectively. We can define inference rules

for terms, atomic and well-formed formulas being secure and vulnerable as shown in Table 6.1.

Observe that the rules in Table 6.1 are not closed under logical equivalence, *i.e.* $\phi_1 \in WFF_{sec}$ and $\phi_2 \equiv \phi_1$ do not imply that $\phi_2 \in WFF_{sec}$. For instance, let $x \in V_{vul}$, $y \in V_{sec}$ and $c \in C_{sec}$, we have: $(y = c) \in WFF_{sec}$ while $(x = x) \wedge (y = c) \in WFF_{vul}$ even though the two formulas are equivalent. This is due to the fact that $(x = x)$ is a potential gateway for SQLIA.

Table 6.1: Inference rules for terms, atomic and well-formed formulas being secure and vulnerable, where $\vartheta \in \{\forall, \exists\}$ and $\theta \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$ and x is a bound variable

$\frac{c \in C_{sec}}{c \in T_{sec}}$	$\frac{v \in V_{sec}}{v \in T_{sec}}$	$\frac{v \in V_{vul}}{v \in T_{vul}}$	$\frac{\forall t_i \in T_{sec},\ i=1,\ldots,n}{f(t_1,\ldots,t_n) \in T_{sec}}$
$\frac{\exists t_i \in T_{vul},\ i=1,\ldots,n}{f(t_1,\ldots,t_n) \in T_{vul}}$	$\frac{\forall t_i \in T_{sec},\ i=1,\ldots,n}{R_n(t_1,\ldots,t_n) \in AF_{sec}}$	$\frac{\exists t_i \in T_{vul},\ i=1,\ldots,n}{R_n(t_1,\ldots,t_n) \in AF_{vul}}$	$\frac{t_1 \in T_{sec} \qquad t_2 \in T_{sec}}{(t_1 = t_2) \in AF_{sec}}$
$\frac{t_1 \in T_{vul}}{(t_1 = t_2) \in AF_{vul}}$	$\frac{t_2 \in T_{vul}}{(t_1 = t_2) \in AF_{vul}}$	$\frac{a \in AF_{sec}}{a \in WFF_{sec}}$	$\frac{a \in AF_{vul}}{a \in WFF_{vul}}$
$\frac{w \in WFF_{sec}}{\neg w \in WFF_{sec}}$	$\frac{w \in WFF_{vul}}{\neg w \in WFF_{vul}}$	$\frac{w \in WFF_{sec}}{(\vartheta\ x)\ w \in WFF_{sec}}$	$\frac{w \in WFF_{vul}}{(\vartheta\ x)\ w \in WFF_{vul}}$
$\frac{w_1 \in WFF_{vul}}{(w_1 \theta\ w_2) \in WFF_{vul}}$	$\frac{w_2 \in WFF_{vul}}{(w_1 \theta\ w_2) \in WFF_{vul}}$	$\frac{w_1 \in WFF_{sec} \qquad w_2 \in WFF_{sec}}{(w_1 \theta\ w_2) \in WFF_{sec}}$	

6.3 Proposed SQLIA Prevention Technique

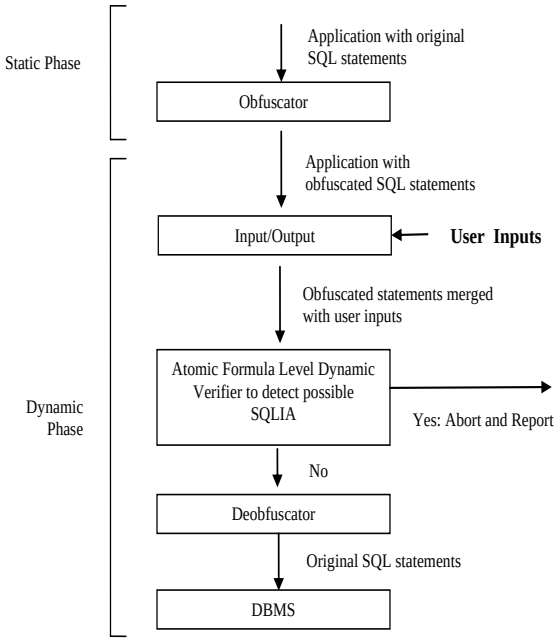
The proposed scheme has three phases, the first one is performed statically, while the latter two are performed dynamically.

1. Obfuscating the legitimate SQL statement Q into Q' at each hotspot of the application.
2. After merging the user inputs into the obfuscated SQL statement at run-time, the dynamic verifier checks the obfuscated statement at atomic formula level in order to detect the presence of possible SQLIA.
3. Reconstruction into the original SQL statement Q from the obfuscated statement Q' before submitting it to the database, if no possible SQLIA was detected.

The overall architecture of our approach is depicted in Figure 6.1 where at the beginning we assume that the user launches a job involving one or more web applications

interacting with the DBMS. In the static analysis phase, the SQL statements in the

Figure 6.1: Architecture of the proposed scheme for SQLIA



application are replaced by the ones in obfuscated form. The main idea behind the obfuscation of a SQL string is just to avoid the string concatenation operation in SQL code generation process, which is considered as the possible root cause of SQLIA (158). The obfuscation is carried out by converting the SQL statement $Q = \langle A, \phi \rangle$ into a form such that the pre-condition ϕ is partitioned into two sets S_c and S_f, where the former contains all connectives (AND, OR, NOT) of ϕ, and the latter contains all atomic formulas present in ϕ.

During run-time, the inputs provided by the user are merged into the atomic formulas in S_f. The dynamic verifier, then, verifies at atomic formula level for the possible existence of SQLIA. The task of the dynamic verifier is to determine whether the elements in the set S_f are valid atomic formulas or not. Thus the input of the dynamic verifier is an atomic formula merged with user input, and the output is a boolean value indicating whether it is a valid atomic formula or not. If any violation is detected in the verification phase, the dynamic verifier reports it as a possible SQLIA, otherwise the run-time converter (which may be part of the verifier) converts the obfuscated statement into the original form before submitting it to the DBMS.

The attractive features of this scheme is that the static phase removes the traditional string concatenation operations (which is the root cause of possible SQLIA) used to build all the SQL code in the application and replace them by an obfuscated form. Since the user inputs are merged into the atomic formulas in obfuscated form, there is no chance to mix-up the malicious input with the other legitimate control elements of the SQL statements, and the dynamic verification at atomic formula level can easily detect the presence of possible SQLIA. The number of dynamic verification is reduced by introducing two categories of terms and formulas: secure and vulnerable ones. The verification is carried out only over the vulnerable atomic formulas. Moreover, we can avoid obfuscation for those statements which have secure pre-conditions.

For the sake of simplicity, in the rest of this section, we consider the following assumptions:

1. The passive part ϕ of the SQL statement is a function of user input, whereas the active part is not.
2. The developer-provided part of SQL statement is reliable, whereas the user-provided part is not trusted and vulnerable to SQLIA.
3. Users are not permitted to provide any control elements of SQL statement. They are only allowed to provide the data elements to the SQL statement.

Observe that the assumptions above do not yield to severe limitations, and are reasonable in practice.

6.3.1 Obfuscation of SQL Statements

In this section, we discuss the obfuscation scheme. We first illustrate it on a simple example, and then we present the actual algorithm.

Suppose, $\phi_1, \phi_2, ..., \phi_n$ represent a set of atomic predicate formulas involved in a pre-condition ϕ of a SQL statement Q. For example, let the following formula represent a pre-condition ϕ which contains the atomic formulas $\phi_1, \phi_2, ..., \phi_7$:

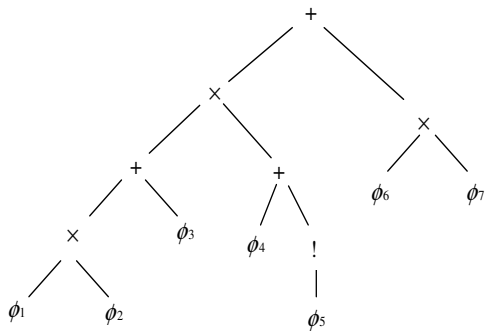
$$\phi = (\phi_1\ AND\ \phi_2\ OR\ \phi_3) AND (\phi_4\ OR\ (NOT \phi_5))\ OR\ \phi_6\ AND\ \phi_7$$

For the simplicity of representation, we denote AND, OR, NOT by $\times, +$ and ! respectively. Thus,

$$\phi = (\phi_1 \times \phi_2 + \phi_3) \times (\phi_4 + (!\phi_5)) + \phi_6 \times \phi_7$$

Since the connectives are unary or binary, the parse tree of ϕ represents a binary tree as shown in Figure 6.2. The obfuscation of the original SQL statement Q is obtained by

Figure 6.2: The parse-tree of ϕ



converting the pre-condition ϕ in such a form in which it is separated into two distinct partitions. The first partition contains all the connectives $(\times, +, !)$ of ϕ, whereas the second partition contains all the atomic formulas $\phi_1, \ldots, \phi_n$ of ϕ.

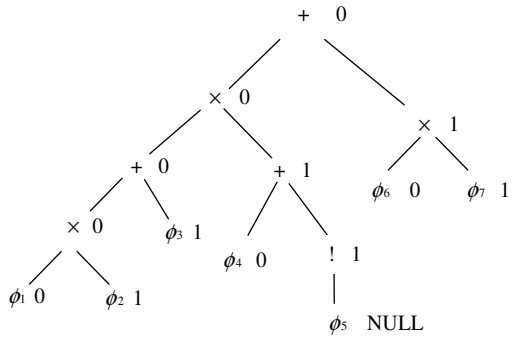
Observe that if we convert ϕ into some prefix or postfix form considering $\times, +, !$ as operators and ϕ_i, $i = 1, \ldots, n$, as operands, still there is a mixing of connectives and formulas. For example, if we convert the above formula ϕ into prefix form we get: $+ \times + \times \phi_1\phi_2\phi_3 + \phi_4!\phi_5 \times \phi_6\phi_7$. Since only a fraction of the connectives $(\times, +, !)$ of ϕ has been separated, still there is a chance of possible SQLIA.

To remove this problem and to obtain two exclusive partitions of connectives and atomic formulas, we adopt a different technique consisting of the following steps:

> *Assign a unique label to each of the connectives* $(\times, +, !)$ *and atomic formulas* ϕ_i*, i=1,..,n in* ϕ*, and partition the connectives and atomic formulas into two different sets.*

The task of assigning unique label is performed as follows: we start assigning a bit to each node in the parse tree by assigning 0 to the root node. If any internal node has single child, assign NULL to that child node. Otherwise, the left and right child of the node are assigned with 0 and 1 respectively. Continue this process until all the nodes of the parse tree has been assigned by bits. The bit-assigned binary parse tree of the formula ϕ of our example is shown in Figure 6.3. We know that each node in a tree has a unique path from the root to that node. To get a unique label for a node v, traverse the path from root to the node v in the tree. Collect all the bits of the nodes appearing in that path and concatenate them in the direction of traversing. This gives a unique binary string which is used as a unique label. For example, in the bit-assigned parse tree of Figure 6.3, the unique label for ϕ_4 is 0010, and for ϕ_5 is 0011 (equivalently,

Figure 6.3: The bit-assigned parse-tree of ϕ



0011NULL). This unique labels allow to reconstruct the original SQL statement from the obfuscated form (as described later).

We convert the above pre-condition ϕ into a new form in which each token is represented by a tuple $\langle c, l_c \rangle$ or $\langle f, l_f \rangle$, where $c \in \{\times, +, !\}$ and $f \in \{\phi_i \mid i = 1, .., n\}$. The unique labels l_c and l_f obtained from the bit-assigned parse tree, are associated with c and f respectively. The converted ϕ results into:

$(\langle \phi_1, 00000 \rangle \langle \times, 0000 \rangle \langle \phi_2, 00001 \rangle \langle +, 000 \rangle \langle \phi_3, 0001 \rangle) \langle \times, 00 \rangle (\langle \phi_4, 0010 \rangle \langle +, 001 \rangle (\langle !, 0011 \rangle \langle \phi_5, 0011 \rangle))$
$\langle +, 0 \rangle \langle \phi_6, 010 \rangle \langle \times, 01 \rangle \langle \phi_7, 011 \rangle$

Now we are in position to construct two distinct sets S_c and S_f taking all the connectives into one and all the atomic formulas into the other respectively. Thus,

$$S_c = \{\langle \times, 0000 \rangle, \langle +, 000 \rangle, \langle \times, 00 \rangle, \langle +, 001 \rangle, \langle !, 0011 \rangle, \langle +, 0 \rangle, \langle \times, 01 \rangle\}$$

$$S_f = \{\langle \phi_1, 00000 \rangle, \langle \phi_2, 00001 \rangle, \langle \phi_3, 0001 \rangle, \langle \phi_4, 0010 \rangle, \langle \phi_5, 0011 \rangle, \langle \phi_6, 010 \rangle, \langle \phi_7, 011 \rangle\}$$

These two sets S_c and S_f represent the obfuscated pre-condition. So, the obfuscated SQL statement can be written as: $Q' = \langle A, [S_c, S_f] \rangle$.

It is worthwhile to mention that the order of the elements in the two partitions S_c and S_f are not relevant at all. At run-time, inputs given by the users are merged into the atomic formulas in S_f. Since S_f is a set of atomic formulas and user inputs are part of the elements in S_f at run-time, there is a strong chance for the attacker to change the number of elements (atomic formulas) in S_f. But their target can not be successful anymore because of the fixed number of connectives present in the set S_c. We can see in the next section that, any change in the number of atomic formulas in S_f would yield the failure of the deobfuscation phase.

The dynamic verifier will verify the atomic formulas for the possible SQLIA after

getting run-time inputs. The input of the dynamic verifier is an atomic formula merged with user input and the output is a boolean value indicating whether it is a valid atomic formula or not. To reduce the number of verifications, we categorize each atomic formula in S_f based on whether it belongs to AF_{sec} or AF_{vul} and tag them accordingly at the end of the static phase. During the dynamic phase, only the atomic formulas which are tagged as vulnerable will be checked for possible SQLIA. Also note that for any SQL statement $Q = \langle A, \phi \rangle$, we do not obfuscate it if $\phi \in WFF_{sec}$. In case of nested queries, the obfuscation-verification-deobfuscation procedure is performed from the inner-most query to the outer-most one.

We are now in position to formalize the algorithm performing this obfuscation part, as shown in Figure 6.4.

Figure 6.4: SQL Obfuscation Algorithm

Algorithm 5: Obfuscate_SQL

Input: Original SQL statement $Q = \langle A, \phi \rangle$
Output: Obfuscated SQL statement $Q' = \langle A, [S_c, S_f] \rangle$

1. Check whether $\phi \in WFF_{sec}$ or not. If not, perform steps 2-8.
2. Generate binary parse tree of the pre-condition ϕ of Q.
3. Assign root node by 0. If any node has single child, assign NULL to this child and if any node has two children, assign left child by 0 and right child by 1, respectively. Continue this step until all the nodes of the tree are assigned.
4. Assign each of the atomic formulas and connectives in ϕ by a unique label obtained from the parse tree as follows: for a node v, traverse from root to node v and collect all bits of the nodes appearing in the path. Concatenate all those bits in the direction of traversing. This gives a bit string which is used as the unique label for the node v.
5. After performing step 4, all connectives c and atomic formulas f of ϕ will be of the form $\langle c, l_c \rangle$ and $\langle f, l_f \rangle$, where l_c and l_f denote the unique labels assigned to c and f, respectively.
6. Partition the connectives $\langle c, l_c \rangle$ and atomic formulas $\langle f, l_f \rangle$ of ϕ into two distinct sets S_c and S_f, respectively.
7. If for $\langle f, l_f \rangle \in S_f$ and $f \in AF_{sec}$, tag it as secure *i.e.* $tag(f) := sec$; otherwise, tag it as vulnerable *i.e.* $tag(f) := vul$.
8. The obfuscated form of the SQL statement Q is, therefore, $Q' = \langle A, [S_c, S_f] \rangle$.

6.3.2 Deobfuscation of SQL Statements

We already mentioned that deobfuscation is done only when no SQLIA has been detected by dynamic atomic formula level verifier. Of course, if the original SQL statements are kept, then there is no need of deobfuscation. If this is not the case, the following technique reconstructs the original SQL statement $Q = \langle A, \phi \rangle$ from the obfuscated form $Q' = \langle A, [S_c, S_f] \rangle$.

To restore ϕ from S_c and S_f (hence, from Q' to Q) in the dynamic phase before submitting it to the database engine, the following steps are performed repeatedly until S_c is empty:

Step 1: Identify the predicate formulas in S_f whose label matches with the label of the connectives in S_c *i.e.* $\exists \langle c, l_c \rangle \in S_c, \exists \langle f, l_f \rangle \in S_f$: $|l_c| = |l_f|$ and $l_c \otimes l_f = 0$. $\otimes$ represents bit-wise XOR operation. Apply the unary connective c on the corresponding matched formula f to get resulting formula f_r. Replace $\langle f, l_f \rangle$ by $\langle f_r, l_c \rangle$ in S_f. Remove $\langle c, l_c \rangle$ from S_c.

Step 2: Identify all the pair of elements $\{\langle f_1, l_1 \rangle, \langle f_2, l_2 \rangle\} \subseteq S_f$ such that, $|l_1| = |l_2|$ and only the last bit of l_1 and l_2 differs. Identify the connective $\langle c, l \rangle \in S_c$ where $|l| = |l_1| - 1 = |l_2| - 1$ and l is equal to the common part of l_1 and l_2. Apply the binary connective c on that pair to obtain the resulting formula f_r. Replace the pair by a new tuple $\langle f_r, l \rangle$ in S_f and remove $\langle c, l \rangle$ from S_c.

The formal description of possible SQLIA detection and deobfuscation algorithm is shown in Figure 6.5.

We illustrate the deobfuscation technique with the same example of section 6.3.1. Suppose, the obfuscated SQL statement $Q' = \langle A, [S_c, S_f] \rangle$ where S_c and S_f are:

$$S_c = \{\langle \times, 0000 \rangle, \langle +, 000 \rangle, \langle \times, 00 \rangle, \langle +, 001 \rangle, \langle !, 0011 \rangle, \langle +, 0 \rangle, \langle \times, 01 \rangle\}$$

$$S_f = \{\langle \phi_1, 00000 \rangle, \langle \phi_2, 00001 \rangle, \langle \phi_3, 0001 \rangle, \langle \phi_4, 0010 \rangle, \langle \phi_5, 0011 \rangle, \langle \phi_6, 010 \rangle, \langle \phi_7, 011 \rangle\}$$

We perform the two steps above repeatedly until the set S_c is empty.

1. In the example, the label of $\langle \phi_5, 0011 \rangle \in S_f$ matches with the label of $\langle !, 0011 \rangle \in S_c$. So after performing step 1 we get:

$$S_c = \{\langle \times, 0000 \rangle, \langle +, 000 \rangle, \langle \times, 00 \rangle, \langle +, 001 \rangle, \langle +, 0 \rangle, \langle \times, 01 \rangle\}$$

$$S_f = \{\langle \phi_1, 00000 \rangle, \langle \phi_2, 00001 \rangle, \langle \phi_3, 0001 \rangle, \langle \phi_4, 0010 \rangle, \langle (!\phi_5), 0011 \rangle, \langle \phi_6, 010 \rangle, \langle \phi_7, 011 \rangle\}$$

Figure 6.5: Algorithm to Detect possible SQLIA and deobfuscation of the SQL statements

Algorithm 6: Deobfuscate_SQL

Input: Obfuscated SQL statement $Q' = \langle A, [S_c, S_f] \rangle$
Output: Claiming for possible SQLIA as true or false; Original SQL statement $Q = \langle A, \phi \rangle$

1. Perform dynamic verification on all atomic formulas $\langle f, l_f \rangle \in S_f \wedge tag(f) = vul$ merged with run-time inputs given by the users, for any possible violation. If violates, claim:=true else claim:=false.
2. If claim=false, perform steps 3(a), 3(b), 4(a), 4(b) & 5 until S_c is empty.

3(a). Identify the predicate formulas in S_f whose label matches with the label of the connectives in S_c *i.e.* $\exists \langle c, l_c \rangle \in S_c, \exists \langle f, l_f \rangle \in S_f$: $|l_c| = |l_f|$ and $l_c \otimes l_f = 0$. $\otimes$ represents bit-wise XOR operation.

3(b). Apply the unary connective c on the corresponding matched formula f to get resulting formula f_r. Replace $\langle f, l_f \rangle$ by $\langle f_r, l_c \rangle$ in S_f. Remove $\langle c, l_c \rangle$ from S_c.

4(a). Identify all the pair of elements $\{\langle f_1, l_1 \rangle, \langle f_2, l_2 \rangle\} \subseteq S_f$ such that, $|l_1| = |l_2|$ and only the last bit of l_1 and l_2 differs.

4(b). Identify the connective $\langle c, l \rangle \in S_c$ where $|l| = |l_1| - 1 = |l_2| - 1$ and l is equal to the common part of l_1 and l_2. Apply the binary connective c on that pair to obtain the resulting formula f_r. Replace the pair by a new tuple $\langle f_r, l \rangle$ in S_f and Remove $\langle c, l \rangle$ from S_c.

5. If S_c is empty, S_f contains the original form of pre-condition ϕ and submit $Q = \langle A, \phi \rangle$ to the DBMS.

2. Following step 2, we get three pairs whose labels are equal in length and differs by the last bit only: $\{\langle \phi_1, 00000 \rangle, \langle \phi_2, 00001 \rangle\}$, $\{\langle \phi_4, 0010 \rangle, \langle (!\phi_5), 0011 \rangle\}$, and $\{\langle \phi_6, 010 \rangle, \langle \phi_7, 011 \rangle\}$. Since the label of $\langle \times, 0000 \rangle$ equals to the common part of the pair $\{\langle \phi_1, 00000 \rangle, \langle \phi_2, 00001 \rangle\}$ and similarly, $\langle +, 001 \rangle$ and $\langle \times, 01 \rangle$ for the pairs $\{\langle \phi_4, 0010 \rangle, \langle (!\phi_5), 0011 \rangle\}$ and $\{\langle \phi_6, 010 \rangle, \langle \phi_7, 011 \rangle\}$ respectively, after performing step 2, we get,

$$S_c = \{\langle +, 000 \rangle, \langle \times, 00 \rangle, \langle +, 0 \rangle\}$$

$$S_f = \{\langle \phi_1 \times \phi_2, 0000 \rangle, \langle \phi_3, 0001 \rangle, \langle \phi_4 + (!\phi_5), 001 \rangle, \langle \phi_6 \times \phi_7, 01 \rangle\}$$

3. Since S_c is not empty, we repeat the same until S_c is empty and finally we obtain,

$$\phi = ((\phi_1 \times \phi_2 + \phi_3) \times (\phi_4 + (!\phi_5)) + \phi_6 \times \phi_7)$$

In this way, we can recover the original SQL statement $Q = \langle A, \phi \rangle$ from the obfuscated form $Q' = \langle A, [S_c, S_f] \rangle$.

Lemma 8 *(Time complexity) The time complexity of obfuscation and deobfuscation of a SQL statement are $O(n + m)$ and $O(n^2)$ respectively, where n and m are the number of atomic formulas and connectives in the pre-condition ϕ.*

Proof The time complexity of the obfuscation step mainly depends on two facts: (*i*) bit-assigned parse tree generation and, (*ii*) assigning unique label to each connectives and atomic formulas of ϕ.

Let n and m be the number of atomic formulas and connectives present in the pre-condition ϕ. All the leaf nodes of the parse tree are atomic formulas whereas internal nodes are the connectives.

The generation of parse tree as well as traversing of it to assign bits and extracting unique labels for each of the nodes, needs traversing every nodes exactly once. Hence, the time complexity of the obfuscation is $O(n + m)$.

Time complexity of the deobfuscation step depends on two main operations: (*i*) match the labels of unary connectives with the labels of elements in S_f, (*ii*) find the pairs in S_f and binary connectives in S_c with specific criterion. Let p be the number of unary connectives in S_c. It is obvious that $m - p = n - 1$. The worst case time complexity for the first operation is $O(np)$ whereas best case is $O(p^2)$. The time needed to perform the second operation is $O(n^2 + nm)$. Since, $n \geq m$ and $n \geq p$, the worst-case time complexity for the deobfuscation step is, therefore, $O(n^2)$.

6.3.3 Example

Let us illustrate the overall scheme with an example.

Example 16 *Recall the example of introduction part where an application generates the following SQL query:*

```
Q="SELECT * FROM emp WHERE username='" +
   request.getParameter("username") + "' AND password='" +
   request.getParameter("password") + "';"
```

The above query can be represented by the following components:

$$
\begin{aligned}
A: &\quad \text{“}\texttt{SELECT * FROM emp}\text{”} \\
\phi_1: &\quad \text{“}\texttt{username= ‘}input_1\texttt{’}\text{”} \\
\phi_2: &\quad \text{“}\texttt{password= ‘}input_2\texttt{’}\text{”} \\
\phi: &\quad \phi_1 \ \texttt{AND}\ \phi_2 \\
Q: &\quad \langle A, \phi \rangle
\end{aligned}
$$

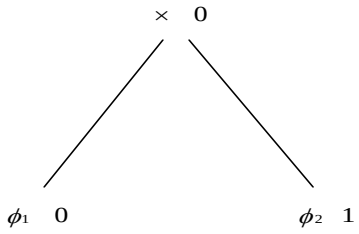
where,

$$input_1 = \texttt{request.getParameter ("username")}$$

$$input_2 = \texttt{request.getParameter("password")}$$

The bit-assigned binary parse tree of ϕ of this example is shown in Figure 6.6. Thus, the

Figure 6.6: The bit-assigned parse-tree of the example pre-condition ϕ



obfuscated query is: $Q' = \langle A, [S_c, S_f] \rangle$ where, $S_c = \{ \langle \times, 0 \rangle \}$ and $S_f = \{ \langle \phi_1, 00 \rangle, \langle \phi_2, 01 \rangle \}$.

Suppose, the attacker gives the following inputs: “`alice' OR '1'='1`*” and “*`secret`*” for $input_1$ and $input_2$ respectively. After merging by these inputs, the atomic formulas in S_f of the obfuscated query would be:*

$$
\begin{aligned}
\phi_1: &\quad \text{“}\texttt{username= ‘alice' OR '1'='1'}\text{”} \\
\phi_2: &\quad \text{“}\texttt{password= ‘secret'}\text{”}
\end{aligned}
$$

Clearly, the dynamic verification at atomic formula level says that ϕ_1 is no more indicating a valid atomic formula. So it can be identified as a possible SQLIA.

6.3.4 Static Vs. Dynamic Issues

Previous works (27, 92, 152) proposed mechanisms to generate static models that are used to verify against the SQL statements at dynamic time where the user inputs are

allowed to merge into the original SQL statements. However, theses schemes may yield to false positive in cases where the users are allowed to provide structural attributes as input. For instance, suppose that a web application generates the following SQL query:

```
"SELECT * FROM users WHERE id=" + request.getParameter("id") + ";"
```

If the web application allows users to provide arbitrary arithmetic expression as input, the structure of the SQL statements depends on the expression in the *id* field. Since (27, 92, 152) rely on the fixed static model built at compile time, they yield to a false positive. However our scheme can treat them as valid atomic formulas.

Systems like ModSecurity (147) are provided with input validation using defensive coding practices. They use a white-list/black-list approach to allow/block the good/bad inputs in order to prevent possible SQLIA. These systems are application-specific and developers are responsible to create and maintain white/black-list for specific applications: this is prone to possible human error and can cause both false positives and false negatives. The advantages of our scheme over defensive coding practices is that the developers need not to be aware about the obfuscation-deobfuscation, and completely application independent.

The technique in (25) is based on randomization of the keywords of all SQL statements in the application by appending each keyword with a random value. In practice, it suffers from many aspects. First, a modified database would require for all applications submitting SQL statements to conform to its new language. Second, the proxy server which is responsible to check the syntactic validity of the whole randomized statements and de-randomize the instruction set for syntactically valid ones, incurs a significant infrastructure overhead. Third, this technique completely relies on the fact that the attacker is unable to discover the random secret number used to randomize (93). However, in our scheme, the dynamic verifier checks only the atomic formulas appearing in the statements and this run-time overhead is further minimized by introducing secure and vulnerable atomic formulas: verification is carried out over vulnerable atomic formulas only. Furthermore, unlike (25), for the SQL statements that contain secure passive part, we do not apply obfuscation-deobfuscation to avoid unnecessary processing.

In brief, the obfuscation/deobfuscation approach presented above has the following advantages: the verification for the presence of possible SQLIA is performed at atomic formula level and only on those atomic formulas which are tagged as vulnerable; the scheme avoids the root cause (string concatenation operation) of SQLIA in traditional

dynamic SQL code generation; the developer can enjoy the traditional application development techniques and need not to be aware about the obfuscation/deobfuscation techniques. Unlike most of the existing methods, this scheme does not depend on the static models or the private key. It depends only on the accuracy of the dynamic verifier at atomic formula level.

Chapter 7

Cooperative Query Answering

[Part of this chapter is already published in (84)]

Traditional query processing system enforces database-users to issue precisely specified queries, while the system provides limited and exact answers, or no information at all when the exact answer is unavailable in the database. Therefore, it is important to the database-users to fully understand problem domain, query syntax, database schema, and underlying database content.

To remedy such shortcomings and to enhance the effectiveness of the information retrieval, the notion of cooperative query answering (34, 34, 36, 61, 115) has been explored. The cooperative query answering system provides users an intelligent database interface that allows them to issue approximate queries independent to the underlying database structure and its content, and provides additional useful information as well as the exact answers.

As an example, in response to the query about "specific flight departing at *10 a.m.* from *Rome Fiumicino* airport to *Paris Orly* airport" the cooperative query answering system may return "all flight information during *morning* time from airports in *Rome* to airports in *Paris*", and thus, the user will be able to choose other flight if the specific flight is unavailable. Such query answering is also known as neighborhood query answering, as instead of providing exact answers it captures neighboring information as well. Cooperative query answering system also gives users the opportunity to issue conceptual or approximate queries where they might ask more general questions, for example, "how to travel from *Rome* to *Paris* at a *reasonable* cost during *morning* time" or "find the flights that fly during *night* only" without knowing the exact database schema and its content. One of the benefits of issuing conceptual queries is to avoid

reissuing of the set of concrete queries if the corresponding conceptual query returns empty result.

Cooperative query answering depends on the context in which queries are issued. The context includes the identity of the issuer, the intent or purpose of the query, the requirements that make explicit the answers relevant to the user etc. The following example illustrates it clearly: suppose a user issues a query asking the list of airports that are similar to "*Venice Marcopolo*" airport. Different contexts define the meaning of "*similarity between airports*" differently. For instance, to any surveyor "*similarity*" may refer in terms of the size and facilities provided in the airport, whereas to any flight company "*similarity*" may refer in terms of business point of view *i.e*. flight landing charges or other relevant taxes.

In chapter 3, we introduced the Abstract Interpretation framework to the field of database query languages as a way to provide sound approximation of the query languages. In this chapter, we extend this to the field of cooperative query answering system: we propose a cooperative query answering scheme based on the Abstract Interpretation framework that consists of three phases - transformation of the whole query system by using abstract representation of the data values, cooperative query evaluation over the abstract domain, and concretization of the cooperative abstract result. The main contributions in this chapter are: (*i*) we express the cooperative query evaluation by abstract databases, (*ii*) we express how to deal with cooperative query evaluation in presence of aggregate and negation operations, (*iii*) we address three key issues: soundness, relevancy and optimality of the cooperative answers.

The structure of this chapter is as follows: Section 7.1 discusses related work in the literature and motivation of our work. Section 7.2 describes the key issues in the context of cooperative query answering. In Section 7.3, we discuss our proposed scheme and we show how our proposal is able to address the key issues.

7.1 Related Work and Motivation

Several techniques have been proposed in the literature based on logic model, semantic distance, fuzzy set theory, abstraction, and so on. The logic-based models (26, 61) use first-order predicate logic to represent the database, the knowledge-base, and the users' queries. Content of the knowledge base helps in guiding query reformulation into more flexible and generalized query that provides relaxed, intelligent cooperative answers. However, these approaches have limitations in guiding the query relaxation process and the less intuitive query answering process due to lack of its expressiveness.

The semantic distance-based approaches (106, 157) use the notion of semantic distance to represent the degree of similarity between data values, and provide ranked cooperative results sorted according to their semantic distances. For categorical data, distances between data values are stored in a table. However, since every pair is supposed to have semantic distances, in realistic application domain these approaches are inefficient as the table size gets extremely larger and becomes harder to maintain the consistency of the distance measures.

In (76), the initial queries are transformed into flexible form based on knowledge base and fuzzy set theory. Finally, these queries are rewritten into boolean queries and evaluated to the traditional database.

In abstraction-based models (35, 36), the data are organized into multilevel abstraction hierarchies where nodes at higher level are the abstract representation of the nodes at lower level. The cooperative query answering is accomplished by generating a set of refined concrete queries by performing query abstraction and query refinement process by moving upward or downward through the hierarchy. Finally, the refined queries are issued to the database that provide additional useful information. These approaches suffer from high overhead when the degree of relaxation for a query is large, as the query abstraction-refinement process produces a large set of concrete queries to be issued to the database. To remedy this, fuzzy set theory or semantic distance approach is combined with abstraction hierarchy (34, 105, 115) in order to control the abstraction-refinement process and to provide a measure of nearness between exact and approximate answers.

However, all the schemes mentioned above do not provide any formal framework to cooperative query answering system. In addition, none of these schemes enlightens the key issues: soundness, relevancy and optimality in the context of cooperativeness of the query answers. Most of the existing techniques (34, 35, 36, 105, 115) suffer from the problem of soundness when query contains set operations UNION, INTERSECT, MINUS. In case of conceptual or approximate queries where approximate results are desirable, none of the schemes focuses on the way to compute aggregate functions when appearing in a query so as to preserve the soundness.

7.2 Key Issues: Soundness, Relevancy, Optimality

Any cooperative query answering scheme should respect three key issues: soundness, relevancy, and optimality. Intuitively, a cooperative query answer is *sound* if it is equal to or it is a superset of the corresponding extensional query answer. The *relevancy* of

the answers *w.r.t.* the context concerns with avoiding the tuples that have no value to the user: all the information in the cooperative answer should have relevancy to the user. The third criterion *optimality* implies that the system should return as much information as possible, while satisfying the first two properties. Since there exist many cooperative answers corresponding to a given query under given context, the optimality describes the "preferability" of the query answers to the user.

Given a cooperative query processing system $\mathfrak{B}$, a database dB, a context C, and a query Q, the result obtained by the cooperative system is $R = \mathfrak{B}(dB, C, Q)$.

Definition 16 (Soundness) *Given a database dB, a query Q, and a context C. Let R be the extensional query answer obtained by processing Q on dB. The cooperative answer* $R' = \mathfrak{B}(dB, C, Q)$ *is sound if* $R' \supseteq R$.

The relevancy of the information to a user depends on his interests. These interests can be expressed in terms of constraints represented by well-formed formulas in first order logic. If the information provided by a cooperative system satisfies the set of constraints representing user's interests, it is treated as relevant. For instance, "flight duration in the result set must be less than 3 hours" can be used as a constraint that determines the relevancy of the information in the cooperative answer.

Definition 17 (Relevancy) *Given a database dB, a query Q, and a context C. Let* $S(Q)$ *be the set of constraints represented by well-formed formulas in first order logic that make explicit the answers relevant to the user. The cooperative answer* $R = \mathfrak{B}(dB, C, Q)$ *respects the relevancy if* $\forall x \in R : \; x \models S(Q)$.

It is worthwhile to mention that the system relaxes users' queries to obtain neighboring and conceptually related answers in addition to the exact answer. However, the formulas appearing in the pre-condition ϕ of the query Q is different from the set of constraints appearing in $S(Q)$ that determines the relevancy of the cooperative answers. The constraints in $S(Q)$, in contrast to ϕ, is strict in the sense that there is no question of relaxing them, and violation of any of these constraints by the information in the cooperative answer will be treated as irrelevant.

A cooperative system may return different cooperative answers to a user in a given context. However, it is sensible to define a measure that describes the "preferability" of each answer. A cooperative answer is called more optimal than another answer if it is more preferable to the user in the given context than the other.

Definition 18 (Optimality) *Given a database dB, a query Q, a context C, and a set of constraints* $S(Q)$ *expressing user's requirements. The cooperative answer* $R = \mathfrak{B}(dB, C, Q)$ *is*

more optimal than $R' = \mathfrak{B}'(dB, C, Q)$ *if*

$$\{x \in R : x \models S(Q)\} \supseteq \{x \in R' : x \models S(Q)\} \text{ and } \{y \in R : y \not\models S(Q)\} \subseteq \{y \in R' : y \not\models S(Q)\}$$

In other words, a cooperative answer is called more optimal than another answer when it contains more relevant information and less irrelevant information *w.r.t.* $S(Q)$ than the other.

7.3 Proposed Scheme

Our proposal consists of three phases: (*i*) Transforming the databases and its query languages by using abstract representation of the data values, (*ii*) Cooperative query evaluation over the abstract domain, and finally, (*iii*) Concretization of the cooperative abstract result.

7.3.1 Transforming from Concrete to Abstract Domain

Given a concrete database dB, we transform it into an abstract database by using the Galois Connection $(\wp(D_x^{con}), \alpha_x, \gamma_x, D_x^{abs})$, where $\wp(D_x^{con})$ and D_x^{abs} represent the powerset of concrete domain of x and the abstract domain of x respectively, whereas α_x and γ_x represent the corresponding abstraction and concretization functions (denoted $\alpha_x : \wp(D_x^{con}) \rightarrow D_x^{abs}$ and $\gamma_x : D_x^{abs} \rightarrow \wp(D_x^{con})$) respectively. In case of partial abstract databases, for some attributes x the abstraction and concretization functions are identity functions id, and thus, use the Galois Connection $(\wp(D_x^{con}), id, id, \wp(D_x^{con}))$.

The level of approximation of the database information obtained by abstraction gives a measure of the "preferability" of the cooperative answers and depends on the context in which queries are issued. The best correct approximation (64) of the database information according to the context provides the optimal cooperative answers to the end-users.

Example 17 *The database in Table 7.1(a) consists of concrete table "*$flight$*" that provides available flight information to the end-users of a travel agent application. The corresponding abstract table "*$flight^\sharp$*" is shown in Table 7.1(b) where source and destination airports are abstracted by the provinces they belong, the numerical values of the cost attribute are abstracted by the elements from the domain of intervals, the values of the start-time/reach-time attributes are abstracted by the periods from the abstract domain* $PERIOD = \{\bot, morning, afternoon, evening, night, \top\}$ *where* $\top$ *represents "anytime" and* $\bot$ *represents "don't know", the flight no. and availability attributes are abstracted by the topmost element* $\top$ *of their corresponding*

abstract lattices. Observe that the number of abstract tuples in an abstract database may be less than that in the corresponding concrete database.

Table 7.1: A concrete and its corresponding Abstract Database

(a) Database containing concrete table "$flight$"

flight no.	source	destination	cost ($)	start-time	reach-time	availability
F001	Fiumicino (FCO)	Orly (ORY)	210.57	8.05	10.45	N
F002	Marcopolo (VCE)	Orly (ORY)	410.30	18.30	21.00	Y
F003	Ciampino (CIA)	Roissy Charles de Gaulle (CDG)	300.00	6.30	8.30	Y
F004	Urbe (LIRU)	Lyon-Saint Exupéry (LYS)	128.28	22.05	23.40	N
F005	Treviso (TSF)	Granby-Grand County (GNB)	200.15	16.00	17.20	Y
F006	Viterbo (LIRV)	Beauvais (BVA)	310.30	7.20	9.30	Y

(b) Abstract database containing abstract table "$flight^\sharp$"

flight no.$^\sharp$	source$^\sharp$	destination$^\sharp$	cost$^\sharp$	start-time$^\sharp$	reach-time$^\sharp$	availability$^\sharp$
$\top$	Rome	Paris	[200.00-249.99]	morning	morning	$\top$
$\top$	Venice	Paris	[400.00-449.99]	evening	night	$\top$
$\top$	Rome	Paris	[300.00-349.99]	morning	morning	$\top$
$\top$	Rome	Lyon	[100.00-149.99]	night	night	$\top$
$\top$	Venice	Lyon	[200.00-249.99]	afternoon	evening	$\top$

7.3.2 Cooperative Query Evaluation

The cooperative query evaluation is performed over (partial) abstract database $dB^\sharp$ in an abstract domain of interest. Given a query Q, the proposed cooperative system generates its abstract version $Q^\sharp$ and executes it on the abstract database $dB^\sharp$. The abstract query evaluation in an abstract domain of interest is described in detail in chapter 3. The cooperative system follows the same to execute users' queries by transforming them into the corresponding abstract versions.

Example 18 *Consider an online booking application interacting with the concrete database depicted in Table 7.1(a). Suppose a user wants to travel from Rome Fiumicino airport to Paris Orly airport by a flight such that flight cost is less than or equal to 300 USD. So the following query satisfying the required criterion can be issued:*

$Q_1 =$ `SELECT * FROM` $flight$ `WHERE` $source = "Fiumicino"$ `AND` $destination = "Orly"$ `AND` $cost \leq 300.00$ `AND` $availability = Y;$

Observe that the result ξ_1 of the query Q_1 is empty i.e. $\xi_1 = \emptyset$, because seats are not available in the flight from Rome Fiumicino to Paris Orly airport.

To obtain cooperative answers, we lift the whole query system from concrete to the abstract domain of interests by abstracting the database information and the associated query languages. The abstract database corresponding to the concrete database (Table 7.1(a)) is depicted in Table 7.1(b), and the abstract query corresponding to the concrete query Q_1 is shown below:

$$Q_1^\sharp = \texttt{SELECT}^\sharp \texttt{ * FROM flight}^\sharp \texttt{ WHERE } source^\sharp =^\sharp \text{ ''Rome'' } \texttt{AND } destination^\sharp =^\sharp \text{ ''Paris'' } \texttt{AND } cost^\sharp \leq^\sharp [300.00, 349.99] \texttt{ AND } availability^\sharp =^\sharp \top;$$

where the abstract operation $\leq^\sharp$ for intervals is defined as follows:

$$[l_i, h_i] \leq^\sharp [l_j, h_j] \triangleq \begin{cases} true \;\; if\, h_i \leq l_j \\ false \;\; if\, l_i > h_j \\ \top \;\; otherwise \end{cases}$$

and the abstract equality $=^\sharp$ is defined as usual.

When the imprecise abstract query $Q_1^\sharp$ is executed over the abstract database (Table 7.1(b)), it returns the flight information depicted in Table 7.2. This way, the abstraction of the databases and its query languages helps in obtaining additional information in the result.

Table 7.2: $\xi_1^\sharp$: Result of $Q_1^\sharp$

flight no.$^\sharp$	source$^\sharp$	destination$^\sharp$	cost$^\sharp$	start-time$^\sharp$	reach-time$^\sharp$	availability$^\sharp$
$\top$	Rome	Paris	[200.00-249.99]	morning	morning	$\top$
$\top$	Rome	Paris	[300.00-349.99]	morning	morning	$\top$

The cooperative query evaluation in presence of aggregate functions and set operations is similar as defined in chapter 3. Observe that soundness is preserved as the concretization of the abstract queries always results into a sound approximation of the corresponding concrete queries.

7.3.3 Concretization of the cooperative abstract result

Given a concrete and the corresponding abstract databases dB and $dB^\sharp$ respectively, let $\xi^\sharp$ be the abstract answer obtained by executing the abstract query $Q^\sharp$ on $dB^\sharp$. The cooperative answer $R = \mathfrak{B}(dB, C, Q)$ returned to the user is obtained by: $R = \gamma(\xi^\sharp) \cap dB$. That is, the cooperative answer is obtained by mapping the abstract result into its concrete counterpart. For instance, after mapping $\xi_1^\sharp$ (Table 7.2), the user gets its concrete counterpart R_1 shown in Table 7.3.

Table 7.3: R_1: Concrete Result obtained by concretizing the abstract result $\xi_1^\sharp$

flight no.	source	destination	cost ($)	start-time	reach-time	availability
F001	Fiumicino (FCO)	Orly (ORY)	210.57	8.05	10.45	N
F003	Ciampino (CIA)	Roissy Charles de Gaulle (CDG)	300.00	6.30	8.30	Y
F006	Viterbo (LIRV)	Beauvais (BVA)	310.30	7.20	9.30	Y

7.3.4 Soundness, Relevancy, and Optimality of the Result

Suppose, dB and $dB^\sharp$ represent a concrete database and its abstract version respectively. If Q and $Q^\sharp$ are representing the queries on concrete and abstract domain respectively, let ξ and $\xi^\sharp$ be the results of applying Q and $Q^\sharp$ on the dB and $dB^\sharp$ respectively.

Definition 19 *Let $dB^\sharp$ be an abstract table and $Q^\sharp$ be an abstract query. $Q^\sharp$ is sound iff* $\forall dB \in \gamma(dB^\sharp).\ \forall Q \in \gamma(Q^\sharp) : Q(dB) \in \gamma(Q^\sharp(dB^\sharp))$.

Let us denote by the notation $\mathfrak{B} \blacktriangleleft D^{abs}$ the fact that the cooperative system $\mathfrak{B}$ uses the abstract domain D^{abs}, and by $D_1^{abs} \sqsupseteq D_2^{abs}$ the fact that the abstract domain D_1^{abs} is an abstraction of D_2^{abs}.

Definition 20 *Given two abstract domain D_1^{abs} and D_2^{abs}. The domain D_1^{abs} is an abstraction of D_2^{abs} (denoted $D_1^{abs} \sqsupseteq D_2^{abs}$) if $\forall X^\sharp \in D_2^{abs}$, $\forall x \in \gamma_2(X^\sharp)$, $\exists!\ l \in D_1^{abs} :\ \alpha_1(x) = l$, where α_1 is the abstraction function corresponding to D_1^{abs}, and γ_2 is the concretization functions corresponding to D_2^{abs}.*

The extensional query answering system uses zero level abstraction and it returns only the exact answer if available. More relaxation of the query indicates more abstraction used by the cooperative system, returning more cooperative information to the users. Thus whenever we tune the level of abstraction from lower to higher, the system returns monotonically increasing answer set, *i.e.*

$$\text{If } (\mathfrak{B} \blacktriangleleft D_1^{abs}) \text{ and } (\mathfrak{B}' \blacktriangleleft D_2^{abs}) \text{ and } (D_1^{abs} \sqsupseteq D_2^{abs}), \text{ then } \mathfrak{B}(dB, C, Q) \supseteq \mathfrak{B}'(dB, C, Q)$$

When the term "relevancy" comes into the context, the tuning of abstraction must end at a particular point, after which the system returns irrelevant additional information that does not satisfy the constraints in $S(Q)$, where $S(Q)$ is the set of constraints that make explicit the answers relevant to the user. We call the abstraction used at that point as the best correct approximation of the database information. Best correct approximation, thus, depends on the context that defines $S(Q)$. More abstraction beyond the best correct approximation level makes the answer partially relevant as it includes additional irrelevant information *w.r.t.* $S(Q)$.

Example 19 *The cooperative answer R_1 of the query Q_1 in Example 18 is shown in Table 7.3. Let the constraint set be $S(Q)$={flights must be destined in the airport ORY/BVA/CDG/LYS, flight duration must be less than 3 hours}. Observe that the cooperative answer in Table 7.3 is completely relevant as all tuples in the answer satisfy $S(Q)$. If we use an higher level of abstraction, for instance, if the source and destination airports in Table 7.1(a) are abstracted by the nations they belong (in our example, Italy and France), the corresponding cooperative answer R'_1 of the query Q_1 will contain all tuples of the concrete Table 7.1(a) except the tuple with flight no. F002. The answer R'_1 is partially relevant because one tuple among them (flight no. equal to F005) does not satisfy $S(Q)$.*

Our system can work together with a filtering system that can filter out those tuples from the partially relevant results that do not satisfy $S(Q)$, and ranks the results based on the satisfiability of tuples *w.r.t.* $S(Q)$. However, the level of abstraction determines the efficiency of the system with respect to the processing time.

Partial Order between Cooperative Answers

Given a database dB, a query Q, and a context C, the cooperative system may return different cooperative answers to the user under context C depending on the level of abstraction of the abstract domain which is used. We define a partial order between any two cooperative answers: a cooperative answer $R = \mathfrak{B}(dB, C, Q)$ is said to be better than another answer $R' = \mathfrak{B}'(dB, C, Q)$ (denoted $R \leq R'$) if R is more optimal than R' (see definition 18). The partial-ordered set of all cooperative answers for a given query under given context forms a lattice. The bottom most element R_0 determines worst cooperative answer which is completely irrelevant *w.r.t.* $S(Q)$, whereas the top most element R_n is the best cooperative answer which is completely relevant *w.r.t.* $S(Q)$.

Example 20 *The cooperative answer R_1 of the query Q_1 in Example 18 is shown in Table 7.3. When we abstract the airports in Table 7.1(a) by the nations they belong, the corresponding cooperative answer R'_1 of the query Q_1 will contain all tuples of the concrete Table 7.1(a) except the tuple with flight no. F002. Since R'_1 contains one irrelevant tuple (tuple with flight no. F005) w.r.t. $S(Q)$ as depicted in Example 19, after filtering out the irrelevant tuple we get the result R_2 shown in Table 7.4. Observe that R_2 is better than the result R_1 (i.e. $R_2 < R_1$) since R_2 is more optimal than R_1 according to definition 18.*

There exists a wide variety of abstract domains with different expressiveness and complexity that focus on the linear relationship among program variables, such as Interval Polyhedra (31, 32) to infer interval linear relationship, or Difference-Bound

Table 7.4: R_2: Cooperative result of Q_1 while using more abstraction

flight no.	source	destination	cost ($)	start-time	reach-time	availability
F001	Fiumicino (FCO)	Orly (ORY)	210.57	8.05	10.45	N
F003	Ciampino (CIA)	Roissy Charles de Gaulle (CDG)	300.00	6.30	8.30	Y
F004	Urbe (LIRU)	Lyon-Saint Exupéry (LYS)	128.28	22.05	23.40	N
F006	Viterbo (LIRV)	Beauvais (BVA)	310.30	7.20	9.30	Y

Matrices (145) representing the constraints of the form $x - y \leq c$ and $\pm x \leq c$ where x, y are program variables and c is constant. We can exploit such abstract domains by focusing on the set of constraint $S(Q)$ that make the answers relevant to the users.

Our new cooperative query answering scheme needs to be further refined, in particular, by investigating its application to more sophisticated scenarios on different abstract domains, in order to properly address the tradeoff between accuracy and efficiency.

7.4 Intensional Query Answering

Recently, there have been several independent efforts aimed at enhancing interfaces to conventional databases with yet another feature, which we shall refer to as the ability to compute intensional answers (148). Instead of always responding to a query with a substitution for the variables in the query, it is sometimes more appropriate to provide the users with an intensional answer. An intensional answer is a complement of the conventional answer, comprising either a terse description of the answer or various useful statements that concern the answer. It teaches users about the structure of the databases and of the domain and help to clear up the misconception. For instance, consider a query asking "Which students are enrolled in the university?". The traditional extensional answer provides the list of all students in the database, which is really misleading. Thus, the answer "All students are enrolled." is better and more informative to the users.

Several works on intensional query answering for relational databases (198), deductive databases (159), XML databases (141) etc have already been done. There might be a very interesting relationship between cooperative query answering approach and the intensional query answering one, as the later can be seen as an application of a suitable abstraction to the result of a concrete query. This could be an interesting direction that deserves to be further studied.

Chapter 8

Refinement of Abstract Program Slicing techniques

[Part of this chapter is already published in (44, 88)]

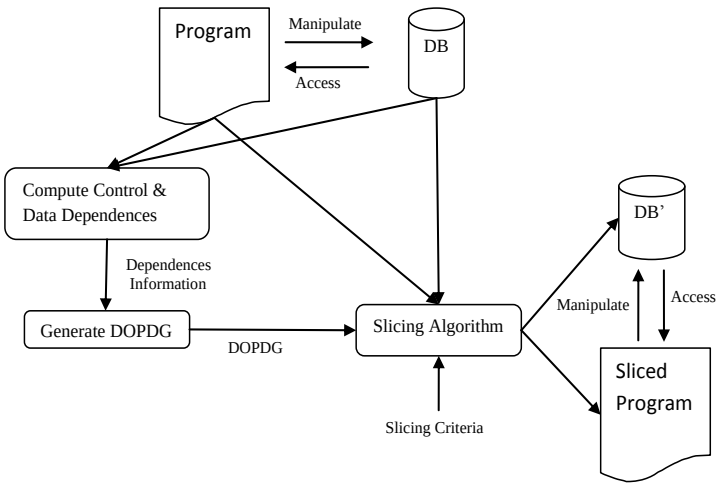
Program slicing is a well-known decomposition technique that extracts from programs the statements which are relevant to a given behavior. It is a fundamental operation for addressing many software-engineering problems, including program understanding, debugging, maintenance, testing, parallelization, integration, software measurement, etc. See, for instance, (37, 62, 63, 75, 109, 120, 123, 127, 149, 156, 160). The notion of program slice was originally introduced by Mark Weiser (193) who defines a static program slice as an executable subset of program statements that preserves the original program's behavior at a particular program point for a subset of program variables for all program inputs. Therefore, the static slicing criterion is denoted by $\langle p, v\rangle$ where p is the program point and v is the variable of interest. This is not restrictive, as it can easily be extended to slicing with respect to a set of variables V, formed from the union of the slices on each variable in V. In contrast, in dynamic slicing (119), programmers are more interested in a slice that preserves the program's behavior for a specific program input rather than for all program inputs: the dependences that occur in a specific execution of the program are taken into account. Therefore, the slicing criterion for dynamic slicing is denoted by $\langle p, v, i\rangle$, where i represents the input sequence of interest.

All the slicing techniques on imperative languages make use (explicitly or implicitly) of the notion of Program Dependence Graph (PDG) (58, 124, 156). Different forms of PDG representation have been proposed, depending on the intended applications

(29, 95). Over the last 25 years, many PDG-based program slicing techniques have been proposed (2, 68, 96, 150, 160, 170, 177). In general, a PDG makes explicit both the data and control dependences for each operation in a program. Data dependences have been used to represent only the relevant data flow relationship of a program, while control dependences are derived from the actual control flow graph and represent only the essential control flow relationships of a program.

The presence of SQL operations such as SELECT, INSERT, UPDATE or DELETE in data-intensive applications that access or manipulate databases, requires the extension of traditional Program Dependence Graphs (PDGs) into Database-Oriented Program Dependence Graphs (DOPDGs), where two additional types of dependences, called Program-Database (PD) Dependences and Database-Database (DD) Dependences, need to be considered (194). A PD-Dependence arises between a SQL statement and an imperative statement where either the database state defined by SQL statement is used by the imperative statement or the database state defined by imperative statement is used by the SQL statement. A DD-Dependence arises between two SQL statements where the database state defined by one SQL statement is used by the other SQL statement. The data dependences between imperative statements are similar as in case of the traditional PDGs. Figure 8.1 depicts a pictorial view of the DOPDG-based slicing approach for the programs embedding SQL statements. It is worthwhile to note that the PDG/DOPDG-based slicing is somewhat restricted: a slice must be taken *w.r.t.* variables that are defined or used at that program point.

Figure 8.1: DOPDG-based slicing for programs embedding SQL statements

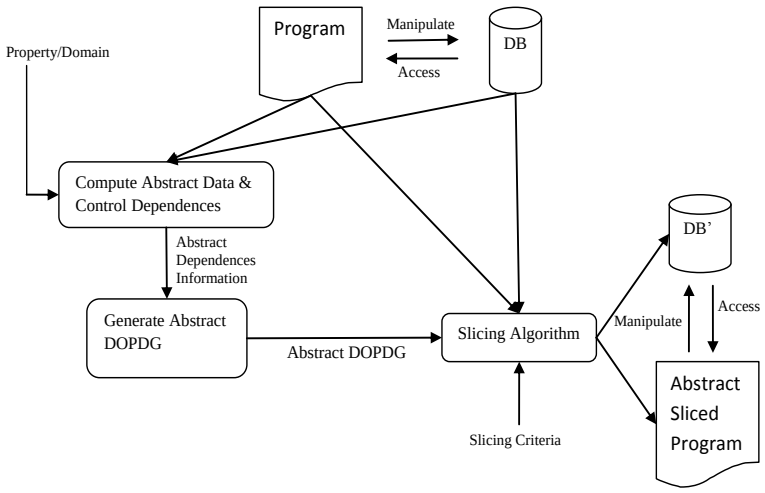


Program slicing can be defined in concrete as well as in abstract domain, where in

the former case we consider exact values of the program variables, while in the latter case we consider some properties instead of their exact values. These properties are represented as abstract domains of the variable domains in the context of Abstract Interpretation. The notion of Abstract Program Slicing was first introduced by Hong, Lee and Sokolsky (172). Some recent works includes semantics-based abstract program slicing (139, 140, 200) and property-driven program slicing (18). Abstract slicing helps in finding the statements affecting some particular properties of the variables of interest. For instance, suppose a program variable at some point of execution is not resulting the correct properties (represented by the domain of intervals, say) as expected. In such case, abstract slicing can effectively be able to identify the statements responsible for this error. Recently, Mastroeni and Nicolić (139) extended the theoretical framework of slicing by Binkley (22) to an abstract domain, in order to define abstract slicing and to represent and compare different forms of slicing in an abstract domain. Slicing criterion in an abstract domain, thus, includes observable properties of the variables of interest as well. For instance, static and dynamic abstract slicing criteria are denoted by $\langle p, v, \rho \rangle$ and $\langle p, v, i, \rho \rangle$ respectively, where ρ is an observable property of v.

A pictorial view of abstract slicing of the programs embedding SQL statements is depicted in Figure 8.2. Observe that the sliced database dB' on which slices perform their computations, in both concrete and abstract slicing, is a part of the original database dB.

Figure 8.2: Abstract DOPDG-based slicing for programs embedding SQL statements



In traditional dependence graphs, the notion of dependences between program

statements depends only on the syntactic presence of one variable in the definition of another variable or in a conditional expression. Therefore, the definition of slices at semantic level creates a gap between slicing and dependences. A first attempt to fill this gap partially is done by Mastroeni and Zanardini by introducing the notion of semantic data dependences (140).

This chapter provides two main contributions:

(i) The first one is the introduction of the notion of semantic relevancy of statements *w.r.t.* a property. It determines whether an imperative or a SQL statement is relevant *w.r.t.* a property of interest, and is computed over all concrete (or abstract) states possibly reaching the statement. For instance, consider the following code fragment: {(1) $x = input$; (2) $x = x + 2$; (3) *print* x;}. If we consider an abstract domain of parity represented by $PAR = \{\top, ODD, EVEN, \bot\}$, we see that the variable x at program point 1 may have any parity from the set $\{ODD, EVEN\}$, and the execution of the statement at program point 2 does not change the parity of x at all. Therefore, the statement at 2 is semantically irrelevant *w.r.t.* PAR. By disregarding all the nodes that correspond to irrelevant statements *w.r.t.* concrete (or abstract) property from a syntactic PDG/DOPDG, we obtain a more precise semantics-based (abstract) PDG/DOPDG. Observe that the combined effort of semantic relevancy of statements with the expression-level semantic data dependences introduced by Mastroeni and Zanardini (140) guarantees a more precise semantics-based (abstract) PDG/DOPDG.

(ii) The second contribution of this chapter is the refinement of the semantics-based PDG/DOPDG obtained so far by applying the notion of conditional dependences proposed by Sukumaran et al. (182). This allows us to transform a PDG or a DOPDG into Dependence Condition Graph (DCG) or Database-Oriented Dependence Condition Graph (DODCG) that enables to identify the conditions for dependences between program points. We lift the semantics of DCG/DODCG from concrete domain to an abstract domain of interest. The satisfiability of the conditions in DCG/DODCG by (abstract) execution traces helps to remove semantically unrealizable dependences from them, yielding to refined semantics-based (abstract) DCG/DODCG.

These two contributions in combination with semantic data dependences (140) lead to a semantics-based abstract program slicing algorithm that strictly improves with respect to the literature. The algorithm constructs a semantics-based abstract DCG/DODCG

from a given program by combining these three notions: (*i*) semantic relevancy of statements, (*ii*) semantic data dependences at expression level (140), and (*iii*) conditional dependences (182). Slicing based on this semantics-based DCG/DODCG, both in concrete and abstract domains, yields to more precise slices.

The structure of this chapter is organized as follows: In section 8.1, we discuss the related works in the literature. Section 8.2 recalls some basic background. Section 8.3 introduces the notion of semantic relevancy of imperative statements *w.r.t.* concrete/abstract property. In section 8.4, we formalize an algorithm to construct semantics-based abstract PDG from a given imperative program. In section 8.5, we lift the semantics of DCGs from the concrete domain to an abstract domain of interest, and we propose a refinement of syntactic DCCs into semantics-based abstract DCGs. The proposed abstract slicing algorithm for imperative programs is formalized in section 8.6. Section 8.7 illustrates the proposed slicing technique with an example. In section 8.8, we prove the soundness and provide an overall complexity analysis of the proposal. Section 8.9 discusses the strength and weakness of our proposal. In section 8.10, we extend our proposed slicing refinement technique to the context of data-intensive applications accessing or manipulating databases.

8.1 Related Work

The original static slicing algorithm by Mark Weiser (193) is expressed as a sequence of data-flow analysis problems and the influence of predicates on statement execution, while Korel and Lasky (119) extended it into the dynamic context and proposed an iterative dynamic slicing algorithm based on dynamic data flow and control influence. Both Weiser's and Korel-Laski's algorithms produce executable form (189) of slices that are easier to fit into a theoretical framework by Binkley (22) that provides a relationship between different well-known forms of slicing by exploiting the requirements of syntactic and semantic equivalence between the slice and the original program. Over the last 25 years, several works on program slicing based on the Program Dependence Graph (PDG) representation have been done (2, 68, 96, 150, 160, 170, 177). However, different forms of PDG representations have been proposed, depending on the intended applications (29, 95).

The notion of Abstract Program Slicing was first introduced by Hong, Lee and Sokolsky (172). Some recent works includes semantics-based abstract program slicing (139, 140, 200) and property-driven program slicing (18). Mastroeni and Nicolić (139) extended the theoretical framework of slicing proposed by Binkley (22) to an abstract

domain in order to define abstract slicing, and to represent and compare different forms of slicing in abstract domain.

Mastroeni and Zanardini (140) first introduced the notion of semantic data dependences which fills up the existing gap between syntax and semantics. The semantic data dependences which is computed for all expressions in the program over the states possibly reaching the associated program points, help in obtaining more precise semantics-based PDGs by removing some false dependences from the traditional syntactic PDGs.

Sukumaran et al. (182) presented a refinement of the traditional PDGs of programs, called Dependence Condition Graph (DCG), based on the notion of conditional dependences. A DCG is built from a PDG by annotating each edge e of the PDG with conditional information $e^b = \langle e^R, e^A \rangle$ under which a particular dependence actually arises in a program execution. The first part e^R is referred to as *Reach Sequences* which represents the conditions that should be true for an execution to ensure that the target $e.tgt$ of e must be executed once the source $e.src$ is executed for a control edge e, and the target $e.tgt$ is reached from the source $e.src$ for a data edge e. The component e^A is referred to as *Avoid Sequences* which is only relevant for data edges (it is $\emptyset$ for control edges) and captures the possible conditions under which the assignment at $e.src$ can get overwritten before it reaches $e.tgt$. So, the conditions in e^A must not hold for an execution to ensure that the variable defined at $e.src$ must used at $e.tgt$.

All the slicing techniques mentioned above refer to imperative languages. They do not take into account the presence of the additional forms of states associated with programs, such as reading from the standard input stream or accessing/manipulating data from external databases. Sivagurunathan et.al. (180) first addressed this and introduced pseudo variables into the program to make the hidden I/O state accessible to the slicer.

Tan and Ling (183) extended the notion of slicing to the context where various database operations are present in the programs. They followed a similar solution and introduced a set of implicit variables to capture the influence among I/O statements operating on database records.

Willmor et.al. (194) introduced a variant of the program dependence graph, known as Database-Oriented Program Dependence Graph (DOPDG), by considering two additional types of data dependences: *Program-Database Dependences (PD-Dependences)* and *Database-Database Dependences (DD-Dependences)*. They applied Condition-Action rules introduced by Baralis and Widom (9) that determine when an action of one rule can affect the condition of another rule.

In the presence of embedded DML statements, Cleve (38) proposed to construct System Dependence Graph (SDG) to represent the control and the dataflow of both the host language and the embedded language. The dependence pseudo-instructions (DIRECT-MAP and INDIRECT-MAP) are used (instead of the original code) to construct the SDG nodes and the data dependence edges corresponding to embedded DML fragments. Once the full SDG has been built, program slices can be computed using the usual algorithm.

Recently, Saha et al. (169) proposed a new key-based dynamic slicing algorithm and two differencing techniques that use the underlying program semantics to localize faults in the data-centric programs that use embedded database specific statements to perform operations on in-memory and persistent data.

8.2 Preliminaries

In this section, we recall some basic backgrounds.

Static Single Assignment (SSA). The SSA form (52) of a program is a semantically equivalent version of the program where each variable has a unique (syntactic) assignment. The SSA form introduces special ϕ-assignments at join nodes of the program where assignments to the same variable along multiple program paths may converge. Each assignment to a variable is given a unique name, and all of the uses reached by that assignment are renamed to match the assignment's new name. Figure 8.3(a) and 8.3(b) depict a program P and its SSA form P_{ssa} respectively. In the rest of this

Figure 8.3: A program and its SSA form

(a) *Program P*

1.	start
2.	$x = input;$
3.	$y = input;$
4.	$w = input;$
5.	$z = 4;$
6.	$if(x == y)\{$
7.	$\quad z = 2 \times (x + y) - 4 \times (x) + 4;$
8.	$\quad x = x + 1;\}$
9.	$else\ \{\ z = x + w;$
10.	$\quad x = x + 2;\}$
11.	$print(x, z);$
12.	stop

(b) P_{ssa}: *SSA form of P*

1.	start
2.	$x_1 = input;$
3.	$y_1 = input;$
4.	$w = input;$
5.	$z_1 = 4;$
6.	$if(x_1 == y_1)\{$
7.	$\quad z_2 = 2 \times (x_1 + y_1) - 4 \times (x_1) + 4;$
8.	$\quad x_2 = x_1 + 1;\}$
9.	$else\ \{\ z_3 = x_1 + w;$
10.	$\quad x_3 = x_1 + 2;\}$
ϕ	$(x_4, z_4) = f((x_2, z_2), (x_3, z_3));$
11.	$print(x_4, z_4);$
12.	stop

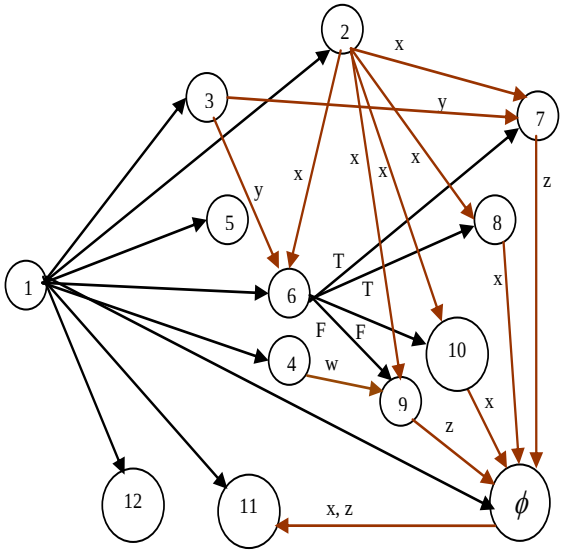
chapter, we use SSA form of programs due to its compact representation and easy to compute DCG annotations as well as to define its semantics. The SSA form also helps in improving the flow-sensitive analysis of any program (94).

Program Dependence Graph. Program Dependence Graph (PDG) (58, 124, 156, 185) for a program is a directed graph with vertices denoting program components (*start*, *stop*, *skip*, *assignment*, *conditional* or *repetitive* statements) and edges denoting dependences between components. An edge represents either *control dependence* or *data dependence*. The sub-graph of the PDG induced by the control dependence edges is called control dependence graph (CDG) and the sub-graph induced by the data dependence edges is called data dependence graph (DDG). The source node of a CDG edge corresponds to either *start* or *conditional* or *repetitive* statement. A CDG edge e whose source node $e.src$ corresponds to *start* statement is denoted by an unlabeled edge $e = e.src \rightarrow e.tgt$, meaning that the condition represented by $e.src$ is implicitly *true*, *i.e.*, during an execution once $e.src$ executed its target $e.tgt$ will eventually be executed. If the source node $e.src$ of any CDG edge e corresponds to *conditional* or *repetitive* statement it is denoted by a labeled edge $e = e.src \xrightarrow{lab} e.tgt$ where $lab \in \{true, false\}$, meaning that $e.tgt$ is lab-control dependent on $e.src$, *i.e.*, whenever the condition represented by $e.src$ is evaluated and its value matches the label lab, then its target node represented by $e.tgt$ will be executed, if the program terminates. A DDG edge is denoted by $e = e.src \xrightarrow{x} e.tgt$, representing that the target node $e.tgt$ is data dependent on the source node $e.src$ for a variable x. The PDG representation of the program P_{ssa} (Figure 8.3(b)) is depicted in Figure 8.4.

Database-Oriented Program Dependence Graph (DOPDG) Willmor et al. (194) introduced a variant of the program dependence graph, known as Database-Oriented Program Dependence Graph (DOPDG), by considering the notion of data dependences in the presence of database states for the programs embedding SQL statements. They defined two additional types of data dependences: *Program-Database Dependences (PD-Dependences)* and *Database-Database Dependences (DD-Dependences)*.

A PD-Dependence arises between a SQL statement and an imperative statement where either the database state defined by SQL statement is used by the imperative statement or the data defined by imperative statement is used by the SQL statement. A DD-Dependence arises between two SQL statements where the database states defined by one SQL statement is used by the other SQL statement. The data dependences between imperative statements and the control dependences are similar as in

Figure 8.4: G_{pdg}: PDG of P_{ssa}



the case of traditional PDGs. The subgraph induced by PD-Dependences is called Program-Database Dependence Graph (PDDG), whereas the subgraph induced by DD-Dependences is called Database-Database Dependence Graph (DDDG).

PDG/DOPDG-based Slicing. The results of the dependence graphs discussed so far have an impact on different forms of static slicing: backward slicing (193), forward slicing (13), and chopping (109, 165). The backward slice with respect to variable v at program point p consists of those program points that affect v directly or indirectly. Forward slicing is the dual of backward slicing. The forward slice with respect to variable v at program point p consists of those program points that are affected by v. Chopping is a combination of both backward and forward slicing. A slicing criterion for chopping is represented by a pair $\langle s, t \rangle$ where s and t denote the source and the sink respectively. In particular, chopping of a program *w.r.t.* $\langle s, t \rangle$ identifies a subset of its statements that account for all influences of source s on sink t.

The slicing based on PDGs/DOPDGs is slightly restrictive in the sense that the dependence graph permits slicing of a program with respect to program point p and a variable v that is defined or used at p, rather than *w.r.t.* arbitrary variable at p. PDG/DOPDG-based backward program slicing is performed by walking the graph backwards from the node of interest in linear time (156). The walk terminates at either entry node or already visited node. In case of PDG/DOPDG-based forward slicing

technique, similarly, we traverse the graph in forward direction from the node of interest. We can use the standard notion of *chop* of a program with respect to two nodes s and t in slicing technique (109, 165): $chop(s,t)$ is defined as the set of inter-procedurally valid PDG/DOPDG paths from s to t where s, t are real program nodes, in contrast to ϕ nodes in SSA form of the program. We define it as follows (182): $AC(s,t)$ is defined to be true if there exists at least one execution ψ that satisfies a valid PDG/DOPDG path η between s and t *i.e.* $AC(s,t) \triangleq \exists\psi : AC(s,t,\psi)$ and $AC(s,t,\psi) \triangleq \exists\eta \in chop(s,t) : \psi \vdash \eta$. The $\neg AC(s,t)$ implies that $\forall\psi$ and $\forall\eta \in chop(s,t) : \psi \nvdash \eta$, that is, $chop(s,t)$ is empty.

In PDG/DOPDG-based dynamic slicing (2, 68, 150) *w.r.t.* a variable for a given execution history, a projection of the PDG/DOPDG *w.r.t.* the nodes that occur in the execution history is obtained, and then static slicing algorithm on the projected dependence graph is applied to find the desired dynamic slice. Agrawal and Horgan (2) also introduced a variant of it based on the graph-reachability framework, called Dynamic Dependence Graph and Reduced Dynamic Dependence Graph, to obtain more precise dynamic slice.

In general, there exist many different slices for a given program *w.r.t.* a slicing criterion where one can be more precise than the other, as defined in Definition 21.

Definition 21 (Precision of Slicing) *Given two programs P' and P'' such that both P' and P'' are slice of P w.r.t. a criterion C. P' is more precise than P'' if P' ($\neq P''$) is a slice of P'' w.r.t. C.*

Abstract Interpretation and Program Slicing. A first attempt to combine Abstract Interpretation with program slicing is done by Mastroeni and Zanardini in (140). In traditional PDGs, the notion of dependences between statements is based on syntactic presence of a variable in the definition of another variable or in a conditional expression. Therefore, the definition of slices at semantic level creates a gap between slicing and dependences. Mastroeni and Zanardini (140) introduced the notion of semantic data dependences which fills up the existing gap between syntax and semantics. For instance, although the expression "$e = x^2 + 4w \bmod 2 + z$" syntactically depends on w, but semantically there is no dependence as the evaluation of "$4w \bmod 2$" is always zero. This can also be lifted to an abstract setting where dependences are computed with respect to some specific properties of interest rather than concrete values. For instance, if we consider the abstract domain $SIGN = \{\top, pos, neg, \bot\}$, the expression e does not semantically depend on x *w.r.t.* SIGN, as the abstract evaluation of x^2 always yields to *pos* for all atomic values of $x \in \{pos, neg\}$. This is the basis to design abstract semantics-based slicing algorithms aimed at identifying the part of the programs which

is relevant with respect to a property (not necessarily the exact values) of the variables at a given program point.

Abstract Semantics: Expressions and Statements. Consider the IMP language (195). The statements of a program P act on a set of constants $\mathbb{C} = const(P)$ and a set of variables $VAR = var(P)$. A program variable $x \in VAR$ takes its values from the semantic domain $\mathbb{V} = Z_{\mho}$ where, $\mho$ represents an undefined or uninitialized value and Z is the set of integers. The arithmetic expressions $e \in Aexp$ and boolean expressions $b \in Bexp$ are defined by standard operators on constants and variables. The set of states Σ consists of functions $\sigma : VAR \to \mathbb{V}$ which map the variables to their values. For the program with k variables $x_1, \ldots, x_k$, the state is denoted by k-tuples: $\sigma = \langle v_1, \ldots, v_k \rangle$, where $v_i \in \mathbb{V}, i = 1, ..., k$ and hence, the set of states $\Sigma = (\mathbb{V})^k$. Given a state $\sigma \in \Sigma$, $v \in \mathbb{V}$, and $x \in VAR$: $\sigma[x \leftarrow v]$ denotes a state obtained from σ by replacing its contents in x by v, *i.e.* define

$$\sigma[x \leftarrow v](y) = \begin{cases} v \;\; if \; x = y \\ \sigma(y) \;\; if \; x \neq y \end{cases}$$

The semantics of arithmetic expression $e \in Aexp$ over the state σ is denoted by $E[\![e]\!](\sigma)$ where, the function E is of the type $Aexp \to (\sigma \to \mathbb{V})$. Similarly, $B[\![b]\!](\sigma)$ denotes the semantics of boolean expression $b \in Bexp$ over the state σ of type $Bexp \to (\sigma \to T)$ where $T \in \{true, false\}$.

The Semantics of statement s is defined as a partial function on states and is denoted by $S[\![s]\!](\sigma)$ which defines the effect of executing s in σ.

Consider an abstract domain ρ on values. The set of abstract states is denoted by $\Sigma^\rho \triangleq \rho(\wp(\mathbb{V}))^k$. The abstract semantics $E[\![e]\!]^\rho(\epsilon)$ of expression e is defined as the best correct approximation of $E[\![e]\!]$: let $\sigma = \langle v_1, \ldots, v_k \rangle \in \Sigma$ and $\epsilon = \langle \rho(v_1), \ldots, \rho(v_k) \rangle \in \Sigma^\rho$: $E[\![e]\!]^\rho(\epsilon) = \rho(\{E[\![e]\!](\langle u_1, \ldots, u_k \rangle) \mid \forall i.\, u_i \in \rho(v_i)\})$.

The syntax and semantics of programs embedding SQL statements, in both concrete and abstract domains, are described in chapter 3.

8.3 Semantic Relevancy of Statements

In this section, we introduce the notion of semantic relevancy of imperative statements in both concrete and abstract domains. The motive behind this is to determine whether the execution of a statement affects a concrete/abstract property of the variables of interest.

Definition 22 (Concrete Semantic Relevancy) *Given a program P and a concrete property ω on states, the statement s at program point p in P is semantically relevant w.r.t. ω if:*

$$\exists \sigma \in \Sigma_p : \; S[\![s]\!](\sigma) = \sigma' \; \wedge \; \omega(\sigma) \neq \omega(\sigma')$$

where Σ_p are the set of states that can possibly reach the program point p.

In other words, the statement s at program point p is semantically irrelevant *w.r.t.* a concrete property ω if the execution of s over any state $\sigma \in \Sigma_p$ yields to a state that is equivalent to σ *w.r.t.* ω.

In particular, whenever ω distributes over program variables, we may use the Definition 23.

Definition 23 (Concrete Semantic Relevancy for a Set of Variables) *Given a program P and a concrete property ω that distributes over program variables, the statement s at program point p in P is semantically relevant w.r.t. ω for a subset of variables U if*

$$\exists \sigma = \langle \sigma(v_1), \sigma(v_2), \ldots, \sigma(v_k) \rangle \in \Sigma_p \text{ and } \exists v_i \in U \text{ such that } \omega(\pi_i(S[\![s]\!]\sigma)) \neq \omega(\pi_i(\sigma))$$

where $\pi_i(\langle \sigma(v_1), \sigma(v_2), \ldots, \sigma(v_k) \rangle) = \sigma(v_i)$.

Example 21 *Consider the concrete property $\omega_{(x \neq 4)} : \Sigma \rightarrow \{true, false\}$, which is true in state σ iff $\sigma(x) \neq 4$. The statement $x = y + 1$ is semantically relevant w.r.t. $\omega_{(x \neq 4)}$ only if $\exists \sigma \in \Sigma_p$ such that $\sigma(y) = 3$ and $\sigma(x) \neq 4$.*

Example 22 *Consider the program P_{ssa} in Figure 8.3(b). The statement at program point 7 is semantically irrelevant w.r.t. all concrete properties, as the execution of this statement over any state possibly reaching program point 7 does not change the state. That is, $\forall \sigma \in \Sigma_7$ where $\sigma(x) = \sigma(y)$ and $\sigma(z) = 4$, the execution of the statement over σ does not modify the value of z.*

Example 23 *Consider the program P_{ssa} in Figure 8.3(b) and the property defined by $\omega \triangleq \#\{x \in VAR : [\![x]\!]\sigma \in EVEN\} = \#\{x \in VAR : [\![x]\!]\sigma \in ODD\}$ where VAR is the set of program variables, $\sigma \in \Sigma$, EVEN represents $\{y \in Z \; : \; y \text{ is even}\}$, ODD represents $\{y \in Z \; : \; y \text{ is odd}\}$, Z is the set of integers, and # denotes the cardinality of set. The statement $s \triangleq x = x + 1$ at program point 8 is relevant w.r.t. ω, since the execution of s over any state $\sigma \in \Sigma_8$ with equal number of variable values belonging to both ODD and EVEN sets, yields to a state σ' where the value of x moves from one set to another. Observe that the statement at program point 10, on the other hand, is irrelevant w.r.t. ω.*

We can lift the notion of semantic relevancy in an abstract domain of interest. The abstract semantic relevancy can be defined in relational as well as non-relational abstract domains.

Definition 24 (Abstract Semantic Relevancy) *Let P be a program and ρ be a closure operator over $\wp(\mathbb{V})$, the statement s at program point p in P is semantically relevant w.r.t. abstract property ρ if*

$$\exists \epsilon \in \Sigma_p^\rho : \; S[\![s]\!]^\rho(\epsilon) \neq \epsilon$$

where Σ_p^ρ are the set of abstract states that can possibly reach the program point p.

In other words, the statement s at program point p is semantically irrelevant *w.r.t.* an abstract property ρ if no changes take place in the abstract states $\epsilon \in \Sigma_p^\rho$, when s is executed over ϵ.

Example 24 *Consider the statement $s \triangleq x = x + 2$ at program point 10 of the program P_{ssa} depicted in Figure 8.3(b). The statement s is semantically relevant w.r.t. $\rho = SIGN$, because $\exists \epsilon = \langle \rho(x), \rho(y), \rho(w), \rho(z) \rangle = \langle -, +, +, + \rangle \in \Sigma_{10}^{SIGN}$: $S[\![s]\!]^\rho(\langle -, +, +, + \rangle) = \langle \top, +, +, + \rangle$. On the other hand, if we consider the abstract domain $\rho = PAR$, we see that s is semantically irrelevant w.r.t. PAR because $\forall \epsilon \in \Sigma_{10}^{PAR}$: $S[\![s]\!]^\rho(\epsilon)$ does not change the parity of x.*

Definition 24 is not parametric on variables. Below we provide a parametric definition for the abstract statements relevancy. This definition may be useful, combined with independence analysis, to further refine the slicing when focussing just on a proper subset of program variables in the slicing criteria.

Definition 25 (Abstract Semantic Relevancy for a Set of Variables) *Let P be a program and ρ be a closure operator over $\wp(\mathbb{V})$, the statement s at program point p in P is semantically relevant w.r.t. abstract property ρ for a subset of variables U if*

$$\exists \epsilon = \langle \rho(v_1), \rho(v_2), \ldots, \rho(v_k) \rangle \in \Sigma_p^\rho \text{ and } \exists v_i \in U \text{ such that } \pi_i(S[\![s]\!]^\rho(\epsilon)) \neq \pi_i(\epsilon)$$

where $\pi_i(\langle \rho(v_1), \rho(v_2), \ldots, \rho(v_k) \rangle) = \rho(v_i)$.

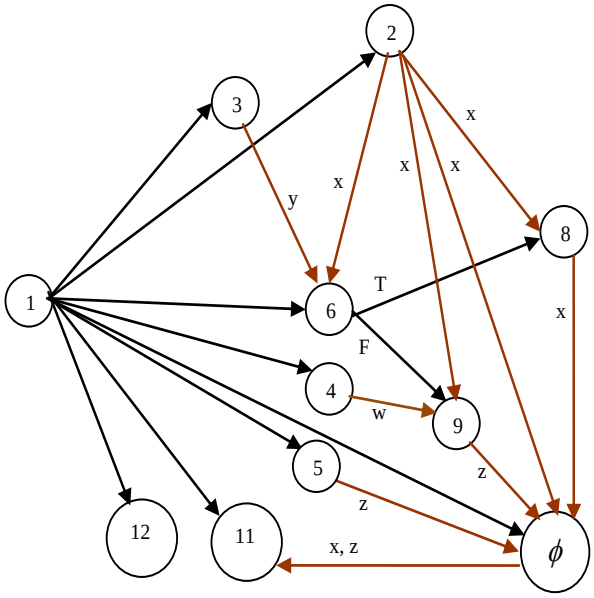
Intuitively, the semantic relevancy of statements just says that an assertion remains true over the states possibly reaching the corresponding program points. Observe that if we consider the predicate ω as an abstraction on the concrete states, Definition 24 is just a rephrasing of Definition 22.

The well-known collecting semantics (also known as static semantics) (47) of programs obtains the set of states possibly reaching each program point in the program.

The relevancy computation of a statement s at p *w.r.t.* a concrete/abstract property is, thus, performed by simply checking whether the execution of s changing the property of any of the states at p generated from the collecting semantics. Note that the only statements that can affect the property of states are the "*assignment*" statements of the form $v = e$, where v is a variable and e is an arithmetic expression.

The notion of statements relevancy have many interesting application areas. For instance, if we are analyzing a speed control engine and we are just interested on the portion of program that may lead to a totally unexpected negative value of a speed variable (yielding to a crash-prone situation), then every statement that does not affect neither directly nor indirectly its sign can immediately be disregarded.

Figure 8.5: G^r_{pdg} : PDG after relevancy computation of P_{ssa} *w.r.t.* PAR



Given a program and its syntactic PDG, we can obtain a more precise semantics-based (abstract) PDG by disregarding from the syntactic PDG all the nodes that corresponds to irrelevant statements *w.r.t.* a concrete/abstract property of interest. For instance, Figure 8.5 depicts the semantics-based abstract PDG G^r_{pdg} which is obtained by disregarding from the graph G_{pdg} (Figure 8.4) the two nodes corresponding to the irrelevant statements 7 and 10 *w.r.t.* PAR.

It is worthwhile to mention that the computation above can be optimized in particular cases by applying slicing on the syntactic PDG first, and then, computing statements relevancy and semantic data dependences on this sliced program.

8.3.1 Semantic Relevancy of Blocks

In the previous section, we defined the semantic relevancy of statements *w.r.t.* concrete/abstract property. Now we define semantic relevancy of blocks *w.r.t.* concrete/abstract property, where by block we mean a set of statements $S = \{s_1, \ldots, s_n\}$. Let us denote a block of a set of statements S by blk_S.

Definition 26 *blk_S is semantically irrelevant w.r.t. a concrete property ω (or abstract property ρ) if*

$$\forall s_i \in blk_S,\ i \in [1..n] :\ s_i \text{ is semantically irrelevant w.r.t. } \omega \text{ (or } \rho)$$

Definition 27 *blk_S is partially relevant w.r.t. a concrete property ω (or abstract property ρ) if*

$$\exists s_i, s_j \in blk_S\ \wedge i,\ j \in [1..n] \wedge i \neq j : s_i \text{ is semantically relevant w.r.t. } \omega \text{ (or } \rho) \text{ and } s_j \text{ is semantically irrelevant w.r.t. } \omega \text{ (or } \rho)$$

Definition 28 *blk_S is completely relevant w.r.t. a concrete property ω (or abstract property ρ) if*

$$\forall s_i \in blk_S,\ i \in [1..n] :\ s_i \text{ is semantically relevant w.r.t. } \omega \text{ (or } \rho)$$

Observe that we can convert any partially relevant block *w.r.t.* ω (or ρ) into a corresponding completely relevant block by removing all the irrelevant statements *w.r.t.* ω (or ρ) present in that block.

8.3.2 Treating Relevancy of Control Statements

In this section, we consider the conditional statements "if", "if-*else*" and the repetitive statement "*while*", and we define their relevancy *w.r.t.* a concrete property ω (or abstract property ρ).

The "if" statement can be expressed as

$$if(cond)\ then\ blk_{if}$$

The "if-*else*" statement can be expressed as

$$if(cond)\ then\ blk_{if}\ else\ blk_{else}$$

Similarly, the repetitive statement "*while*" can be expressed as

$$while(cond)\ blk_{while}$$

The semantic relevancy of the "if" and "*while*" statements solely depend on the relevancy of their corresponding blocks blk_{if} and blk_{while} respectively. As an example, the semantic irrelevancy of a repetitive statement "*while*" *w.r.t.* PAR is illustrated in Example 25.

Example 25 *Consider a program depicted in Figure 8.6(a) and the abstract domain PAR. Its traditional syntactic PDG is shown in Figure 8.6(b). Consider the repetitive statement "while" and the corresponding block blk_{while} that contains two statements at program points 5 and 6. At program point 2 and 3, the parity of x and y are odd and even respectively. These properties of x and y remain unchanged during the execution of the while loop. Statements 5 and 6, therefore, are not semantically relevant w.r.t. PAR. In fact, at 5 the parity of x_2 is not affected as its value is not changed by the assignment, and the parity of x_3 is preserved by adding an even value like the constant 2. Similar in case of statement at 6. Therefore, the while-block blk_{while} is irrelevant w.r.t. PAR since all the statements (statements 5 and 6) in the block are irrelevant, and we replace statements 5 and 6 by a skip statement q, yielding to a more precise PDG depicted in Figure 8.6(c).*

The relevancy of "if-*else*" statement, *i.e.*, "$if(cond)$ *then* blk_{if} *else* blk_{else}" *w.r.t.* a concrete property ω (or abstract property ρ) can be defined as follows:

1. An "if-*else*" statement "$if(cond)$ *then* blk_{if} *else* blk_{else}" is semantically irrelevant *w.r.t.* ω (or ρ) if both blk_{if} and blk_{else} are semantically irrelevant *w.r.t.* ω (or ρ).

2. If any or both of the blocks blk_{if} and blk_{else} in "if-*else*" statement are partially relevant *w.r.t.* ω (or ρ), we say that the "if-*else*" statement is partially relevant *w.r.t.* ω (or ρ). The partial relevancy of the "if-*else*" statement can be converted into complete relevancy by converting the corresponding partial relevant blocks into completely relevant blocks.

3. If blk_{else} in "if-*else*" statement is semantically irrelevant *w.r.t.* ω (or ρ), then the relevancy of the "if-*else*" statement is equivalent to the relevancy of the statement "$if(cond)$ *then* blk_{if}" *w.r.t.* ω (or ρ).

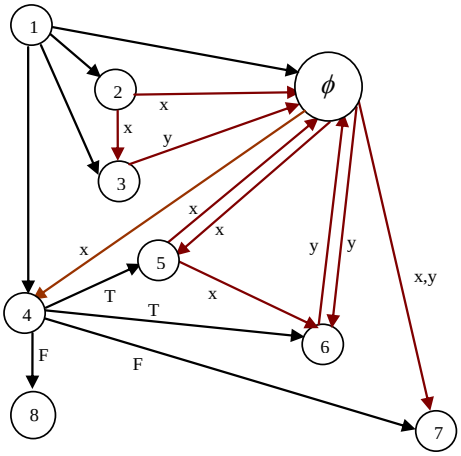
4. If blk_{if} in "if-*else*" statement is semantically irrelevant *w.r.t.* ω (or ρ), then the relevancy of the "if-*else*" statement is equivalent to the relevancy of the statement "$if(cond)$ *then skip else* blk_{else}" *w.r.t.* ω (or ρ).

Figure 8.6: Treating "*while*" block

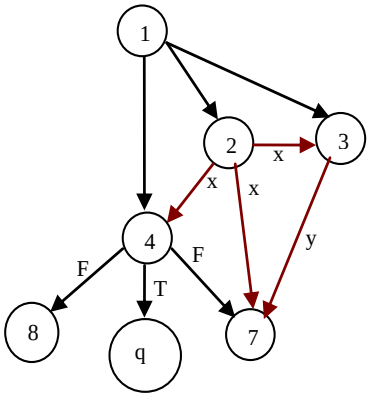
(a) Program P_{while}

1.	start
2.	$x_1 = 1;$
3.	$y_1 = 2 \times x_1;$
	while(
ϕ	$(x_2, y_2) = f((x_1, y_1), (x_3, y_3))$
4.	$x_2 < 20$){
5.	$x_3 = x_2 + 2;$
6.	$y_3 = y_2 + 2 \times x_3^2;$}
7.	*print*(x_2, y_2);
8.	stop

(b) Syntactic PDG of P_{while}



(c) PDG after relevancy computation of P_{while} *w.r.t.* PAR



Example 26 *Consider the program in Figure 8.7(a). Observe that the statements 6 and 7 in the "if" block blk_{if} and the statement 8 in the "else" block blk_{else} are semantically irrelevant w.r.t. the sign property of the variables. The execution of the statements 6, 7, and 8 over all possible abstract states reaching these program points does not change the sign of y and z. Thus, the "if" block blk_{if} is completely irrelevant, whereas the "else" block blk_{else} is partially relevant as it contains one relevant statement 9 w.r.t. the sign property. According to the rules discussed above, we can't remove blk_{if}. Hence, we replace blk_{if} with the statement "skip" (denoted by label "q"). The corresponding semantics-based form of the program and semantics-based abstract PDG w.r.t. the sign property are shown in Figure 8.7(b) and 8.7(c) respectively.*

Figure 8.7: Treating "if-$else$" block

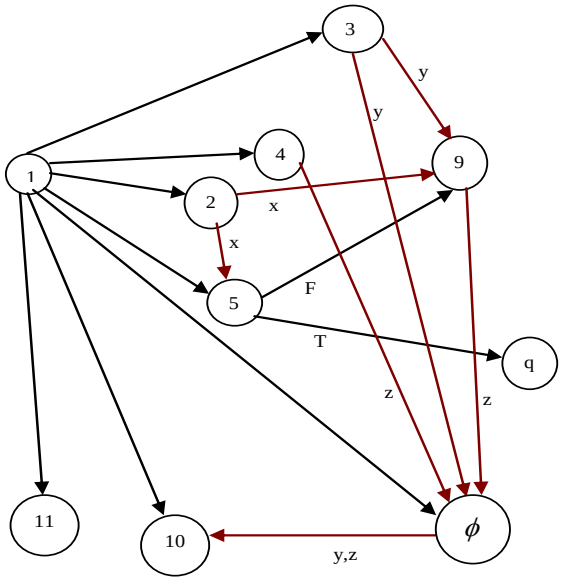
(a) Program P_{ifelse}

```
1.    start
2.    x1 = input;
3.    y1 = 5;
4.    z1 = 2;
5.    if(x1 ≥ 0){
6.          y2 = y1 + 2;
7.          z2 = y2 + x1;}
      else{
8.          y3 = y1 + 5;
9.          z3 = y3 + x1};
φ.    (y4, z4) = f((y2, z2), (y3, z3));
10.   print(y4, z4);
11.   stop
```

(b) P^{rel}_{ifelse}: after relevancy computation of P_{ifelse} *w.r.t.* SIGN

```
1.    start
2.    x1 = input;
3.    y1 = 5;
4.    z1 = 2;
5.    if(x1 ≥ 0)
q.          skip;
      else{
9.          z3 = y1 + x1};
φ.    (y4, z4) = f((y1, z1), (y1, z3));
10.   print(y4, z4);
11.   stop
```

(c) PDG of P^{rel}_{ifelse}



8.4 Algorithm for Semantics-based Abstract PDG

We are now in position to formalize a new algorithm to construct semantics-based abstract PDG $G^{r,d}_{pdg}$ of a program P *w.r.t.* an abstract property ρ as depicted in Figure 8.8. In this proposed algorithm, we combine (*i*) the notion of semantic relevancy of statements, and (*ii*) the notion of semantic data dependences of expressions.

We use the notation ϵ_{ij} to denote the j^{th} abstract state *w.r.t.* ρ possibly reaching program point p_i. The input of the algorithm is a program P and output is the semantics-based abstract PDG $G^{r,d}_{pdg}$ of P *w.r.t.* ρ.

Step 2 computes the semantic relevancy of all "*assignment*" statements in the program P *w.r.t.* ρ, and thus at step 4, P_{rel} contains only the relevant non-control statements along with all the control statements from P. Steps 5 computes the relevancy of control statements in P_{rel} *w.r.t.* ρ. Step 6 deals with the repetitive statement "*while*(*cond*) *then* blk_{while}", step 7 deals with the conditional statement "if(*cond*) *then* blk_{if}" and step 8 deals with the conditional statement "if(*cond*) *then* blk_{if} *else* blk_{else}", where we denote by blk_S a block of a set of statements S. Observe that steps 7 and 9 disregard the irrelevant control statements from P_{rel}, whereas steps 6, 10 and 11 replace the control statements by another form with equivalent relevancy *w.r.t.* ρ. In step 13, we compute abstract semantic data dependences for all expressions in P_{rel} by following the algorithm of Mastroeni and Zanardini (140). Finally, in step 14, we construct PDG from P_{rel} that contains only the relevant statements and the relevant data dependences *w.r.t.* ρ.

The idea to obtain a semantics-based abstract PDG is to unfold the program into an equivalent program where only statements that have an impact *w.r.t.* the abstract domain are combined with the semantic data flow *w.r.t.* the same domain.

8.5 Dependence Condition Graph (DCG)

In this section, we extend the semantics-based abstract PDGs obtained so far into semantics-based abstract Dependence Condition Graphs (DCGs).

The notion of Dependence Condition Graphs (DCGs) is introduced by Sukumaran et al. in (182). A DCG is built from the PDG by annotating each edge $e = e.src \rightarrow e.tgt$ in the PDG with information $e^b = \langle e^R, e^A \rangle$ that captures the conditions under which the dependence represented by that edge is manifest. The first component e^R refers to *Reach Sequences*, whereas the second component e^A refers to *Avoid Sequences*. The informal interpretation of e^R is that the conditions represented by it should be true for an execution to ensure that $e.tgt$ is reached from $e.src$. The *Avoid Sequences* e^A captures

Figure 8.8: Algorithm to generate Semantics-based Abstract PDG

Algorithm 7: REFINE-PDG

Input: Program P and an abstract domain ρ
Output: Semantics-based Abstract PDG $G^{r,d}_{pdg}$ of P *w.r.t.* ρ

1. `FOR` each *assignment*-statement s at program point p_i in P `DO`
2. $\forall \epsilon_{ij} \in \Sigma^{\rho}_{p_i}$: execute s on ϵ_{ij} and determine its relevancy;
3. `END FOR`
4. Disregard all the irrelevant *assignment*-statements from P and generate its relevant version P_{rel};
5. `FOR` each *control*-statement in P_{rel} `DO`
6. Case 1: Repetitive statement "$while(cond)$ *then* blk_{while}":
   If the block blk_{while} is semantically irrelevant *w.r.t.* ρ, replace "$while(cond)$ *then* blk_{while}" in P_{rel} by the statement "$while(cond)$ *then skip*";
7. Case 2: Conditional statement "$if(cond)$ *then* blk_{if}":
   If the block blk_{if} is semantically irrelevant *w.r.t.* ρ, disregard "$if(cond)$ *then* blk_{if}" from P_{rel};
8. Case 3: Conditional statement "$if(cond)$ *then* blk_{if} *else* blk_{else}":
9. Case 3a: Both blk_{if} and blk_{else} are semantically irrelevant *w.r.t.* ρ:
   Disregard the statement "$if(cond)$ *then* blk_{if} *else* blk_{else}" from P_{rel};
10. Case 3b: Only blk_{else} is semantically irrelevant *w.r.t.* ρ:
    Replace the statement "$if(cond)$ *then* blk_{if} *else* blk_{else}" in P_{rel} by the statement "$if(cond)$ *then* blk_{if}";
11. Case 3c: Only blk_{if} is semantically irrelevant *w.r.t.* ρ:
    Replace the statement "$if(cond)$ *then* blk_{if} *else* blk_{else}" in P_{rel} by the statement "$if(cond)$ *then skip else* blk_{else}";
12. `END FOR`
13. Compute abstract semantic data dependences for all expressions in P_{rel} *w.r.t.* ρ by following the algorithm of Mastroeni and Zanardini;
14. Construct PDG from P_{rel} by using only the relevant statements and relevant data dependences *w.r.t.* ρ, as obtained in previous steps;

the possible conditions under which the assignment at *e.src* can get over-written before it reaches *e.tgt*. The interpretation of e^A is that the conditions represented by it must not hold in an execution to ensure that the variable being assigned at *e.src* is used at *e.tgt*. It is worthwhile to note that e^A is relevant only for DDG edges and it is $\emptyset$ for CDG edges. Example 27 illustrates briefly how to compute annotations over the edges in a PDG.

Example 27 *Consider the edge* $2 \xrightarrow{x} 8$ *in the semantics-based abstract PDG of Figure 8.5. For this edge, we see that node* 8 *does not post-dominate the node* 2*. Thus, the reach sequence for the edge is* $(2 \xrightarrow{x} 8)^R = \{1 \xrightarrow{true} 6 \xrightarrow{true} 8\}$*. This means that the condition at* 6 *must be true in the execution trace to ensure that both* 2 *and* 8 *are executed, and the data x assigned at* 2 *can reach* 8*. For the edge* $2 \xrightarrow{x} \phi$*, the reach sequence* $(2 \xrightarrow{x} \phi)^R$ *is empty, meaning that no condition has to satisfy for the x assigned at* 2 *to reach* ϕ*, because* ϕ *post-dominates* 2 *and once* 2 *is executed* ϕ *is also executed. To compute the avoid sequences for the edge* $2 \xrightarrow{x} \phi$*, we consider two data dependence edges* $e_1 = 2 \xrightarrow{x} \phi$ *and* $e_2 = 8 \xrightarrow{x} \phi$ *with* ϕ *as target. By following the algorithm of (182), we get the avoid sequences* $(2 \xrightarrow{x} \phi)^A = \{1 \xrightarrow{true} 6 \xrightarrow{true} 8\}$*. This reflects the fact that the "if" condition at 6 must be false in order to guarantee that the definition of x at* 2 *is not re-defined at* 8 *and can reach* ϕ*. Table 8.1 depicts the DCG annotations for all DDG edges of the semantics-based abstract PDG in Figure 8.5.*

Table 8.1: DCG annotations $\langle e^R, e^A \rangle$ for DDG edges e of G^r_{pdg}

e	e^R	e^A
$2 \xrightarrow{x} 6$	$\emptyset$	$\emptyset$
$3 \xrightarrow{y} 6$	$\emptyset$	$\emptyset$
$2 \xrightarrow{x} 8$	$1 \xrightarrow{true} 6 \xrightarrow{true} 8$	$\emptyset$
$2 \xrightarrow{x} 9$	$1 \xrightarrow{true} 6 \xrightarrow{false} 9$	$\emptyset$
$4 \xrightarrow{w} 9$	$1 \xrightarrow{true} 6 \xrightarrow{false} 9$	$\emptyset$
$2 \xrightarrow{x} \phi$	$\emptyset$	$1 \xrightarrow{true} 6 \xrightarrow{true} 8$
$8 \xrightarrow{x} \phi$	$\emptyset$	$\emptyset$
$5 \xrightarrow{z} \phi$	$\emptyset$	$1 \xrightarrow{true} 6 \xrightarrow{false} 9$
$9 \xrightarrow{z} \phi$	$\emptyset$	$\emptyset$
$\phi \xrightarrow{x,z} 11$	$\emptyset$	$\emptyset$

Sukumaran et al. (182) described the semantics of DCG annotations in terms of execution semantics of the program over concrete domain.

Below, we first define the abstract semantics of DCG annotations in an abstract domain of interest. Then, we propose a refinement of the DCGs by removing semantically unrealizable dependences from them under their abstract semantics.

Abstract Semantics of $e^b \triangleq \langle e^R, e^A \rangle$

The program executions are recorded in finite or infinite sequences of states over a given set of commands, called traces. An execution trace ψ of a program P over an abstract domain ρ is a (possibly infinite) sequence $\langle (p_i, \epsilon_{p_i}) \rangle_{i \geq 0}$ where ϵ_{p_i} represents the abstract data state at the entry of the statement at program point p_i in P. We use the

notion "$\iota : (p_i, \epsilon_{p_i})$" to denote that (p_i, ϵ_{p_i}) is the ι-th element in the sequence ψ. The trace ψ holds the following conditions:

1. The first abstract state in the sequence is (p_0, ϵ_{p_0}) where p_0 = "start" and ϵ_{p_0} is the initial abstract data state.
2. Each state (p_i, ϵ_{p_i}), $i = 1, 2, 3, \ldots$ is the successor of the previous state $(p_{i-1}, \epsilon_{p_{i-1}})$.
3. The last abstract state in the sequence ψ of length $\#\psi = m$, if it exists, is (p_m, ϵ_{p_m}) where p_m="stop".

Note that the DCG nodes corresponding to the statements at program points p_i are labeled by p_i. We denote the control dependence edge (CDG edge) in DCG by $e = p_i \xrightarrow{lab} p_j$, where the node p_i corresponds to the conditional or repetitive statement containing the condition $p_i.cond$ and the label $e.lab$ associated with e represents the truth value (either *true* or *false*). We denote the data dependence edge (DDG edge) in DCG by $e = p_i \xrightarrow{x} p_j$, where x is the data defined by the statement corresponding to the node p_i.

We now define the semantics of the annotations $e^b \triangleq (e^R, e^A)$ on dependence edges e in DCG in terms of the execution traces ψ over an abstract domain ρ.

Definition 29 (Execution satisfying e^b for a CDG edge e at index ι) *An execution trace ψ over an abstract domain ρ is said to satisfy e^b at index ι for a CDG edge $e = p_i \xrightarrow{lab} p_j$ (written as $\psi \vdash^{\rho}_{\iota} e$) if the following conditions hold:*

- $e^b \triangleq \langle e^R, e^A \rangle = \langle \{e\}, \emptyset \rangle$, *and*
- ψ *contains* $\iota : (p_i, \epsilon_{p_i})$ *such that* $[\![p_i.cond = e.lab]\!](\epsilon_{p_i})$ *yields either to "true" or to the logic value "unknown" (meaning possibly true or possibly false).*

Definition 30 (Execution satisfying e^b for a CDG edge e) *An execution trace ψ over an abstract domain ρ satisfying e^b for a CDG edge $e = p_i \xrightarrow{lab} p_j$ (written as $\psi \vdash^{\rho} e$) is defined as:*

$$\psi \vdash^{\rho} e \triangleq \exists \iota \geq 0 : \psi \vdash^{\rho}_{\iota} e$$

Definition 31 (Execution satisfying e^R for a DDG edge e at ι) *An execution trace ψ over an abstract domain ρ is said to satisfy e^R at index ι for a DDG edge $e = p_i \xrightarrow{x} p_j$ (written as $\psi \vdash^{R}_{\iota} e$) if the following conditions hold:*

- *The trace ψ contains $\iota : (p_i, \epsilon_{p_i})$, and*

- *Either* $e^R = \emptyset$ *or for each* $(p_{s_1} \xrightarrow{lab_{s_1}} p_{s_2} \xrightarrow{lab_{s_2}} \dots p_{s_n} \xrightarrow{lab_{s_n}} p_j) \in e^R$ *where* $p_{s_1}, p_{s_2}, \dots, p_{s_n}$ *correspond to conditional statements, the trace* ψ *contains* $\iota_k : (p_{s_k}, \epsilon_{p_{s_k}})$ *for* $k = 1, \dots, n$ *and* $\iota_1 < \iota < \iota_2 < \dots < \iota_n$ *and* $\bigwedge_{1 \le k \le n} [\![p_{s_k}.cond = lab_{s_k}]\!](\epsilon_{p_{s_k}})$ *yields to "true" or "unknown".*

Definition 32 (Execution satisfying e^A for a DDG edge e at ι) *An execution trace* ψ *over an abstract domain* ρ *is said to satisfy* e^A *at index* ι *for a DDG edge* $e = p_i \xrightarrow{x} p_j$ *(written as* $\psi \vdash_\iota^A e$*) if* ψ *contains* $\iota : (p_i, \epsilon_{p_i})$*, and for each* $(p_{s_1} \xrightarrow{lab_{s_1}} p_{s_2} \xrightarrow{lab_{s_2}} \dots p_{s_n} \xrightarrow{lab_{s_n}} p') \in e^A$ *the subtrace* ψ' *of* ψ *from the index* ι *to the next occurrence of* (p_j, ϵ_{p_j}) *(or, if* (p_j, ϵ_{p_j}) *does not occur then* ψ' *is the suffix of* ψ *starting from* ι*), satisfies exactly one of the following conditions:*

- ψ' *does not contain* $(p_{s_k}, \epsilon_{p_{s_k}})$ *for* $1 \le k \le n$*, or*
- $\exists\, k :\ 1 \le k \le n$*:* ψ' *contains* $(p_{s_k}, \epsilon_{p_{s_k}})$ *such that* $[\![p_{s_k}.cond = lab_{s_k}]\!](\epsilon_{p_{s_k}})$ *yields to false.*

Definition 33 (Execution satisfying e^b for a DDG edge e at ι) *An execution trace* ψ *over an abstract domain* ρ *satisfying* e^b *at index* ι *for a DDG edge* $e = p_i \xrightarrow{x} p_j$ *(written as* $\psi \vdash_\iota^\rho e$*) is defined as*

$$\psi \vdash_\iota^\rho e \triangleq (\psi \vdash_\iota^R e) \wedge (\psi \vdash_\iota^A e)$$

Definition 34 (Execution satisfying e^b for a DDG edge e) *An execution trace* ψ *over an abstract domain* ρ *satisfying* e^b *for a DDG edge* $e = p_i \xrightarrow{x} p_j$ *(written as* $\psi \vdash^\rho e$*) is defined as*

$$\psi \vdash^\rho e \triangleq \exists \iota \ge 0 : \psi \vdash_\iota^\rho e$$

Theorem 2 *Given a DDG edge* $e = p_i \xrightarrow{x} p_j$ *and an execution trace* ψ *over an abstract domain* ρ*, the trace* ψ *satisfies* e^b *for* e *(denoted* $\psi \vdash^\rho e$*) iff the abstract value of* x *computed at* p_i *reaches the next occurrence of* p_j *in* ψ*.*

Proof *Since the execution trace* ψ *over an abstract domain* ρ *satisfies* e^b *for* $e = p_i \xrightarrow{x} p_j$*, we have*

$$\begin{aligned} \psi \vdash^\rho e &\triangleq \exists \iota \ge 0 :\ \psi \vdash_\iota^\rho e \\ &\triangleq \exists \iota \ge 0 :\ (\psi \vdash_\iota^R e) \wedge (\psi \vdash_\iota^A e) \end{aligned}$$

This means that ψ *satisfies both the Reach Sequences* e^R *and the Avoid Sequences* e^A *for* e *at some index* ι *in* ψ*.*

The trace ψ satisfies e^R at some index ι meaning that ψ contains $\iota : (p_i, \epsilon_{p_i})$, and ψ satisfies all the conditions in e^R which ensures that p_j is reached from p_i i.e. the abstract value of x computed at p_i can reach p_j. At the same time, the trace ψ satisfies e^A at ι meaning that ψ avoids the execution of all the other possibilities (if exists) so that the abstract value of x computed at p_i can not be overwritten by any other intermediate statements that also define x. Thus, $\psi \vdash^{\rho} e$ implies that the abstract value of x computed at p_i reaches the next occurrence of p_j in ψ.

On the other side, we should also prove that if the abstract value of x computed at p_i reaches the next occurrence of p_j in ψ, then $\psi \vdash^{\rho} e$ where $e = p_i \xrightarrow{x} p_j$.

Since the abstract value of x computed at p_i reaches to the next occurrence of p_j, we can say that there is entries $\iota : (p_i, \epsilon_{p_i})$ and $\iota' : (p_j, \epsilon_{p_j})$ in ψ where ι' is the smallest index greater than ι and the statements corresponding to both p_i and p_j are executed in ψ (i.e. ψ satisfies Reach Sequences e^R), and at the same time it avoids the execution of all other intermediate statements which can overwrite the abstract value of x coming from p_i (i.e. ψ satisfies Avoid Conditions e^A). Thus, $\exists \iota \geq 0 : \ (\psi \vdash_{\iota}^{R} e) \wedge (\psi \vdash_{\iota}^{A} e)$ i.e. $\psi \vdash^{\rho} e$.

Example 28 *Consider the program P and its PDG depicted in Figure 8.9(a) and 8.9(b) respectively. The set of program variables in P is $VAR = \{x, y\}$. Consider the DDG edge $e = 2 \xrightarrow{x} \phi$. By following the algorithm in (182), we get $e^R = \{1 \xrightarrow{true} 4 \xrightarrow{true} \phi\}$ and $e^A = \{1 \xrightarrow{true} 4 \xrightarrow{true} 5 \xrightarrow{true} 6\}$. The DCG annotations over the DDG edges are shown in Figure 8.9(c).*

Consider the abstract domain SIGN. The initial state of P is defined by $\langle start, \epsilon_{start} \rangle = \langle start, (\perp, \perp) \rangle$ where $\epsilon_{start} = (\perp, \perp)$ are the initial abstract values for $x, y \in VAR$ respectively. Consider the execution trace

$\psi = \iota_1 : \langle 1, (\perp, \perp) \rangle \ \iota_2 : \langle 2, (\perp, \perp) \rangle \ \iota_3 : \langle 3, (+, \perp) \rangle \ \iota_4 : \langle 4, (+, -) \rangle \ \iota_5 : \langle 5, (+, -) \rangle \ \iota_{\phi} : \langle \phi, (+, -) \rangle$
$\iota_7 : \langle 7, (+, -) \rangle \ \iota_8 : \langle 8, (+, -) \rangle$

where in each state of ψ the first component represents program point of the corresponding statement and the other component represents abstract values of x and y respectively. Note that the condition from the statement at program point 1 to 4 is implicitly "true" irrespective of the states at 1. We have $\psi \vdash_{\iota_2}^{R} (2 \xrightarrow{x} \phi)$ because

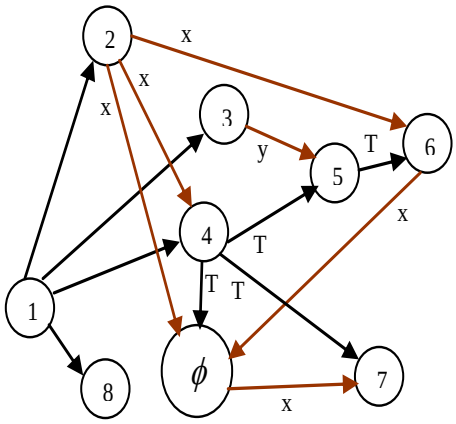
- *ψ contains the entry $\iota_2 : \langle 2, (\perp, \perp) \rangle$ corresponding to the statement 2 at index ι_2, and*
- *ψ is of the form $\psi = \iota_1 : \langle 1, (\perp, \perp) \rangle \ \iota_2 : \langle 2, (\perp, \perp) \rangle \ldots \iota_4 : \langle 4, (+, -) \rangle \ldots \iota_{\phi} : \langle \phi, (+, -) \rangle \ldots$ for $1 \xrightarrow{true} 4 \xrightarrow{true} \phi \in (2 \xrightarrow{x} \phi)^R$ such that $[\![1.cond = true]\!](\perp, \perp)$ and $[\![4.cond = true]\!](+, -)$ are evaluated to "true".*

Figure 8.9: A program and its DCG annotations

(a) Program P

```
1.    start
2.    x1 = input;
3.    y = input;
4.    if(x1 > 0){
5.        if(y == 5)
6.            x2 = x1 × 2;
φ.        x3 = f(x1, x2);
7.        print(x3); }
8.    stop
```

(b) G_{pdg} : PDG of P



(c) DCG annotations $\langle e^R, e^A \rangle$ for DDG edges e of G_{pdg}

e	e^R	e^A
$2 \xrightarrow{x} 6$	$1 \xrightarrow{true} 4 \xrightarrow{true} 5 \xrightarrow{true} 6$	$\emptyset$
$2 \xrightarrow{x} 4$	$\emptyset$	$\emptyset$
$2 \xrightarrow{x} \phi$	$1 \xrightarrow{true} 4 \xrightarrow{true} \phi$	$1 \xrightarrow{true} 4 \xrightarrow{true} 5 \xrightarrow{true} 6$
$3 \xrightarrow{y} 5$	$1 \xrightarrow{true} 4 \xrightarrow{true} 5$	$\emptyset$
$6 \xrightarrow{x} \phi$	$\emptyset$	$\emptyset$
$\phi \xrightarrow{x} 7$	$\emptyset$	$\emptyset$

Similarly, $\psi \vdash^A_{\iota_2} (2 \xrightarrow{x} \phi)$*, because for* $1 \xrightarrow{true} 4 \xrightarrow{true} 5 \xrightarrow{true} 6 \in (2 \xrightarrow{x} \phi)^A$ *the sub-trace of* ψ *contains the entry* $\iota_5 : \langle 5, (+,-) \rangle$ *such that* $[\![5.cond = true]\!](+,-)$ *is "false".*

As $\psi \vdash^R_{\iota_2} (2 \xrightarrow{x} \phi)$ *and* $\psi \vdash^A_{\iota_2} (2 \xrightarrow{x} \phi)$*, we can say* $\psi \vdash^{SIGN}_{\iota_2} (2 \xrightarrow{x} \phi)$ *meaning that in* ψ *the sign of* x *defined at program point* 2 *reaches program point* ϕ*, and it is not changed or overwritten by the intermediate statement* 6*.*

Abstract Semantics of Dependence Paths in DCGs

The final step, in order to combine Dependence Condition Graphs with abstract semantics-based Program Dependence Graphs, is to define the abstract semantics of the dependence paths in a Dependence Condition Graph.

Given a program P and its DCG, we consider dependence paths in this graph. First we define the ϕ-sequence and then the semantics of a dependence path over an abstract domain ρ.

Definition 35 (PhiSeqs) *A ϕ-sequence η_ϕ is a DDG path of the form: $n_1 \rightarrow \phi_1 \rightarrow \phi_2 \rightarrow \cdots \rightarrow \phi_k \rightarrow n_2$, where n_1 and n_2 are nodes of the program and all the ϕ_i $(1 \leq i \leq k)$ are ϕ-nodes (that correspond to assignments to the same variable along different paths). Observe that all edges on a ϕ-sequence will be labeled with the same variable.*

Consider an arbitrary dependence path $\eta = e_1 e_2 \ldots e_n$ in DCG representing a chain of dependences. To satisfy η by an execution trace ψ over an abstract domain ρ, we need to satisfy the annotations e^b of each edge e_i, $i \in [1..n]$, at some ι_i (*i.e.*, $\psi \vdash^\rho_{\iota_i} e_i$) such that the execution sub-traces of ψ corresponding to the e_i are contiguous.

Definition 36 (Evidence) *For an execution trace ψ over an abstract domain ρ and a dependence edge e, s.t. $\psi \vdash^\rho_\iota e$, $evidence(\psi, e, \iota) = \iota'$ where ι' is the index of the first occurrence of $(e.tgt, -)$ in ψ from index ι.*

Definition 37 (Execution satisfying a dependence path) *A series of program dependences represented by a dependence path $\eta = e_1 e_2 \ldots e_n$ is said to be satisfied by an execution ψ over an abstract domain ρ (written as $\psi \vdash^\rho \eta$) if*

$$\bigwedge_{1 \leq i \leq n} \psi \vdash^\rho_{\iota_i} e_i \quad \wedge \; (\; \forall 1 \leq i \leq n : evidence(\psi, e_i, \iota_i) = \iota_{i+1} \;)$$

Theorem 3 *Given a ϕ-sequence $\eta_\phi = e_1 e_2 \ldots e_n$ and the execution trace ψ over an abstract domain ρ, the trace ψ satisfies η_ϕ (denoted $\psi \vdash^\rho \eta_\phi$) iff the abstract value computed at $e_1.src$ reaches $e_n.tgt$ in ψ along the execution path that satisfies η_ϕ.*

Proof *Since $\psi \vdash^\rho \eta$, we have*

$$\bigwedge_{1 \leq i \leq n} \psi \vdash^\rho_{\iota_i} e_i \quad \wedge \; (\; \forall 1 \leq i \leq n : evidence(\psi, e_i, \iota_i) = \iota_{i+1} \;)$$

This means that the sub-traces ψ_i of ψ satisfying the annotations of e_i of η, where $i = 1, \ldots, n$, are contiguous. Observe that all the edges e_i of η are labeled with the same variable x. Consider any two consecutive edges $e_i = p_r \xrightarrow{x} p_s$ and $e_{i+1} = p_s \xrightarrow{x} p_t$ in η where $1 \leq i < i+1 \leq n$ and the corresponding contiguous sub-traces ψ_i and ψ_{i+1} that satisfy e_i and e_{i+1} respectively. From Theorem 2, we can say that the abstract value of x can reach from p_r to p_s and from p_s to p_t in ψ_i and ψ_{i+1} respectively. If the intermediate node p_s is a ϕ node (which does not recompute the value but only pass through), then this transitivity implies that the abstract value of x can reach from p_r to p_t in ψ_q where ψ_q is the concatenation of ψ_i and ψ_{i+1}. Since in a ϕ-sequence all the intermediate nodes are ϕ nodes, we can extend this transitivity for all i from 1 to n, and we can say that the abstract value of x can reach from $e_1.src$ to $e_n.tgt$ in ψ where ψ is the concatenation of all subtraces ψ_i, $i = 1, \ldots, n$. Observe that in a ϕ-sequence the starting and

end nodes are not ϕ nodes, and the datum is computed at starting node and is used by the end node. Thus, the abstract value computed at $e_1.src$ reaches $e_n.tgt$ in ψ.

Let us now prove the "only if" part of the theorem: given a ϕ-sequence $\eta = e_1 e_2 \ldots e_n$ and an execution trace ψ over an abstract domain ρ, if the abstract value at $e_1.src$ reaches $e_n.tgt$ in ψ, then $\psi \vdash^{\rho} \eta$.

Since in ϕ-sequence $\eta = e_1 e_2 \ldots e_n$ all the intermediate nodes are ϕ nodes except the starting and end ones, and the datum computed at $e_1.src$ reaches $e_n.tgt$ in ψ, from Theorem 2 we can say that $\forall i$, $1 \leq i \leq n$: $\psi \vdash^{\rho}_{\iota_i} e_i$ where ι_i is the index of $(e_i.src, -)$ in ψ. Now we show that $evidence(\psi, e_i, \iota_i) = \iota_{i+1}$ for all i, $1 \leq i \leq n$. Consider two consecutive edges $e_i = p_r \xrightarrow{x} p_s$ and $e_{i+1} = p_s \xrightarrow{x} p_t$ in η where $1 \leq i < i+1 \leq n$. Since $\psi \vdash^{\rho}_{\iota_i} e_i$ and $\psi \vdash^{\rho}_{\iota_{i+1}} e_{i+1}$, the trace ψ contains $\iota_i : (p_r, \epsilon_{p_r})$ and $\iota_{i+1} : (p_s, \epsilon_{p_s})$. Thus, we have $evidence(\psi, e_i, \iota_i) = \iota_{i+1}$ because ι_{i+1} is the index of the first occurrence of $(e_i.tgt, -)$ i.e. (p_s, ϵ_{p_s}) in ψ from the index ι_i. Therefore, we have

$$\bigwedge_{1 \leq i \leq n} \psi \vdash^{\rho}_{\iota_i} e_i \quad \wedge \; (\forall 1 \leq i \leq n : evidence(\psi, e_i, \iota_i) = \iota_{i+1})$$

That is,

$$\psi \vdash^{\rho} \eta.$$

Example 29 *Consider the dependence path $\eta = 2 \xrightarrow{x} 6 \xrightarrow{x} \phi \xrightarrow{x} 7$ in the graph of Figure 8.9, and the following execution trace over the abstract domain SIGN:*

$\psi = \iota_1 : \langle 1, (\bot, \bot) \rangle \; \iota_2 : \langle 2, (\bot, \bot) \rangle \; \iota_3 : \langle 3, (+, \bot) \rangle \; \iota_4 : \langle 4, (+, +) \rangle \; \iota_5 : \langle 5, (+, +) \rangle \; \iota_6 : \langle 6, (+, +) \rangle$
$\iota_7 : \langle \phi, (+, +) \rangle \; \iota_8 : \langle 7, (+, +) \rangle \; \iota_9 : \langle 8, (+, +) \rangle$

The trace ψ satisfies e^b for all the edges $2 \xrightarrow{x} 6$, $6 \xrightarrow{x} \phi$ and $\phi \xrightarrow{x} 7$ of η, and the sub-traces of ψ that satisfy these edges are contiguous, that is,

- $\psi \vdash^{SIGN}_{\iota_2} (2 \xrightarrow{x} 6)$ *and* $evidence(\psi, 2 \xrightarrow{x} 6, \iota_2) = \iota_6$,

  where $1 \xrightarrow{true} 4 \xrightarrow{true} 5 \xrightarrow{true} 6 \in (2 \xrightarrow{x} 6)^R$ and $(2 \xrightarrow{x} 6)^A = \emptyset$ and ψ is of the form $\iota_1 : \langle 1, (\bot, \bot) \rangle \; \iota_2 : \langle 2, (\bot, \bot) \rangle \ldots \iota_4 : \langle 4, (+, +) \rangle \; \iota_5 : \langle 5, (+, +) \rangle \; \iota_6 : \langle 6, (+, +) \rangle \ldots$ such that $[\![1.cond = true]\!](\bot, \bot)$, $[\![4.cond = true]\!](+, +)$ are evaluated to "true" and $[\![5.cond = true]\!](+, +)$ is evaluated to "unknown".

- $\psi \vdash^{SIGN}_{\iota_6} (6 \xrightarrow{x} \phi)$ *and* $evidence(\psi, 6 \xrightarrow{x} \phi, \iota_6) = \iota_7$,

  where $(6 \xrightarrow{x} \phi)^R = \emptyset$ and $(6 \xrightarrow{x} \phi)^A = \emptyset$ and ψ is of the form $\ldots \iota_6 : \langle 6, (+, +) \rangle$ $\iota_7 : \langle \phi, (+, +) \rangle \ldots$.

- $\psi \vdash_{\iota_7}^{SIGN} (\phi \xrightarrow{x} 7)$ *and* $evidence(\psi, \phi \xrightarrow{x} 7, \iota_7) = \iota_8$,

  where $(\phi \xrightarrow{x} 7)^R = \emptyset$ *and* $(\phi \xrightarrow{x} 7)^A = \emptyset$ *and* ψ *is of the form* $\ldots \iota_7 : \langle \phi, (+,+) \rangle$ $\iota_8 : \langle 7, (+,+) \rangle \ldots$.

Thus, we can say that the dependence path η *is satisfied by* ψ *over the abstract domain SIGN i.e.* $\psi \vdash^{SIGN} \eta$.

Satisfiability of dependence paths combined with semantic relevancy computation

Let η be a dependence path in a DCG. Suppose an execution trace ψ over an abstract domain ρ satisfies η (denoted $\psi \vdash^{\rho} \eta$). If we compute semantic relevancy *w.r.t.* ρ and we disregard the irrelevant entries from both η and ψ, we see that the satisfiability of the refined path is also preserved, as depicted in Theorem 4.

Theorem 4 *Given a program P and its DCG* G_{dcg}. *Let* ψ *be an execution trace of P over an abstract domain* ρ, *and* $\eta = e_1 e_2 \ldots e_l e_{l+1} \ldots e_h$ *be a dependence path in* G_{dcg} *where* $e_l : p_i \xrightarrow{x} p_j$ *and* $e_{l+1} : p_j \xrightarrow{x} p_k$ (e_l *and* e_{l+1} *are contiguous). Suppose removal of the element corresponding to irrelevant statement at* p_j *w.r.t.* ρ *from* η *and* ψ *yield to a dependence path* $\eta' = e_1 e_2 \ldots e_q \ldots e_h$, *where* $e_q : p_i \xrightarrow{x} p_k$, *and a trace* ψ' *respectively. Then,*

$$if \psi \vdash^{\rho} \eta, \text{ then } \psi' \vdash^{\rho} \eta'$$

.

Proof *Since* $\psi \vdash^{\rho} \eta$, *we can say that for the edges* $e_l : p_i \xrightarrow{x} p_j$ *and* $e_{l+1} : p_j \xrightarrow{x} p_k$ *in* η:

- $\psi \vdash_{\iota_i}^{R} e_l \wedge \psi \vdash_{\iota_i}^{A} e_l \wedge evidence(\psi, e_l, \iota_i) = \iota_j$, *and*
- $\psi \vdash_{\iota_j}^{R} e_{l+1} \wedge \psi \vdash_{\iota_j}^{A} e_{l+1} \wedge evidence(\psi, e_{l+1}, \iota_j) = \iota_k$

where, ι_i, ι_j *and* ι_k *are the indexes where* (p_i, ϵ_{p_i}), (p_j, ϵ_{p_j}) *and* (p_k, ϵ_{p_k}) *occur respectively in* ψ.

We already know that the dependence edges e in DCG are annotated by the Reach Sequences e^R *and the Avoid Sequences* e^A. *The Reach Sequences* e^R *for the edge* e : $e.src \xrightarrow{x} e.tgt$ *represents the conditions that need to be satisfied by the execution trace* ψ *to reach the data x from e.src to e.tgt. If* $e^R = \emptyset$, *it means that e.tgt post-dominates e.src and thus, the execution trace should contain e.src, and once e.src is executed e.tgt will also be executed which yield x to reach from e.src to e.tgt. When* $e^R \neq \emptyset$, *e.tgt does not post-dominate e.src and thus, the conditions in* e^R *need to be satisfied by the trace* ψ *so that e.tgt executes, and since* ψ *also contains e.src the data x must reach from e.src to e.tgt. Therefore, we have* ψ *as follows under the four different cases of Reach Sequences:*

r_1: $e_l^R = e_{l+1}^R = \emptyset$: *$p_i$ is post-dominated by p_j and p_j is post-dominated by p_k, and ψ contains $\iota_i : (p_i, \epsilon_{p_i})$, $\iota_j : (p_j, \epsilon_{p_j})$ and $\iota_k : (p_k, \epsilon_{p_k})$ where $\iota_i < \iota_j < \iota_k$.*

r_2: $e_l^R \neq \emptyset$ *and* $e_{l+1}^R = \emptyset$: *ψ contains $\iota_i : (p_i, \epsilon_{p_i})$, $\iota_j : (p_j, \epsilon_{p_j})$ and $\iota_k : (p_k, \epsilon_{p_k})$ where $\iota_i < \iota_j < \iota_k$. For each $(p_{s_1} \xrightarrow{lab_{s_1}} p_{s_2} \xrightarrow{lab_{s_2}} \dots p_{s_n} \xrightarrow{lab_{s_n}} p_j) \in e_l^R$, ψ contains $\iota_m : (p_{s_m}, \epsilon_{p_{s_m}})$ for $m = 1, \dots, n$ and $\iota_1 < \iota_i < \iota_2 < \dots < \iota_n < \iota_j < \iota_k$ and $\bigwedge_{1 \leq m \leq n} [\![p_{s_m}.cond = lab_{s_m}]\!](\epsilon_{p_{s_m}})$ yields to "true" or "unknown", and p_j is post-dominated by p_k.*

r_3: $e_l^R = \emptyset$ *and* $e_{l+1}^R \neq \emptyset$: *p_i is post-dominated by p_j and ψ contains $\iota_i : (p_i, \epsilon_{p_i})$, $\iota_j : (p_j, \epsilon_{p_j})$ and $\iota_k : (p_k, \epsilon_{p_k})$ where $\iota_i < \iota_j < \iota_k$. For each $(p_{s_1} \xrightarrow{lab_{s_1}} p_{s_2} \xrightarrow{lab_{s_2}} \dots p_{s_n} \xrightarrow{lab_{s_n}} p_k) \in e_{l+1}^R$, ψ contains $\iota_m : (p_{s_m}, \epsilon_{p_{s_m}})$ for $m = 1, \dots, n$ and $\iota_1 < \iota_i < \iota_j < \iota_2 < \dots < \iota_n < \iota_k$ and $\bigwedge_{1 \leq m \leq n} [\![p_{s_m}.cond = lab_{s_m}]\!](\epsilon_{p_{s_m}})$ yields to "true" or "unknown".*

r_4: $e_2^R \neq \emptyset$ *and* $e_3^R \neq \emptyset$: *ψ contains $\iota_i : (p_i, \epsilon_{p_i})$, $\iota_j : (p_j, \epsilon_{p_j})$ and $\iota_k : (p_k, \epsilon_{p_k})$ where $\iota_i < \iota_j < \iota_k$. For each $(p_{s_1} \xrightarrow{lab_{s_1}} p_{s_2} \xrightarrow{lab_{s_2}} \dots p_{s_n} \xrightarrow{lab_{s_n}} p_j) \in e_l^R$ and $(p_{s_n} \xrightarrow{lab_{s_n}} p_{s_{n+1}} \xrightarrow{lab_{s_{n+1}}} \dots p_{s_r} \xrightarrow{lab_{s_r}} p_k) \in e_{l+1}^R$, ψ contains $\iota_m : (p_{s_m}, \epsilon_{p_{s_m}})$ for $m = 1, \dots, r$ and $\iota_1 < \iota_i < \iota_2 < \dots < \iota_n < \iota_j < \iota_{n+1} < \dots < \iota_r < \iota_k$ and $\bigwedge_{1 \leq m \leq r} [\![p_{s_m}.cond = lab_{s_m}]\!](\epsilon_{p_{s_m}})$ yields to "true" or "unknown".*

We know that after computing semantic relevancy w.r.t. ρ and after removing the irrelevant element corresponding to p_j from η and ψ, we get $\eta' = e_1 e_2 \dots e_q \dots e_h$ where $e_q : p_i \xrightarrow{x} p_k$ and the execution trace ψ'. Now we have to show that $\psi' \vdash^{\rho} \eta'$. That is $\psi' \vdash^R_{\iota_i} e_q \wedge \psi' \vdash^A_{\iota_i} e_q$ and $evidence(\psi, e_q, \iota_i) = \iota_k$.

Corresponding to the above four cases r_1, r_2, r_3 and r_4, we have the following four cases:

r'_1: *$[e_l^R = \emptyset, e_{l+1}^R = \emptyset]$ $e_q^R = \emptyset$: Since p_i is post-dominated by p_j and p_j is post-dominated by p_k, after removing the irrelevant entry corresponding to p_j, we have that p_i is post-dominated by p_k. Since the trace ψ' contains $\iota_i : (p_i, \epsilon_{p_i})$ and $\iota_k : (p_k, \epsilon_{p_k})$ where $\iota_i < \iota_k$, we get $\psi' \vdash^R_{\iota_i} e_q$.*

r'_2: *$[e_l^R \neq \emptyset, e_{l+1}^R = \emptyset]$:*

- *$e_q^R = \emptyset$: Since p_i is not dominated by p_j and p_j is post-dominated by p_k, after removing the irrelevant element corresponding to p_j, it may happen that p_i is post-dominated by p_k. Since ψ' contains $\iota_i : (p_i, \epsilon_{p_i})$ and $\iota_k : (p_k, \epsilon_{p_k})$ where $\iota_i < \iota_k$, we get $\psi' \vdash^R_{\iota_i} e_q$.*
- *$e_q^R \neq \emptyset$: After removing the irrelevant element corresponding to p_j, we have that p_i is not post-dominated by p_k. In such case, the trace ψ' contains $\iota_i : (p_i, \epsilon_{p_i})$ and*

$\iota_k : (p_k, \epsilon_{p_k})$ *where* $\iota_i < \iota_k$ *and for each* $(p_{s_1} \xrightarrow{lab_{s_1}} p_{s_2} \xrightarrow{lab_{s_2}} \dots p_{s_t} \xrightarrow{lab_{s_t}} p_k) \in e_q^R$ *where* $t \in [2...n]$, *the trace* ψ' *contains* $\iota_m : (p_{s_m}, \epsilon_{p_{s_m}})$ *for* $m = 1, \dots, t$ *and* $\iota_1 < \iota_i < \iota_2 < \dots < \iota_t < \iota_k$ *and* $\bigwedge_{1 \le m \le t} [\![p_{s_m}.cond = lab_{s_m}]\!](\epsilon_{p_{s_m}})$ *yields to "true" or "unknown". Thus,* $\psi' \vdash^R_{\iota_i} e_q$.

r'_3**:** $[e_l^R = \emptyset, e_{l+1}^R \neq \emptyset]$ $e_q^R \neq \emptyset$*: Here removal of irrelevant element corresponding to* p_j, *we have that* p_i *is not post-dominated by* p_k. *In such case* ψ' *contains* $\iota_i : (p_i, \epsilon_{p_i})$ *and* $\iota_k : (p_k, \epsilon_{p_k})$ *where* $\iota_i < \iota_k$ *and for each* $(p_{s_1} \xrightarrow{lab_{s_1}} p_{s_2} \xrightarrow{lab_{s_2}} \dots p_{s_n} \xrightarrow{lab_{s_n}} p_k) \in e_{l+1}^R$, ψ' *contains* $\iota_m : (p_{s_m}, \epsilon_{p_{s_m}})$ *for* $m = 1, \dots, n$ *and* $\iota_1 < \iota_i < \iota_2 < \dots < \iota_n < \iota_k$ *and* $\bigwedge_{1 \le m \le n} [\![p_{s_m}.cond = lab_{s_m}]\!](\epsilon_{p_{s_m}})$ *yields to "true" or "unknown". Thus,* $\psi' \vdash^R_{\iota_i} e_q$.

r'_4**:** $[e_l^R \neq \emptyset, e_{l+1}^R \neq \emptyset]$ $e_q^R \neq \emptyset$*: Since* p_i *is not post-dominated by* p_j *and* p_j *is not post-dominated by* p_k, *the removal of element corresponding to* p_j *results that* p_i *is never post-dominated by* p_k. *In such case the trace* ψ' *contains* $\iota_i : (p_i, \epsilon_{p_i})$ *and* $\iota_k : (p_k, \epsilon_{p_k})$ *where* $\iota_i < \iota_k$ *and for each* $(p_{s_1} \xrightarrow{lab_{s_1}} p_{s_2} \xrightarrow{lab_{s_2}} \dots p_{s_t} \xrightarrow{lab_{s_t}} p_{s_{t+1}} \xrightarrow{lab_{s_{t+1}}} \dots p_{s_r} \xrightarrow{lab_{s_r}} p_k) \in e_q^R$ *where* $t \in [1...n]$, ψ' *contains* $\iota_m : (p_{s_m}, \epsilon_{p_{s_m}})$ *for* $m = 1, \dots, r$ *and* $\iota_1 < \iota_i < \iota_2 < \dots < \iota_t < \iota_{t+1} < \dots < \iota_r < \iota_k$ *and* $\bigwedge_{1 \le m \le r} [\![p_{s_m}.cond = lab_{s_m}]\!](\epsilon_{p_{s_m}})$ *yields to "true" or "unknown".*

Similarly, we can prove that $\psi' \vdash^A_{\iota_1} e_q$.

Thus, for any syntax-based dependence path $\eta = e_1 e_2 \dots e_l e_{l+1} \dots e_h$ *where* $e_l : p_i \xrightarrow{x} p_j$ *and* $e_{l+1} : p_j \xrightarrow{x} p_k$, *and an execution trace* ψ *over an abstract domain* ρ: *if* $\psi \vdash^\rho \eta$ *then* $\psi' \vdash^\rho \eta'$, *where* $\eta' = e_1 e_2 \dots e_q \dots e_h$ *(where* $e_q = p_i \xrightarrow{x} p_k$*) and* ψ' *are the semantics-based dependence path and semantics-based execution trace corresponding to* η *and* ψ *respectively obtained after computing semantic relevancy w.r.t.* ρ .

Example 30 *Look at Figure 8.9 and consider the dependence path* $\eta = 2 \xrightarrow{x} 6 \xrightarrow{x} \phi \xrightarrow{x} 7$ *and the following execution trace over the abstract domain SIGN:*

$\psi = \iota_1 : \langle 1, (\bot, \bot) \rangle$ $\iota_2 : \langle 2, (\bot, \bot) \rangle$ $\iota_3 : \langle 3, (+, \bot) \rangle$ $\iota_4 : \langle 4, (+, +) \rangle$ $\iota_5 : \langle 5, (+, +) \rangle$ $\iota_6 : \langle 6, (+, +) \rangle$
$\iota_7 : \langle \phi, (+, +) \rangle$ $\iota_8 : \langle 7, (+, +) \rangle$ $\iota_9 : \langle 8, (+, +) \rangle$

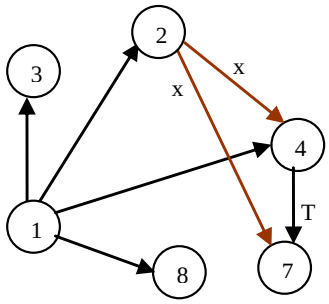
Note that $\psi \vdash^{SIGN} \eta$, *as already shown in Example 29. Figure 8.10(a) and 8.10(b) depict the program* P^{rel} *and its PDG* G^{rel}_{pdg} *which are obtained after computing semantic relevancy w.r.t. SIGN. Observe that in P (Figure 8.9(a)) the statement at program point 6 is irrelevant w.r.t. SIGN. Therefore, we can remove the conditional statement at 5 and the* ϕ *statement, because after removing statement 6 the corresponding "if" block becomes semantically irrelevant and the SSA function* f *is not necessary anymore, as x has just a single definition. The DCG annotations over the DDG edges of* G^{rel}_{pdg} *are shown in Figure 8.10(c).*

Figure 8.10: A program and satisfiability of its DCG after relevancy computation

(a) P^{rel}: Program after relevancy computation of P *w.r.t.* SIGN

```
1.  start
2.  x = input;
3.  y = input;
4.  if(x > 0){
7.        print(x);}
8.  stop
```

(b) G^{rel}_{pdg}: PDG of P^{rel}



(c) DCG annotations for the edges of G^{rel}_{pdg}

e	e^R	e^A
$2 \xrightarrow{x} 7$	$1 \xrightarrow{true} 4 \xrightarrow{true} 7$	$\emptyset$
$2 \xrightarrow{x} 4$	$\emptyset$	$\emptyset$

After removing the irrelevant entries from η and ψ, we get the dependence path $\eta' = 2 \xrightarrow{x} 7$, and the execution trace ψ' as follows:

$$\psi' = \iota_1 : \langle 1, (\bot, \bot)\rangle\ \iota_2 : \langle 2, (\bot, \bot)\rangle\ \iota_3 : \langle 3, (+, \bot)\rangle\ \iota_4 : \langle 4, (+, +)\rangle\ \iota_8 : \langle 7, (+, +)\rangle\ \iota_9 : \langle 8, (+, +)\rangle$$

Now, let us show that $\psi' \vdash^{SIGN} \eta'$.

In η', for the edge $2 \xrightarrow{x} 7$, the statement at 7 does not post-dominate the statement at 2. The Reach Sequences and the Avoid Sequences for the edge $e = 2 \xrightarrow{x} 7$ are $(2 \xrightarrow{x} 7)^R = \{1 \xrightarrow{true} 4 \xrightarrow{true} 7\}$ and $(2 \xrightarrow{x} 7)^A = \emptyset$ respectively. For $1 \xrightarrow{true} 4 \xrightarrow{true} 7 \in (2 \xrightarrow{x} 7)^R$: $[\![1.cond = true]\!](\bot, \bot)$ and $[\![4.cond = true]\!](+, +)$ yields to true. Thus, $\psi' \vdash^R_{\iota_1} (2 \xrightarrow{x} 7)$.

The Avoid Sequence behaves similarly, yielding to $\psi' \vdash^{SIGN} \eta'$.

8.5.1 Refinement into Semantics-based Abstract DCG

Given a DCG, we can refine it into more precise semantics-based abstract DCG by removing from it all the semantically unrealizable dependences where conditions for a control dependence never be satisfiable or data defined at a source node can never be reachable to a target node in all possible abstract execution traces. The notion of semantically unrealizable dependence path is defined in Definition 38.

Definition 38 (Semantically Unrealizable Dependence Path) *Given a DCG G_{dcg} and an abstract domain ρ. A dependence path $\eta \in G_{dcg}$ is called semantically unrealizable in the*

abstract domain ρ if $\forall\psi$: $\psi \nvdash^{\rho} \eta$, where ψ is an abstract execution trace.

Figure 8.11: Algorithm to generate Semantics-based Abstract DCG

Algorithm 8: REFINE-DCG

Input: Syntactic DCG G_{dcg} and an abstract domain ρ
Output: Semantics-based abstract DCG G^{s}_{dcg} *w.r.t.* ρ

1. `FOR` each nodes $q \in G_{dcg}$ `DO`
2. If $\forall\psi$: $\psi \nvdash^{\rho} (p \xrightarrow{lab} q)$ where $lab \in \{true, false\}$ `THEN`
3. Remove from G_{dcg} the node q and all its associated dependences. If q is a control node, the removal of q also removes all the nodes transitively control-dependent on it. Data dependences have to be re-adjusted accordingly;
4. `END IF`
5. `FOR` each data dependence edge $e = (q \xrightarrow{x} p_i)$ `DO`
6. `IF` $\forall\psi$: $\psi \nvdash^{\rho} e$ `THEN`
7. Remove e from G_{dcg} and re-adjust the data dependence of p_i for the data x;
8. `END IF`
9. `END FOR`
10. FLAG:=true;
11. `FOR` each ϕ-sequences $\eta_{\phi} = (q \xrightarrow{x} \phi_1 \xrightarrow{x} \ldots \xrightarrow{x} \phi_j \xrightarrow{x} p_i)$ starting from q `DO`
12. `IF` $\exists\psi$: $\psi \vdash^{\rho} \eta_{\phi}$ `THEN`
13. FLAG:=false;
14. BREAK;
15. `END IF`
16. `END FOR`
17. `IF` FLAG==true `THEN`
18. Remove the edge $q \xrightarrow{x} \phi_1$;
19. `END IF`
20. `END FOR`

The refinement algorithm of a DCG from syntactic to semantic one is depicted in Figure 8.11. Step 2 says that if the condition on which a node q is control-dependent, never be satisfied by any of the execution traces, then the node and all its associated dependences are removed. In that case, if q is a control node, we remove all the nodes transitively control-dependent on q. If any DDG edge with q as source is semantically unrealizable under its abstract semantics, the corresponding DDG edge is removed in step 5. If all the ϕ-sequences emerging from q are semantically unrealizable under its abstract semantics, we remove the dependence of the ϕ-sequences on q in step 11.

Figure 8.12: Slicing Algorithm

Algorithm 9: GEN-SLICE

Input: Program P and an abstract slicing criterion $\langle p, v, \rho \rangle$
Output: Abstract Slice *w.r.t.* $\langle p, v, \rho \rangle$

1. Generate a semantics-based abstract PDG $G^{r,d}_{pdg}$ from the program P by following the algorithm `REFINE-PDG`.
2. Convert $G^{r,d}_{pdg}$ into the corresponding DCG G_{dcg} by computing annotations over all the data/control edges of it.
3. Generate a semantics-based abstract DCG G^{s}_{dcg} from G_{dcg} by following the algorithm `REFINE-DCG`.
4. Apply the criterion $\langle p, v \rangle$ on G^{s}_{dcg} by following PDG-based slicing techniques (156) and generate a sub-DCG G_{sdcg} that includes the node corresponding to the program point p as well.
5. Refine G_{sdcg} into more precise one G'_{sdcg} by performing the following operation for all nodes $q \in G_{sdcg}$:

   $\forall \eta_\phi = (q \xrightarrow{x} \phi_1 \xrightarrow{x} \ldots \xrightarrow{x} \phi_j \xrightarrow{x} p_i)$ and $\forall \psi$: if $\psi \not\vdash^{\rho} \eta_\phi$, then remove the edge $q \xrightarrow{x} \phi_1$ from G_{sdcg}.

6. Apply again the criterion $\langle p, v \rangle$ on G'_{sdcg} that results into the desired slice.

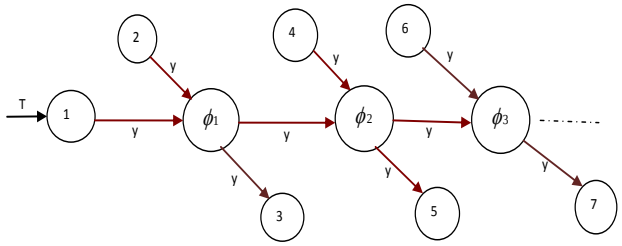
Observe that in case of static slicing the satisfiability of the dependence paths are checked against all possible traces of the program, whereas in case of dynamic slicing or other forms of slicing the checking is performed against the traces generated only for the inputs of interest.

8.6 Slicing Algorithm

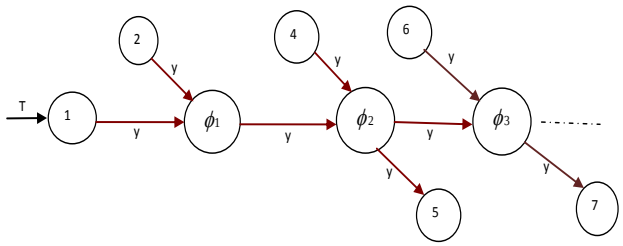
We are now in position to formalize our proposed slicing algorithm, depicted in Figure 8.12, that takes a program P and an abstract slicing criterion $\langle p, v, \rho \rangle$ as inputs, and produces an abstract slice *w.r.t.* $\langle p, v, \rho \rangle$ as output. The proposed slicing algorithm make use of the semantics-based abstract DCG of the program that is obtained in two steps: first by generating semantics based abstract PDG by following the algorithm `REFINE-PDG` (Figure 8.8), and then by converting it into semantics-based abstract DCG

Figure 8.13: Refinement of sub-DCG during slicing

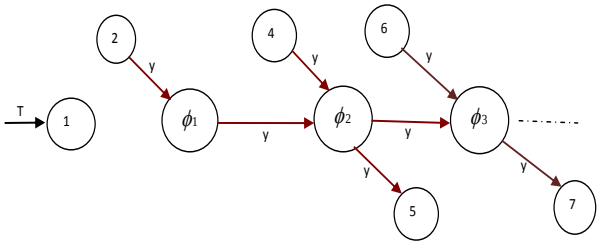
(a) A part of a DCG containing three ϕ-sequences



(b) Sub-DCG after applying Slicing Criterion C



(c) Refiement of Sub-DCG



by following the algorithm `REFINE-DCG` (Figure 8.11).

Observe that the sub-DCG G_{sdcg} which is obtained in step 4 by applying slicing criterion on the semantics-based abstract DCG G^s_{dcg}, is further refined in step 5 by removing unrealizable data dependences, if present, from it. Let us illustrate the reason behind it with an example. Consider the graph in Figure 8.13(a) showing a portion of DCG with three ϕ-sequences ϕ_1, ϕ_2 and ϕ_3 that describe the data dependences of the nodes 3, 5 and 7 respectively on the node 1 for a data y.

- $\eta_1 = 1 \xrightarrow{y} \phi_1 \xrightarrow{y} 3$
- $\eta_2 = 1 \xrightarrow{y} \phi_1 \xrightarrow{y} \phi_2 \xrightarrow{y} 5$
- $\eta_3 = 1 \xrightarrow{y} \phi_1 \xrightarrow{y} \phi_2 \xrightarrow{y} \phi_3 \xrightarrow{y} 7$

Suppose, $\exists\psi$: $\psi \vdash^{\rho} \eta_1 \bigwedge \forall\psi$: $\psi \nvdash^{\rho} (\eta_2 \bigwedge \eta_3)$. During refinement of a DCG in the

algorithm REFINE-DCG, we can not remove the dependence edge $1 \xrightarrow{y} \phi_1$ because there is one semantically realizable ϕ-sequence ϕ_1 from node 1.

Given a slicing criterion C. In algorithm GEN-SLICE, suppose the sub-DCG generated after applying C does not include the node 3, as depicted in Figure 8.13(b). Now if we apply step 5 on the sub-DCG, we see that all the ϕ-sequences emerging from node 1 (ϕ_2 and ϕ_3) are not semantically realizable. Therefore, we can remove the edge $1 \xrightarrow{y} \phi_1$ from it as depicted in Figure 8.13(c). The further application of the slicing criterion C (in step 6) on this refined sub-DCG generate a slice that does not include the statement corresponding to the node 1 any more.

8.7 Illustration of the Proposal with an Example

In this section, we illustrate the proposed slicing technique with an example by showing the combination of all the phases of computation (statement relevancy, semantic data dependences, conditional dependences and slicing) step by step. We show that our proposal results into a more precise slice (according to def. 21) than the one proposed by Mastroeni and Zanardini (140).

Example 31 *Consider the program P and the corresponding traditional Program Dependence Graph (G_{pdg}) for its SSA correspondent code, as depicted in Figure 8.14.*

Suppose, we are interested only in the sign of the program variables and the abstract domain SIGN is represented by SIGN $= \{\bot, +, 0, -, 0^+, 0^-, \top\}$ *where* 0^+ *denotes* $\{x \in Z : x \geq 0\}$, 0^- *denotes* $\{x \in Z : x \leq 0\}$, *and Z is the set of integers. Consider the abstract slicing criterion* $\langle 13, y, SIGN\rangle$ *where* 13 *is the program point and y is the variable used at* 13.

Computation of Statement Relevancy. *At program point* 9, *the variable x may have any abstract value from the set* $\{+, 0, -\}$. *Since the abstract evaluation of the assignment statement* $x_3 = x_2 \times 2$ *at* 9 *does not change the sign property of x, the statement at* 9 *is irrelevant w.r.t. SIGN. After disregarding the node corresponding to this statement from the syntactic PDG* G_{pdg}, *we get a refined semantics-based abstract PDG* G^r_{pdg} *depicted in Figure 8.15.*

Computation of Semantic Data Dependences. *The computation of semantic data dependences (140) for all expressions in* P_{ssa} *w.r.t. SIGN reveals the fact that there is no semantic data dependence between* y_4 *and w at statement 12, as "*$4w$ *mod* 2*" always yields to* 0. *Therefore, by disregarding the corresponding data dependence edge* $5 \xrightarrow{w} 12$ *from* G^r_{pdg} *obtained in previous phase, we get a more refined semantics-based abstract PDG* $G^{r,d}_{pdg}$ *depicted in Figure 8.16.*

Figure 8.14: A program and its traditional Program Dependence Graph (PDG)

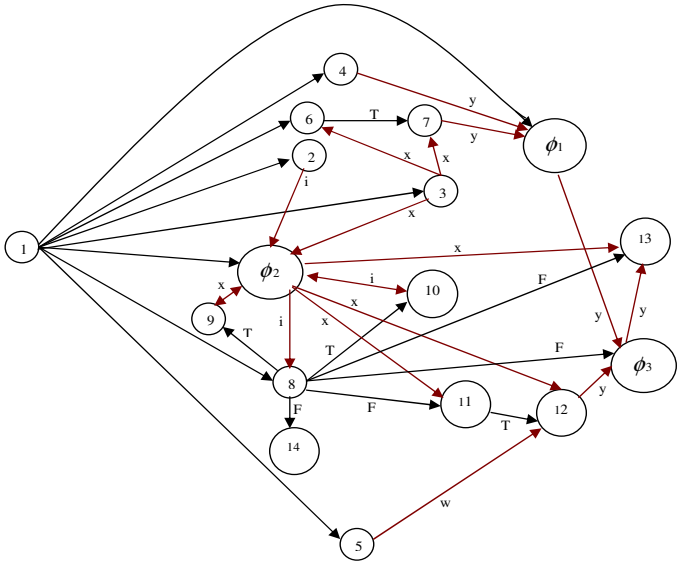
(a) Program P

```
1.    start
2.    i = −2;
3.    x = input;
4.    y = input;
5.    w = input;
6.    if(x ≥ 0)
7.          y = 4x^3;
8.    while(i ≤ 0){
9.          x = x × 2;
10.         i = i + 1;}
11.   if(x ≤ 0)
12.         y = x^2 + 4w mod 2;
13.   print(x, y);
14.   stop
```

(b) P_{ssa}: SSA form of P

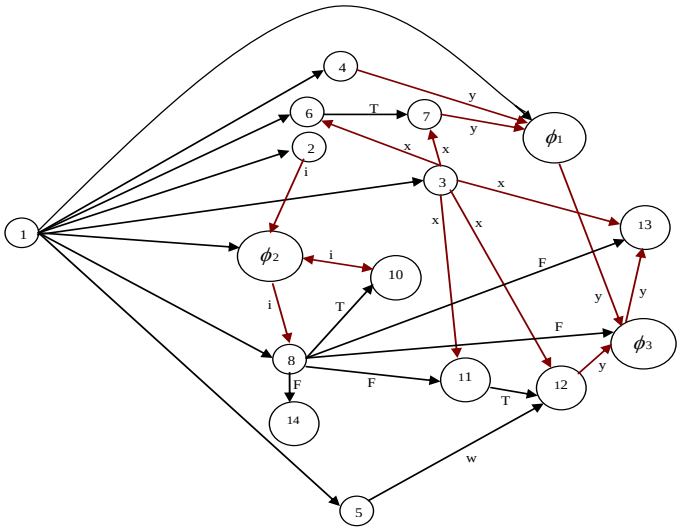
```
1.    start
2.    i1 = −2;
3.    x1 = input;
4.    y1 = input;
5.    w = input;
6.    if(x1 ≥ 0)
7.          y2 = 4 × (x1)^3;
φ1    y3 = f(y1, y2);
      while(
φ2    (x2, i2) = f((x1, i1), (x3, i3));
8.    i2 ≤ 0
      ){
9.          x3 = x2 × 2;
10.         i3 = i2 + 1;}
11.   if(x2 ≤ 0)
12.         y4 = (x2)^2 + 4w mod 2;
φ3    y5 = f(y3, y4);
13.   print(x2, y5);
14.   stop
```

(c) G_{pdg}: PDG of P_{ssa}



Computation of Conditional Dependences. *Given a semantics-based abstract PDG* $G^{r,d}_{pdg}$ *obtained so far, we can easily convert it into the corresponding DCG* G_{dcg} *depicted in Figure 8.17(a). Let us consider the node 4 and its associated* ϕ*-sequence* $\eta_\phi = 4 \xrightarrow{y} \phi_1 \xrightarrow{y} \phi_3 \xrightarrow{y} 13$ *in* G_{dcg} *representing the flow of definition at 4 to 13. Since*

Figure 8.15: G^r_{pdg}: PDG after relevancy computation of P_{ssa} *w.r.t.* SIGN



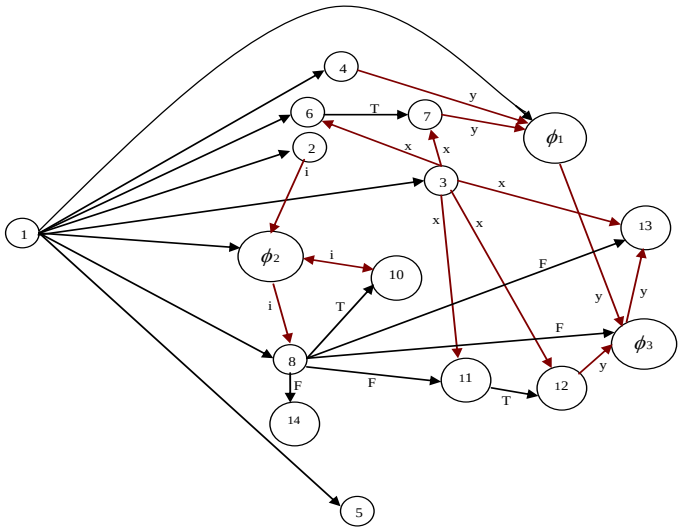
the abstract values of x may have any value from the set $\{+, 0, -\}$ *at program point 3, there is no such execution trace* ψ *over the abstract domain SIGN that can avoid both* $(4 \xrightarrow{y} \phi_1)^A = \{1 \xrightarrow{true} 6 \xrightarrow{true} 7\}$ *and* $(\phi_1 \xrightarrow{y} \phi_3)^A = \{1 \xrightarrow{true} 8 \xrightarrow{false} 11 \xrightarrow{true} 12\}$ *simultaneously. For all execution traces over the abstract domain of sign, at least one of the conditions among* $6 \xrightarrow{true} 7$ *and* $11 \xrightarrow{true} 12$ *must be satisfied. This means that the definition at 4 is over-written either by 7 or by 12 or by both, and can never reach 13. In short,* η_ϕ *is semantically unrealizable i.e.* $\forall \psi : \psi \not\vdash^{SIGN} \eta_\phi$.

Therefore, if we execute the algorithm `REFINE-DCG` *on* G_{dcg}, *since there exists no semantically realizable* ϕ*-sequence from the node 4 to any target node t such that y defined at 4 can reach t, we remove the edge* $4 \xrightarrow{y} \phi_1$ *from* G_{dcg}, *resulting into a more precise semantics-based abstract DCG* G^s_{dcg} *as depicted in Figure 8.17(b).*

Slicing w.r.t. ⟨13, y, SIGN⟩. *Given the semantics-based abstract DCG* G^s_{dcg} *in Figure 8.17(b). We apply PDG-based backward slicing technique (156) on it w.r.t.* $\langle 13, y \rangle$ *and generate a sub-DCG* G_{sdcg} *as depicted in Figure 8.18(a). Observe that we can not refine it anymore. Therefore, slicing of* G_{sdcg} *again w.r.t.* $\langle 13, y \rangle$ *yields a slice shown in Figure 8.18(b).*

Observe that if we apply only the slicing technique of Mastroeni and Zanadini (140) on the program P, we get the slice depicted in Figure 8.19. According to the definition 21, the slice obtained in Figure 8.18(b) is more precise than the Mastroeni and Zanadini's one obtained in Figure 8.19.

Figure 8.16: $G^{r,d}_{pdg}$: PDG after computing statements relevancy first, and then semantic data dependences of P_{ssa} *w.r.t.* SIGN



8.8 Soundness and Complexity Analysis

In this section, we prove that the abstract semantic relevancy computation is sound, and we perform the complexity analysis of the proposed slicing technique.

8.8.1 Semantic Relevancy: Soundness

When we lift the semantics-based program slicing from the concrete domain to an abstract domain, we are loosing some information about the states occurring at different program points in P. Thus, some relevant statements at the concrete level may be treated as irrelevant in an abstract domain as they do not have any impact on the property observed through the abstract domains.

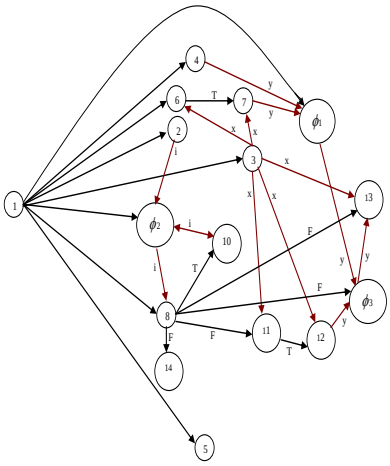
In order to prove the soundness of the abstract semantic relevancy of statements, we need to show that if any statement s at program point p in the program P is irrelevant *w.r.t.* an abstract property ρ, then the execution of s over all the concrete states possibly reaching p does not change the property ρ of the variables in those concrete states.

Theorem 5 (Soundness) *If a statement s at program point p in the program P is semantically irrelevant w.r.t. an abstract property ρ, then s is semantically irrelevant with respect to the concrete property ω defined by: $\omega \triangleq \forall \sigma \in \Sigma_p,\ \forall x_i \in VAR\ :\ \rho(\sigma[x_i]) = \rho((S[\![s]\!]\sigma)[x_i])$.*

Proof *Given an abstract domain ρ on values, the set of abstract states is denoted by Σ^ρ whose elements are tuples $\epsilon = \langle \rho(v_1), \ldots, \rho(v_k) \rangle$ where $v_i = \sigma(x_i)$ for $x_i \in VAR$ being the set of*

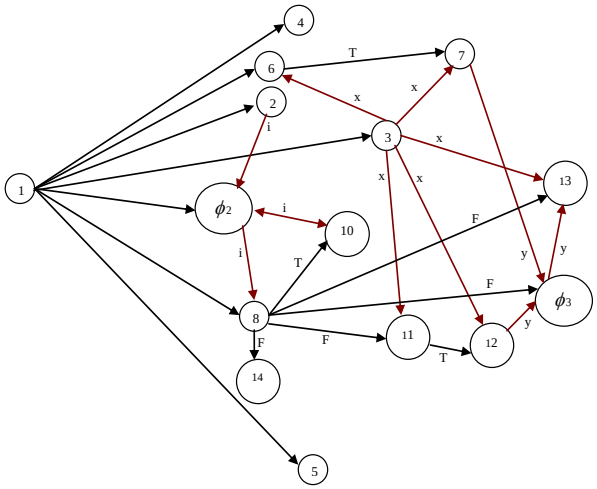
Figure 8.17: Semantics-based abstract DCG

(a) G_{dcg}: DCG after computing annotations on $G^{r,d}_{pdg}$



e	e^R	e^A
$4 \xrightarrow{y} \phi_1$	$\emptyset$	$1 \xrightarrow{true} 6 \xrightarrow{true} 7$
$7 \xrightarrow{y} \phi_1$	$\emptyset$	$\emptyset$
$3 \xrightarrow{x} 6$	$\emptyset$	$\emptyset$
$3 \xrightarrow{x} 7$	$1 \xrightarrow{true} 6 \xrightarrow{true} 7$	$\emptyset$
$3 \xrightarrow{x} 11$	$1 \xrightarrow{true} 8 \xrightarrow{false} 11$	$\emptyset$
$3 \xrightarrow{x} 12$	$1 \xrightarrow{true} 8 \xrightarrow{false} 11 \xrightarrow{true} 12$	$\emptyset$
$3 \xrightarrow{x} 13$	$1 \xrightarrow{true} 8 \xrightarrow{false} 13$	$\emptyset$
$12 \xrightarrow{y} \phi_3$	$\emptyset$	$\emptyset$
$\phi_1 \xrightarrow{y} \phi_3$	$1 \xrightarrow{true} 8 \xrightarrow{false} \phi_3$	$1 \xrightarrow{true} 8 \xrightarrow{false} 11 \xrightarrow{true} 12$
$\phi_3 \xrightarrow{y} 13$	$\emptyset$	$\emptyset$
$2 \xrightarrow{i} \phi_2$	$\emptyset$	$\emptyset$
$\phi_2 \xrightarrow{i} 8$	$\emptyset$	$\emptyset$
$10 \xrightarrow{i} \phi_2$	$\emptyset$	$\emptyset$
$\phi_2 \xrightarrow{i} 10$	$1 \xrightarrow{true} 8 \xrightarrow{true} 10$	$\emptyset$

(b) G^s_{dcg} : DCG after removing $e = 4 \xrightarrow{y} \phi_1$ from G_{dcg}

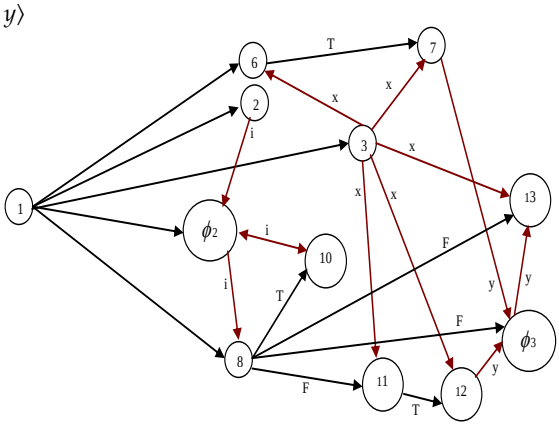


program variables.

Let $\sigma = \langle v_1, \ldots, v_k \rangle \in \Sigma$ and $\epsilon = \langle \rho(v_1), \ldots, \rho(v_k) \rangle \in \Sigma^\rho$. Observe that since $\forall x_i \in VAR$: $\sigma(x_i)$ is a singleton and ρ is partitioning, each variable x_i will have the atomic property obtained from the induced partition $\Pi(\rho)$ (140). The concretization of the abstract state ϵ is represented

Figure 8.18: Slicing *w.r.t.* $\langle 13, y, SIGN\rangle$

(a) G_{sdcg}: sub-DCG after performing backward slicing on G^{s}_{dcg} *w.r.t.* $\langle 13, y\rangle$



(b) Slice *w.r.t.* $\langle 13, y\rangle$ computed from G_{sdcg}

```
1.    start
2.    i = −2;
3.    x = input;
6.    if(x ≥ 0)
7.          y = 4x^3;
8.    while(i ≤ 0){
10.         i = i + 1;}
11.   if(x ≤ 0)
12.         y = x^2 + 4w mod 2;
```

by $\gamma(\epsilon) = \{\langle u_1, \ldots, u_k\rangle \mid \forall i.\, u_i \in \rho(v_i)\}$. *We denote the* j^{th} *concrete state in* $\gamma(\epsilon)$ *by the notation* $\langle u_1, \ldots, u_k\rangle^j$ *and we denote by* u_i^j *the elements of that tuple.*

As $S[\![s]\!]^\rho(\epsilon)$ *is defined as the best correct approximation of* $S[\![s]\!]$ *on the concrete states in* $\gamma(\epsilon)$*, we get:*

$$
\begin{aligned}
S[\![s]\!]^\rho(\epsilon) &= \rho\Big(\bigcup_{j\in[1..|\gamma(\epsilon)|]} \{S[\![s]\!]\langle u_1, \ldots, u_k\rangle^j\}\Big) \\
&= \rho\Big(\bigcup_{j\in[1..|\gamma(\epsilon)|]} \{\langle u'_1, \ldots, u'_k\rangle^j \mid S[\![s]\!]\langle u_1, \ldots, u_n\rangle^j = \langle u'_1, \ldots, u'_k\rangle^j\}\Big) \\
&= \langle\, \rho\Big(\bigcup_{j\in[1..|\gamma(\epsilon)|]} \{u'^j_1\}\Big), \ldots, \rho\Big(\bigcup_{j\in[1..|\gamma(\epsilon)|]} \{u'^j_k\}\Big)\,\rangle
\end{aligned}
$$

where u'^j_i *denotes the concrete value of the* i^{th} *variable* $x_i \in VAR$ *in the state obtained after the*

Figure 8.19: Slice *w.r.t.* $\langle 13, y, SIGN \rangle$ by Mastroeni and Zanardini

```
1.    start
2.    i = −2;
3.    x = input;
4.    y = input;
6.    if(x ≥ 0)
7.          y = 4x^3;
8.    while(i ≤ 0){
9.          x = x × 2;
10.         i = i + 1;}
11.   if(x ≤ 0)
12.         y = x^2 + 4w mod 2;
```

execution of the statement s over the j^{th} concrete state in $\gamma(\epsilon)$. Observe that the later equality relies on the distributivity of ρ, that comes from the assumption of the atomicity of abstract domain obtained from induced partitioning.

From the definition of abstract irrelevancy of a statement s at program point p w.r.t. abstract property ρ, we get

$$\forall \epsilon \in \Sigma_p^\rho : \;\; S[\![s]\!]^\rho(\epsilon) = \epsilon$$

Therefore,

$$S[\![s]\!]^\rho(\epsilon) = \langle\, \rho(\bigcup_{j\in[1..|\gamma(\epsilon)|]} \{u_1'^j\}), \ldots, \rho(\bigcup_{j\in[1..|\gamma(\epsilon)|]} \{u_k'^j\}) \,\rangle = \epsilon$$

Then, by def. of ϵ,

$$\langle\, \rho(\bigcup_{j\in[1..|\gamma(\epsilon)|]} \{u_1'^j\}), \ldots, \rho(\bigcup_{j\in[1..|\gamma(\epsilon)|]} \{u_k'^j\}) \,\rangle = \langle\, \rho(v_1), \ldots, \rho(v_k) \,\rangle \tag{8.1}$$

And so, by def. of $\gamma(\epsilon)$ we get:

$$\langle\, \rho(\bigcup_{j\in[1..|\gamma(\epsilon)|]} \{u_1'^j\}), \ldots, \rho(\bigcup_{j\in[1..|\gamma(\epsilon)|]} \{u_k'^j\}) \,\rangle = \langle\, \rho(\bigcup_{j\in[1..|\gamma(\epsilon)|]} \{u_1^j\}), \ldots, \rho(\bigcup_{j\in[1..|\gamma(\epsilon)|]} \{u_k^j\}) \,\rangle \tag{8.2}$$

We already mentioned that given an abstract property ρ, since $\forall x_i \in VAR : \sigma(x_i)$ is a singleton and ρ is a partitioning, each variable x_i will have the property obtained from the induced partition $\Pi(\rho)$ (140). Thus, $\forall x_i \in VAR : \; \rho(\sigma(x_i)) = \rho(v_i)$ is atomic.

Therefore, from the Equations 1 and 2, we get

$$\forall x_i \in VAR,\; \rho(\bigcup_{j\in[1..|\gamma(\epsilon)|]} \{u_i'^j\}) = \rho(v_i) = \rho(\bigcup_{j\in[1..|\gamma(\epsilon)|]} \{u_i^j\}) \text{ is atomic.}$$

This allows us to conclude that for each i^{th} program variables $x_i \in VAR$ (where $i \in [1...k]$) in all the j^{th} concrete states (where $j \in [1..|\gamma(\epsilon)|]$), the concrete value $u_i'^j$ which is obtained after

the execution of s over those concrete states have the same property as the concrete value u_i^j *before the execution of s. This means that for any irrelevant statement s at program point p in P w.r.t. abstract property* ρ*, the execution of s over the concrete states possibly reaching p does not lead to any change of the property* ρ *of the concrete values of the program variables* $x_i \in VAR$ *in those concrete states.*

Thus, s is semantically irrelevant w.r.t. the concrete property $w \triangleq \forall \sigma \in \Sigma_p,\ \forall x_i \in VAR\ :\ \rho(\sigma[x_i]) = \rho((S[\![s]\!]\sigma)[x_i])$.

8.8.2 Complexity Analysis

Given an abstract domain ρ, our proposal has the following four subsequence steps to obtain abstract slice *w.r.t.* a slicing criterion *C*:

1. Compute semantic relevancy of program statements *w.r.t.* ρ.
2. Obtain semantic data dependences of each expression on the variables appearing in it *w.r.t.* ρ.
3. Generation of semantics-based abstract DCG by removing all the unrealizable dependences *w.r.t.* ρ.
4. Finally, slice the semantics-based abstract DCG *w.r.t. C*.

8.8.2.1 Complexity in computing semantic relevancy

To compute semantic relevancy of a statement *s* at program point *p w.r.t.* ρ, we compare each abstract state $\epsilon \in \Sigma_p^\rho$ with the corresponding state $\epsilon' = S[\![s]\!]^\rho(\epsilon)$. If $\forall \epsilon \in \Sigma_p^\rho : \epsilon = \epsilon'$, we say that *s* is irrelevant *w.r.t.* ρ.

To obtain all possible abstract states reaching each program point in a program, we compute its abstract collecting semantics.

Complexity to compute abstract collecting semantics. Given a set of abstract states Σ^ρ and a set of program points *Label*, the context vector is defined by *Context-Vector*$^\sharp$: *Label* $\rightarrow$ *Context*$^\sharp$, where *Context*$^\sharp = \wp(\Sigma^\rho)$.

The context vector associated with a program *P* of size *n* is, thus, denoted by $Cv_P^\sharp = \langle Cx_1^\sharp, Cx_2^\sharp, \ldots, Cx_n^\sharp \rangle$, where $Cx_i^\sharp$ is the context associated with program point *i* in *P*.

Let $F_i^\sharp : \mathit{Context\text{-}Vector}^\sharp \rightarrow \mathit{Context}^\sharp$ be a collection of abstract monotone functions. For the program P, we therefore have

$$\begin{aligned} Cx_1^\sharp &= F_1^\sharp(Cx_1^\sharp, \ldots, Cx_n^\sharp) \\ Cx_2^\sharp &= F_2^\sharp(Cx_1^\sharp, \ldots, Cx_n^\sharp) \\ &\ldots\ldots \\ Cx_n^\sharp &= F_n^\sharp(Cx_1^\sharp, \ldots, Cx_n^\sharp) \end{aligned}$$

Combining the above abstract functions, we get

$$F^\sharp : \mathit{Context\text{-}Vector}^\sharp \rightarrow \mathit{Context\text{-}Vector}^\sharp$$

That is,

$$F^\sharp(Cx_1^\sharp, \ldots, Cx_n^\sharp) = (F_1^\sharp(Cx_1^\sharp, \ldots, Cx_n^\sharp), \ldots, F_n^\sharp(Cx_1^\sharp, \ldots, Cx_n^\sharp))$$

Each function $F_i^\sharp$ includes the transition function defined as follows:

$$Cx_i^\sharp = \bigcup_{s_j \in pred(s_i)} \cup_{\epsilon_j \in Cx_j^\sharp} S[\![s_j]\!]^\rho(\epsilon_j) \tag{8.3}$$

where, $pred(s_i)$ is the set of predecessors of the statement s_i.

Starting from the initial context vector $Cv_P^\sharp = \langle \bot, \ldots, \bot \rangle$ which is the bottom element of the lattice L^n where $L = (\wp(\Sigma^\rho), \sqsubseteq, \sqcap, \sqcup, \top, \bot)$, the computation of least fix-point of $F^\sharp$ results into the collecting semantics for P. With this collecting semantics, we can easily obtain the abstract states possibly reaching each program points in a program that help in computing semantic relevancy of all statements *w.r.t.* ρ.

Equation 8.3 says that the time complexity of each $F_i^\sharp$ depends on the no. of predecessors of s_i and the execution time of $S[\![.]\!]^\rho$, assuming the no. of possible abstract states appearing at each program point as constant. For "skip" statement, $S[\![skip]\!]^\rho$ is constant, whereas for assignment/conditional/repetitive statements, it depends on the execution time for arithmetic and boolean expressions occurred in those statements. Theoretically, there is no limit of the length of expressions *i.e.* the no. of variables/constants/operations present in the expressions. However, practically, we assume that β is the maximum no. of operations (arithmetic or boolean) that can be present in any expression. Assuming the time needed to perform each operation as constant, we get the time complexity of $S[\![.]\!]^\rho$ as $O(\beta)$. Since in a control flow graph the no. of predecessors of each s_i is constant, and $F^\sharp$ involves n monotone functions, the time complexity of $F^\sharp$ is $O(n\beta)$, where n is the no. of statements in the program.

The least solution for $F^\sharp$ depends on the number of iteration performed to reach the fix-point. In case of finite height lattice, let h be the height of the context-lattice $L = (\wp(\Sigma^\rho), \sqsubseteq, \sqcap, \sqcup, \top, \bot)$. The height of L^n is, thus, nh which bounds the number of iteration we perform to reach the fix-point. So the time complexity for $Fix(F^\sharp)$ is $O(n^2\beta h)$.

However, for the lattice with infinite height, a widening operation is mandatory (45) and the overall complexity of $Fix(F^\sharp)$ depends on it.

Complexity to compute statements semantic relevancy. Once we obtain the collecting semantics for a program P in an abstract domain ρ, the time complexity to compute semantic relevancy of each statement depends only on the comparison between the abstract states in the contexts associated with it and in the corresponding contexts of its successors. Any change in the abstract states determines its relevancy *w.r.t.* ρ. For a program with n statements, the time complexity to compute semantic relevancy is, thus, $O(n)$.

8.8.2.2 Complexity in computing semantic data dependences

Mastroeni and Zanardini (140) introduced an algorithm to compute semantic data dependences of an expression on the variables appearing in it. Before discussing the complexity, we briefly mention the algorithm.

Given an expression e and an abstract state ϵ, the atomicity condition $A_e^U(\epsilon)$ holds iff execution of e over ϵ *i.e.* $E[\![e]\!]^\rho(\epsilon)$ results an atomic abstract value U, or there exists a covering $\{\epsilon_1, \ldots, \epsilon_k\}$ of ϵ such that $A_e^U(\epsilon_i)$ holds for every i.

In order to compute semantic data dependences of an expression e on the variables $var(e)$ appearing in it, the algorithm calls a recursive function with $X = var(e)$ as parameter. The recursive function uses an assertion $A'_e(\epsilon, X)$, where ϵ is an abstract state possibly reaching the statement containing the expression e. The assertion $A'_e(\epsilon, X)$ holds iff $\exists U : A_e^U(\epsilon)$, or there exists an X-covering $\{\epsilon_1, \ldots, \epsilon_k\}$ of ϵ such that $\forall i : A'_e(\epsilon_i, X)$. Intuitively, X-covering is a set of restriction on a state, which do not involve X. If $A'_e(\epsilon, X)$ holds, it implies the non-relevance of X in the computation of e, otherwise for each $x \in X$ the same is repeated with $X \backslash x$ as parameter.

Thus, the time complexity to compute semantic data dependences at expression level for the whole program depends on the following factors:

- The time complexity of $E[\![e]\!]^\rho$: Theoretically there is no limit of the length of expression e *i.e.* the no. of variables/constants/operations present in e. However,

practically, we assume that β is the maximum no. of operations (arithmetic or boolean) that can be present in e. Assuming the time needed to perform each operation as constant, we get the time complexity of $E[\![e]\!]^{\rho}$ as $O(\beta)$.

- The time complexity of the atomicity condition $A_e^U(\epsilon)$: In worst case, the time complexity of $A_e^U(\epsilon)$ depends on the time complexity of $E[\![e]\!]^{\rho}$ and the no. of elements in the covering of ϵ. Let m be the no. of atomic values in the abstract domain ρ. Since the no. of elements in a covering depends on the no. of atomic values in the abstract domain, the time complexity of $A_e^U(\epsilon)$ is $O(m\beta)$.

- The time complexity of the assertion $A'_e(\epsilon, X)$: In worst case, the time complexity of $A'_e(\epsilon, X)$ depends on the time complexity of atomicity condition $A_e^U(\epsilon)$ and the no. of elements in the X-covering of ϵ. Thus, the time complexity of $A'_e(\epsilon, X)$ is $O(m^2\beta)$.

In worst case, the recursive function that uses $A'_e(\epsilon, X)$ executes for all subset of variables appearing in e *i.e.* $\forall X \in \wp(var(e))$. So, it depends on the set of program variables VAR. Therefore, the time-complexity of the recursive function is $O(m^2\beta\text{VAR})$, where VAR is the set of program variables. For a program P of size n, the no. of expressions that can occur in worst case is n. Thus, finally we get the time complexity to compute semantic data dependences for a program P of size n is $O(m^2\beta n\text{VAR})$.

8.8.2.3 Complexity to generate semantics-based abstract DCG and its slicing

Given a program P (in IMP language) and its PDG, the time complexity to construct DCG from a PDG is $O(n)$ (182), where n is the no. of nodes in the PDG. However, to obtain semantics-based abstract DCG G^s_{dcg} from a syntactic DCG G_{dcg}, our algorithm removes all the unrealizable dependences present in G_{dcg}.

To do this, the algorithm checks the satisfiability of the annotations of all the outgoing DDG edges, incoming CDG edge and outgoing ϕ-sequences associated with each node in the DCG against all the abstract execution traces.

For a DCG with n nodes, the maximum no. of edges need to check is $O(n^2)$. Thus, in case of lattice of finite height h, the worst case time complexity to verify all dependences for their satisfiability against abstract execution traces is $O(n^3h)$.

As we know that the slicing which is performed by walking a DCG backwards or forwards from the node of interest takes $O(n)$ (156), the worst case time complexity to obtain DCG-based slicing is, therefore, $O(n^3h)$.

8.8.2.4 Overall complexity of the proposal

We may assume that the number of atomic values (m) present in an abstract domain is constants, and that $O(\beta) = O(\text{VAR}) = O(n)$.

As an overall complexity evaluation of the techniques presented so far, we can say that it has worst case time complexity, in case of finite height abstract lattices, is $O(n^3h))$, where n is the no. of statements in the program and h is the height of the lattice of context.

8.9 Discussions

We acknowledge that there are other possible improvements that deserve to be considered in future. As for instance, the semantic relevancy at statement-level does not take into account the semantic interaction between statements. For example, if consider a block consisting of two statements $\{y = y + 3;\ y = y - 1;\}$, we observe that each of the two statements is not semantically irrelevant *w.r.t.* PAR, while the block as a whole is irrelevant *w.r.t.* PAR.

Therefore, to be more precise, we should start to compute the relevancy of a program from block-level to statement-level. If any block is irrelevant, we disregard all statements in that block; otherwise, we compute relevancy for all sub-blocks of that block. In this way, we compute the relevancy by moving from block-level towards the statement-level. Instead, we can also use the partial evaluation technique, although costlier, to resolve this issue. For instance, the above two statements can be replaced by a single statement y=y+2 which is irrelevant *w.r.t.* PAR.

In (140), the problem related to the control dependence is not addressed. For example, consider the following example:

```
4.   ....
5.   if ((y + 2x mod 2) == 0) then
6.          w=5;
7.   else w=5;
8.   ....
```

Here the abstract semantic data dependence says that the condition in "if" statement is only dependent on y. But it does not say anything about the dependence between w and y.

Observe that although w is invariant *w.r.t.* the evaluation of the guard, this is not captured by (140).

The block-level semantic relevancy, rather than statement-level, can resolve this issue of independences. Let us denote the complete "if-*else*" block by s. The semantics

of s says that $\forall\sigma_1, \sigma_2 \in \Sigma_5$, $S[\![s]\!](\sigma_1) = \sigma_1'$ and $S[\![s]\!](\sigma_2) = \sigma_2'$ implies $\sigma_1' = \sigma_2'$, where $\sigma_1'(w) = \sigma_2'(w) = 5$. It means that there is no control of the "if-$else$" over the resultant state which is invariant. So we can replace the whole conditional block s by the single statement $w = 5$. Notice that this is also true if we replace the statement at line 6 by w=y+5, as the line 6 is executed when y==0.

The combined result of semantic relevancy, semantic data dependences and conditional dependences in the refinement of the PDGs can be applied to all forms of slicing: static, dynamic, conditional, amorphous etc. Since the allowed initial states are different for different forms of slicing, we compute statements relevancy and semantic data dependences of expressions over all the possible states reaching the program points in the program by starting only from the allowed initial states, according to the criterion. Similarly, the satisfiability of the depenedence paths in the DCG are checked against the traces generated from the allowed initial states only. For instance, in case of conditional slicing, a condition is specified in the slicing criterion to disregard some initial states that do not satisfy it. In case of dynamic slicing, inputs of interest are specified in the slicing criterion. Therefore, the collecting semantics and execution traces are generated based on the allowed initial states specified or satisfying the conditions in the criterion, and are used to compute statements relevancy, semantic data dependences and satisfiability of the dependence paths.

Although program slicing is a program analysis technique and is not directly related to the Information System scenario, but indirectly slicing can help in security aspect, in particular information flow security analysis. The combination of results on the refinement of dependence graphs with static analysis techniques discussed in this paper may give rise to further interesting applications to enhance the accuracy of the static analysis and for accelerating the convergence of the fixed-point computation. This is the topic of our ongoing research.

8.10 Slicing of Database Applications

The presence of SQL data manipulation language operations such as SELECT, INSERT, UPDATE or DELETE in data-intensive applications that access or manipulate databases, requires the extension of traditional Program Dependence Graphs (PDGs) into Database-Oriented Program Dependence Graphs (DOPDGs), where two additional types of dependences, called Program-Database (PD) Dependences and Database-Database (DD) Dependences need to be considered (194). A PD-Dependence arises between a SQL statement and an imperative statement where either the database

state defined by SQL statement is used by the imperative statement or the database state defined by imperative statement is used by the SQL statement. A DD-Dependence arises between two SQL statements where the database state defined by one SQL statement is used by the other SQL statement. The data dependences between imperative statements are similar as in the case of traditional PDGs.

In this section, we extend the three notions (statement relevancy, semantic data dependeces and conditional dependences) to the context of programs embedding SQL statements, and we propose a refinement of DOPDG-based slicing algorithms thereof.

In particular, we provide the following two contributions:

- We define syntax-based DOPDG by expressing the conditions of DD- and PD-Dependences in denotational form, as an alternative to the rules introduced by Baralis and Widom (9).

- We propose a refinement of syntax-based DOPDGs into semantics-based abstract DOPDGs by combining the notions of (*i*) semantic relevancy of statements, (*ii*) semantic data dependences, and (*iii*) conditional dependences.

These two contributions lead to an abstract program slicing algorithm for programs embedding SQL statements that strictly improves with respect to the literature.

8.10.1 A Motivating Example

All the proposed slicing techniques for programs embedding SQL statements (33, 183, 194) are syntax-based, and are computed over concrete domains of data. Slicing of programs embedding SQL statements in an abstract domain helps in finding a subset of statements that affect some particular properties of the data in the database or in the applications. Moreover, the computation of semantics-based dependences removes some false dependences, yielding to more precise slices. The following example illustrates an abstract slicing of a program embedding SQL statements, and we see how semantics-based dependence computations make the slice more precise.

Example 32 *Consider the database dB in Table 8.2 and a portion of code P depicted in Figure 8.20. It is clear from the code that the employees are promoted from lower rank to higher rank belonging to same job category. Finally, the code computes the average salary of employees under each of the groups with same job category.*

Although we follow the syntax of java programs embedding SQL statements, for the sake of simplicity we simplify the syntax for some statements, for instance, statements 3, 4, 5 that accept run-time input, and statement 12 that prints all values in the application variable rs.

Table 8.2: Database *dB*

(a) Table "citizen"

ID	name	age	locID	jobname
1	Alberto	28	2	Programmer
2	Matteo	30	1	HR
3	Francesco	35	3	Analyst

(b) Table "location"

locID	locname
1	Venice
2	Marghera
3	Mestre

(c) Table "job"

jobname	category	rank	sal
Programmer	A	1	1500
Analyst	A	2	1800
Project Manager	A	3	2100
Receptionist	B	1	1000
Secretary	B	2	1100
HR	B	3	2000

Suppose an auditing officer observes that the average salary of employees under the group of job category "A" (displaying at program point 12) is out of the expected range. For instance, suppose, according to the company policy, that the average salary for job category "A" should belong to the interval [1500, 2100], while the computed result is 800. Abstract program slicing, in such case, can help to identify the program statements that are responsible for this error. To do so, we consider an abstract slicing criterion $\langle 12, rs, Ival \rangle$*, where "rs" is the variable of interest at program point 12 and "Ival" is the domain of intervals representing the property of "rs".*

In order to compute an abstract slice of P w.r.t. $\langle 12, rs, Ival \rangle$ *we need to consider the abstract computation of the code over an abstract version of the database in an abstract domain of interest. The abstract database* $dB^\sharp$ *corresponding to the concrete database dB is depicted in Table 8.3, where some of the numeric values are abstracted by the elements from the domain of intervals and jobs are represented by the abstract domain ABSJOB=*$\{\bot$*, TechStaff, AdminStaff,* $\top\}$ *and* γ *is a concretization function such that*

Figure 8.20: Program P

```
1.  start;
2.  Statement myStmt = DriverManager.getConnection("jdbc:mysql://200.210.220.1:1114/demo",
    "scott", "tiger").createStatement();
3.  $empID=read();
4.  $oldJob=read();
5.  $newJob=read();
6.  ResultSet rs1=myStmt.executeQuery("SELECT category, rank FROM job WHERE job-
    name=$oldJob");
7.  ResultSet rs2=myStmt.executeQuery("SELECT category, rank FROM job WHERE job-
    name=$newJob");
8.  if(rs1.next() && rs2.next()){
9.      if(rs1.getString("category").equals(rs2.getString("category"))  &&
        rs2.getInt("rank") > rs1.getInt("rank")){
10.         myStmt.executeQuery("UPDATE citizen SET jobname=$newjob WHERE
            ID=$empID");}}
11. ResultSet rs = myStmt.executeQuery("SELECT avg(J.sal) FROM citizen C, job J WHERE
    C.jobname=J.jobname GROUP BY J.category");
12. display(rs);
13. stop;
```

$$\gamma(X) \triangleq \begin{cases} \emptyset \ \ if\ X = \bot \\ \{Programmer, Analyst, Project\ Manager\} \ \ if\ X = TechStaff \\ \{Receptionist, Secretary, HR\} \ \ if\ X = AdminStaff \\ \{Programmer, Analyst, Project\ Manager, Receptionist, Secretary, HR\} \ \ if\ X = \top \end{cases}$$

The abstract computation of P on $dB^\sharp$ says that statement 10 is semantically irrelevant w.r.t. ABSJOB, since the update operation on jobs by the statement 10 is performed within the same job category from lower to higher rank and it does not affect the property of jobs (represented by ABSJOB) at all. Therefore, although the statement 11 syntactically depends on the statement 10 due to the presence of "jobname" attribute in it, semantically there is no such dependence. The removal of irrelevant statement 10 also allows to disregard the statements 3 to 9 on which statement 10 depends. Therefore, the semantics-based abstract slice of P w.r.t. $\langle 12, rs, Ival \rangle$

Table 8.3: Abstract Database $dB^\sharp$

(a) Table "citizen$^\sharp$"

ID	name	age	locID	jobname
1	Alberto	28	2	TechStaff
2	Matteo	30	1	AdminStaff
3	Francesco	35	3	TechStaff

(b) Table "location$^\sharp$"

locID	locname
1	Venice
2	Marghera
3	Mestre

(c) Table "job$^\sharp$"

jobname	category	rank	sal
TechStaff	A	[1, 3]	[1500, 2100]
AdminStaff	B	[1, 3]	[1000, 2000]

and the relevant part of the database on which the slice performs its necessary computation are shown in Figures 8.21(a) and 8.21(b) respectively. This way the computation of statement relevancy removes some false dependences between program statements, resulting into more precise abstract slices.

In this example, we just emphasized the impact of abstract program slicing for programs embedding SQL statements by combining with it the notion of semantic relevancy of statements w.r.t. a property of interest. The impact of semantic data dependences and conditional dependences will be further described in the subsequent sections.

Let e, t and f be an expression, a database table and a function respectively. As in chapter 3, we use the following functions that will be used in the rest of this chapter:

- *const(e)* returns the constants appearing in e.
- *var(e)* returns the variables appearing in e.
- *attr(t)* returns the attributes associated with t.
- *dom(f)* returns the domain of f.
- *target(f)* returns a subset of dom(f) on which the application of f is restricted.

8.10.2 Database-Oriented Program Dependence Graph (DOPDG)

In this section, we express the conditions for DD-Depenedences based on the denotational semantics of the programs embedding SQL statements. This formulation can be seen as an alternative to the rules introduced by Baralis and Widom (9).

Figure 8.21: Slice and its relevant database part

(a) abstract slice *w.r.t.* $\langle 12, rs, Ival \rangle$ of P_{sql}

```
1.  start;
2.  Statement myStmt = DriverManager.getConnection("jdbc:mysql://200.210.220.1:1114/demo",
    "scott", "tiger").createStatement();
11. ResultSet rs = myStmt.executeQuery("SELECT avg(J.sal) FROM citizen C, job J WHERE
    C.jobname=J.jobname GROUP BY J.category");
12. display(rs);
```

(b) Part of the database *dB* relevant to the slice

Table "citizen"

jobname
Programmer
HR
Analyst

Table "job"

jobname	category	sal
Programmer	A	1500
Analyst	A	1800
Project Manager	A	2100
Receptionist	B	1000
Secretary	B	1100
HR	B	2000

8.10.2.1 Identifying DD-Dependences

The data dependence between two imperative statements I_1 and I_2 for a data x is denoted by $I_1 \xrightarrow{x} I_2$. Similarly, we denote a DD-Dependence between two SQL statements Q_1 and Q_2 by $Q_1 \xrightarrow{\Upsilon} Q_2$, where Υ is the part of the database information that is defined by Q_1 and subsequently used by Q_2. Below we describe how to determine Υ.

Any SQL statement involves variables from two distinct sets: application variables $\mathbb{V}_a$ and database variables (or attributes) $\mathbb{V}_d$. We define the following functions that extract from statements $\mathbb{I}$ the sets of database and application variables involved in those statements:

$$\mathtt{USE}^d_v : \mathbb{I} \to \mathbb{V}_d \qquad \mathtt{USE}^a_v : \mathbb{I} \to \mathbb{V}_a \qquad \mathtt{USE}_v : \mathbb{I} \to \mathbb{V}_d \cup \mathbb{V}_a$$

$$\mathtt{DEF}^d_v : \mathbb{I} \to \mathbb{V}_d \qquad \mathtt{DEF}^a_v : \mathbb{I} \to \mathbb{V}_a \qquad \mathtt{DEF}_v : \mathbb{I} \to \mathbb{V}_d \cup \mathbb{V}_a$$

Therefore, given a SQL statement $Q = \langle A, \phi \rangle \in \mathbb{I}$, the sets of used and defined variables in it are as follows:

$$\mathtt{USE}_v(Q) = \mathtt{USE}^d_v(Q) \cup \mathtt{USE}^a_v(Q) \qquad \mathtt{DEF}_v(Q) = \mathtt{DEF}^d_v(Q) \cup \mathtt{DEF}^a_v(Q)$$

The imperative statements $I \in \mathbb{I}$ do not involve database variables, and therefore, $\mathtt{USE}^d_v(I) = \emptyset = \mathtt{DEF}^d_v(I)$. Table 8.4 depicts the sets of defined and used variables for

various SQL statements based on the syntactic presence of variables in the statements.

Table 8.4: Sets of defined and used variables involved in SQL Statements

	SQL Statements	Sets of defined and used variables
Q_{select}	$= \langle A_{sel}, \phi \rangle$	$\texttt{DEF}_v(Q_{select}) = v_a$
	$= \langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_1,\ g(\vec{e})), \phi \rangle$	$\texttt{USE}_v(Q_{select}) = var(\phi) \cup var(e) \cup var(\phi_1) \cup var(\vec{x}) \cup var(e')$
Q_{insert}	$= \langle A_{ins}, \phi \rangle$	$\texttt{DEF}_v(Q_{insert}) = var(\vec{v_d})$
	$= \langle insert(\vec{v_d},\ \vec{e}), true \rangle$	$\texttt{USE}_v(Q_{insert}) = var(\vec{e})$
Q_{update}	$= \langle A_{update}, \phi \rangle$	$\texttt{DEF}_v(C_{update}) = var(\vec{v_d})$
	$= \langle update(\vec{v_d},\ \vec{e}), \phi \rangle$	$\texttt{USE}_v(C_{update}) = var(\vec{e}) \cup var(\phi)$
Q_{delete}	$= \langle A_{del}, \phi \rangle$	$\texttt{DEF}_v(Q_{delete}) = var(\vec{v_d})$
	$= \langle delete(\vec{v_d}), \phi \rangle$	$\texttt{USE}_v(Q_{delete}) = var(\phi)$

Consider a SQL statement $Q = \langle A, \phi \rangle$ where $target(Q) = t$, $\texttt{USE}^d_v(Q) = \vec{a}_{use} \subseteq attr(t)$ and $\texttt{DEF}^d_v(Q) = \vec{a}_{def} \subseteq attr(t)$. Suppose, according to the denotational semantics of Q, that $S[\![Q]\!](\rho_t, \rho_a) = (\rho_{t'}, \rho_{a'})$ where S is a semantic function (see, chapter 3). We define two functions $\mathfrak{A}_{use}$ and $\mathfrak{A}_{def}$ as follows:

$$\mathfrak{A}_{use}(Q, t) = \rho_{t \downarrow \phi}(\vec{a}_{use}) \tag{8.4}$$

$$\mathfrak{A}_{def}(Q, t) = \Delta(\rho_{t'}(\vec{a}_{def}), \rho_t(\vec{a}_{def})) \tag{8.5}$$

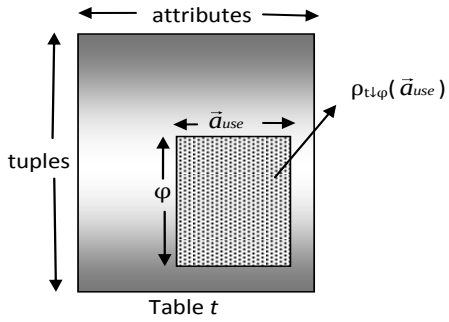
Given a SQL statement Q and its target table t, the function $\mathfrak{A}_{use}$ maps to the part of the database information accessed or used by Q depicted in Figure 8.22. We denote by the notation $(t \downarrow \phi)$ the set of tuples in t for which ϕ holds "*true*". The function $\mathfrak{A}_{def}$ defines the changes occurred in the database states when data is updated or deleted or inserted by SQL statements. Δ computes the differences between the original database states on which SQL statements operate and the new database states obtained after performing the SQL actions.

By following equations 8.4 and 8.5, we can compute the part of the database information Υ that is updated or deleted or inserted by $Q_1 = \langle A_1, \phi_1 \rangle$ and subsequently used by $Q_2 = \langle A_2, \phi_2 \rangle$ as follows:

$$\Upsilon = \mathfrak{A}_{use}(Q_2, t') \cap \mathfrak{A}_{def}(Q_1, t) \tag{8.6}$$

where $target(Q_1) = t$ and $S[\![Q_1]\!](\rho_t, \rho_a) = (\rho_{t'}, \rho_a)$ and $target(Q_2) = t'$. Observe that Q_1 can only be UPDATE or INSERT or DELETE statements and it does not change the application environment at all. In case of SELECT statement, $\Upsilon = \emptyset$ because $\mathfrak{A}_{def}(Q_{select}, t) = \emptyset$ where Q_{select} is a SELECT statement with $target(Q_{select}) = t$. Therefore, there exist no DD-Dependence on SELECT statements.

Figure 8.22: Part of database state used by $Q = \langle A, \phi \rangle$



Definition 39 *(DD-Dependences)*
Consider a SQL statement $Q_1 = \langle A_1, \phi_1 \rangle$ with $target(Q_1) = t$ such that $S[\![Q_1]\!](\rho_t, \rho_a) = (\rho_{t'}, \rho_a)$. The SQL statement $Q_2 = \langle A_2, \phi_2 \rangle$ with $target(Q_2) = t'$ is DD-Dependent on Q_1 for Υ (denoted $Q_1 \xrightarrow{\Upsilon} Q_2$) if $Q_1 \in \{Q_{update}, Q_{insert}, Q_{delete}\}$ and $\Upsilon = \mathfrak{A}_{use}(Q_2, t') \cap \mathfrak{A}_{def}(Q_1, t) \neq \emptyset$.

Below, we define $\mathfrak{A}_{def}$, $\mathfrak{A}_{use}$ and Υ by expressing the conditions for dependences in denotational form for different SQL statements.

UPDATE Statement. Consider a table t and an update statement

$$Q_{update} = \langle update(\vec{v_d}, \vec{e}), \phi \rangle$$

where $target(Q_{update}) = t$. According to the denotational semantics of Q_{update}, we get

$$S[\![Q_{update}]\!](\rho_t, \rho_a) = S[\![\langle update(\vec{v_d}, \vec{e}), \phi \rangle]\!](\rho_t, \rho_a) = (\rho_{t'}, \rho_a)$$

where,

$$\rho_{t'}(x) = \begin{cases} \rho_{t\downarrow\phi}(x) \cup \rho_{t\downarrow\neg\phi}(x) & \text{if } x \notin \vec{v_d} \\ E[\![e_i]\!](\rho_{t\downarrow\phi}, \rho_a) \cup \rho_{t\downarrow\neg\phi}(x) & \text{if } x \text{ is the } i^{th} \text{ component of } \vec{v_d} \text{ and } e_i \text{ is the } i^{th} \text{ component of } \vec{e} \end{cases}$$

Given an ordered set of database variables $\vec{x}$, we get

$$\rho_{t'}(\vec{x}) = \biguplus_{\forall x_i \in \vec{x}} \rho_{t'}(x_i) \tag{8.7}$$

The operator $\biguplus$ is defined as follows: Given two lists of elements M and N where $|M| = |N| = k$, then

$$M \biguplus N = \langle m_i \times n_i \mid m_i \in M,\ n_i \in N,\ i \in [1 \ldots k] \rangle \tag{8.8}$$

where $\times$ is the cartesian product.

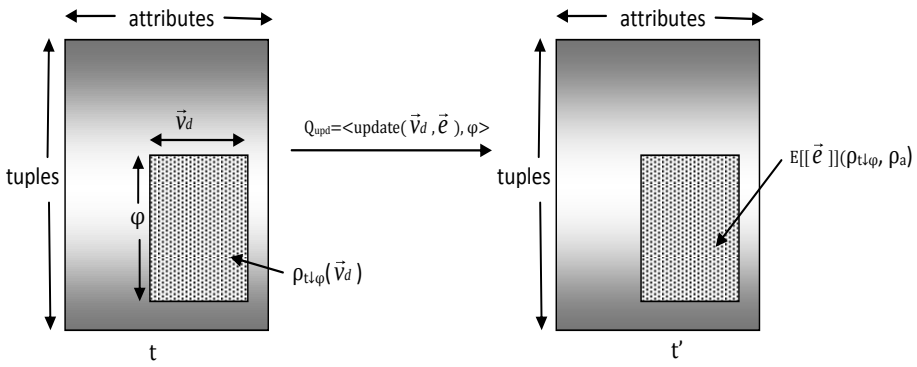
Area updated by Q_{update}: By following equation 8.5, we get the part of the database information updated by Q_{update} as

$$\mathfrak{A}_{def}(Q_{update}, t) = \Delta(\rho_{t'}(\vec{v_d}), \rho_t(\vec{v_d})) = E[\![\vec{e}]\!](\rho_{t\downarrow\phi}, \rho_a)$$

This fact is depicted in Figure 8.23.

Area used by other SQL statements: Consider a SQL statement $Q = \langle A, \phi_1 \rangle$ where

Figure 8.23: $\mathfrak{A}_{def}(Q_{update}, t)$ when updated by $Q_{update} = \langle update(\vec{v_d}, \vec{e}), \phi \rangle$



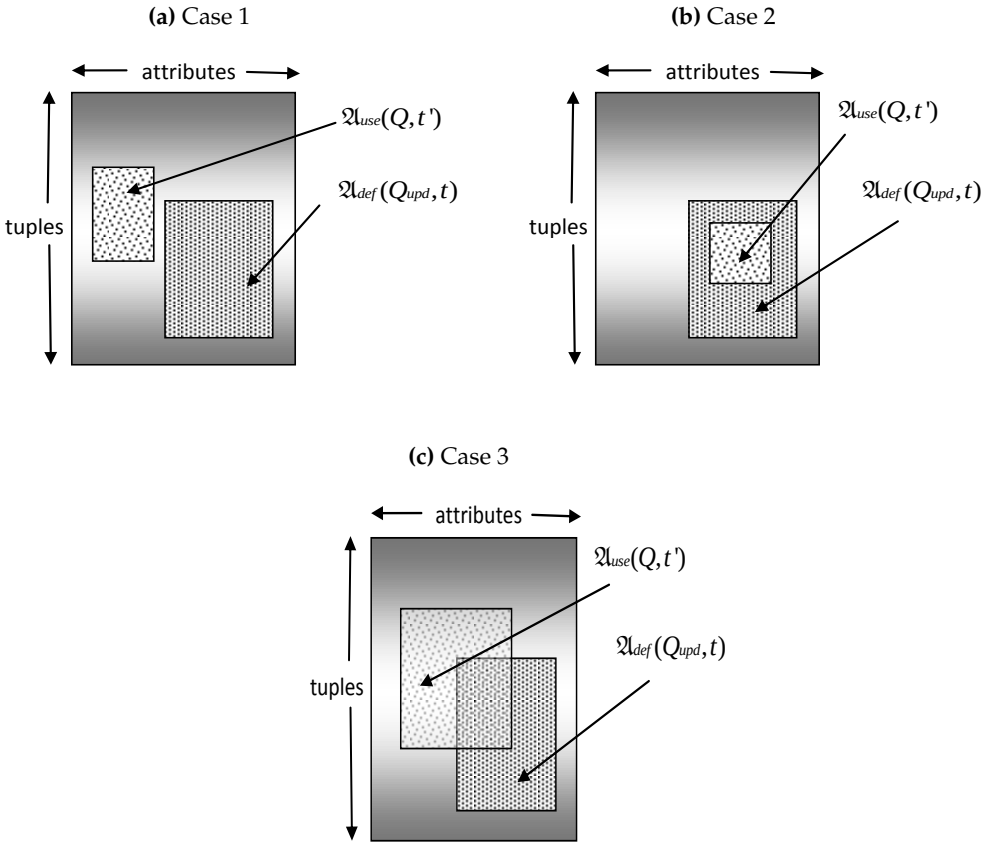
$target(Q) = t'$ and $\mathtt{USE}^d_v(Q) = \vec{x} \subseteq attr(t')$. From equations 8.4 and 8.7, we get the part of the data in table t' that Q is using as

$$\mathfrak{A}_{use}(Q, t') = \rho_{t'\downarrow\phi_1}(\vec{x})$$

When computing Υ we consider following three situations: (*i*) $\mathfrak{A}_{use}(Q, t')$ is not covered by $\mathfrak{A}_{def}(Q_{update}, t)$, (*ii*) $\mathfrak{A}_{use}(Q, t')$ is completely covered by $\mathfrak{A}_{def}(Q_{update}, t)$, and (*iii*) $\mathfrak{A}_{use}(Q, t')$ is partially covered by $\mathfrak{A}_{def}(Q_{update}, t)$. All these three cases are depicted in Figure 8.24(a), 8.24(b) and 8.24(c) respectively, and are formalized as follows:

$$\Upsilon = \mathfrak{A}_{use}(Q,t') \cap \mathfrak{A}_{def}(Q_{update},t) = \begin{cases} \textbf{Case 1} : Non-Dependence \\ \text{if } \rho_{t'\downarrow\phi_1}(\vec{x}) \nsubseteq E[\![\vec{e}]\!](\rho_{t\downarrow\phi},\rho_a) \\ \\ \textbf{Case 2} : Full-Dependence \\ \text{if } \rho_{t'\downarrow\phi_1}(\vec{x}) \neq \emptyset \text{ and } \rho_{t'\downarrow\phi_1}(\vec{x}) \subseteq E[\![\vec{e}]\!](\rho_{t\downarrow\phi},\rho_a) \\ \\ \textbf{Case 3} : Partial-Dependence \\ \text{Otherwise} \end{cases}$$

Figure 8.24: Various scenarios of DD-Dependences on Q_{update} when updates table t



INSERT Statement. Given a table t, according to the denotational semantics of INSERT statement $Q_{insert} = \langle insert(\vec{v_d},\vec{e}),\phi\rangle$ where $target(Q_{insert}) = t$, we get

$$S[\![Q_{insert}]\!](\rho_t,\rho_a) = S[\![\langle insert(\vec{v_d},\vec{e}),\phi\rangle]\!](\rho_t,\rho_a) = (\rho_{t'},\rho_a)$$

where,

$$\begin{aligned}&\text{let } \vec{v_d} = \langle a_1, a_2, \ldots, a_n\rangle = attr(t), \text{ and } E[\![\vec{e}]\!](\rho_a) = \vec{r} = \langle r_1, r_2, \ldots, r_n\rangle,\\ &\text{and } l_{new} = \langle r_1/a_1, r_2/a_2, \ldots, r_n/a_n\rangle, \text{ and } \rho_{t'}(\vec{x}) = \rho_{t \cup l_{new}}(\vec{x})\end{aligned} \tag{8.9}$$

Area inserted by Q_{insert}: The part of the data inserted by Q_{insert} into table t, by following equation 8.5, is

$$\mathfrak{A}_{def}(Q_{insert}, t) = \Delta(\rho_{t'}(\vec{v_d}), \rho_t(\vec{v_d})) = \rho_{l_{new}}(\vec{v_d})$$

Area used by other SQL statements: Consider a SQL statement $Q = \langle A, \phi_1\rangle$ where $target(Q) = t'$ and $\mathtt{USE}^d_v(Q) = \vec{x} \subseteq attr(t')$. From equations 8.4 and 8.9, we get the part of the data in table t' that Q is using as

$$\mathfrak{A}_{use}(Q, t') = \rho_{t' \downarrow \phi_1}(\vec{x})$$

The three situations of DD-Dependences on Q_{insert} are formalized as follows:

$$\Upsilon = \mathfrak{A}_{use}(Q, t') \cap \mathfrak{A}_{def}(Q_{insert}, t) = \begin{cases} \textbf{Case 1} : Non - Dependence \\ \text{if } \rho_{t' \downarrow \phi_1}(\vec{x}) \cap \rho_{l_{new}}(\vec{x}) = \emptyset \\ \\ \textbf{Case 2} : Full - Dependence \\ \text{if } \rho_{t' \downarrow \phi_1}(\vec{x}) \cap \rho_{l_{new}}(\vec{x}) = \rho_{t' \downarrow \phi_1}(\vec{x}) \\ \\ \textbf{Case 3} : Partial - Dependence \\ \text{Otherwise} \end{cases}$$

DELETE Statement Given a table t, according to the denotational semantics of DELETE statement $Q_{delete} = \langle delete(\vec{v_d}), \phi\rangle$ where $target(Q_{delete}) = t$, we get

$$S[\![Q_{delete}]\!](\rho_t, \rho_a) = S[\![\langle delete(\vec{v_d}), \phi\rangle]\!](\rho_t, \rho_a) = (\rho_{t'}, \rho_a)$$

where,

$$\rho_{t'}(\vec{x}) = \rho_{t \downarrow \neg \phi}(\vec{x}) \tag{8.10}$$

Area deleted by Q_{delete}: The part of the data deleted by Q_{delete} from table t, by following equation 8.5, is

$$\mathfrak{A}_{def}(Q_{delete}, t) = \Delta(\rho_{t'}(\vec{v_d}), \rho_t(\vec{v_d})) = \rho_{t \downarrow \phi}(\vec{v_d})$$

Area used by other SQL statements: Consider a SQL statement $Q = \langle A, \phi_1 \rangle$ where $target(Q) = t'$ and $\mathtt{USE}^d_v(Q) = \vec{x} \subseteq attr(t')$. From equations 8.4 and 8.10, we get the part of the data in table t' that Q is using as

$$\mathfrak{A}_{use}(Q, t') = \rho_{t' \downarrow \phi_1}(\vec{x})$$

The three situations of DD-Dependences on Q_{delete} are formalized as follows:

$$\Upsilon = \mathfrak{A}_{use}(Q, t') \cap \mathfrak{A}_{def}(Q_{delete}, t) = \begin{cases} \textbf{Case 1} : Non - Dependence \\ \text{if } \rho_{t\downarrow(\phi \wedge \phi_1)}(\vec{x}) = \emptyset \\ \\ \textbf{Case 2} : Full - Dependence \\ \text{if } \rho_{t\downarrow(\phi \wedge \phi_1)}(\vec{x}) = \rho_{t\downarrow\phi_1}(\vec{x}) \\ \\ \textbf{Case 3} : Partial - Dependence \\ \text{Otherwise} \end{cases}$$

We already mentioned that $\mathfrak{A}_{def}(Q_{select}, t) = \emptyset$, yielding $\Upsilon = \mathfrak{A}_{use}(Q, t') \cap \mathfrak{A}_{def}(Q_{select}, t) = \emptyset$, as no changes occurs when Q_{select} executes on t.

8.10.2.2 Identifying PD-Dependences

An imperative statement I is called *PD-Dependent* on a SQL statement Q for some variable x (denoted $Q \xrightarrow{x} I$) if I uses x whose value is obtained from the database by Q and there is an x-definition-free path from Q to I. For instance, consider the following java code interacting with database dB (Table 8.2) where the statements 6 and 7 are PD-Dependence on the SELECT statement at 5 for the variable x:

```
4.  ....
5.  ResultSet x = myStmt.executeQuery("SELECT name, age FROM citizen WHERE sal≥2000");
6.  while ( x.next() ) {
7.        System.out.println(x.getString("name")+x.getString("age"));}
8.  ....
```

Observe that the only SQL statements that can involve in PD-Dependences are SELECT statements, because only the SELECT statements are able to define the values of x by retrieving information from the databases which is then used by the imperative statement I.

Consider a SELECT statement $Q_{select} = \langle select(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_1,\ g(\vec{e})), \phi \rangle$ with $target(Q_{select}) = t$. From table 8.4, we get $\mathtt{DEF}_v(Q_{select}) = v_a$ where v_a is a resultset type of

application variable.

According to the semantics of Q_{select}, we get

$$S[\![Q_{select}]\!](\rho_t, \rho_a) = (\rho_t, \rho_{a'})$$

Therefore, the data for which an imperative statement is PD-Dependent on Q_{select} is $\rho_{a'}(v_a)$.

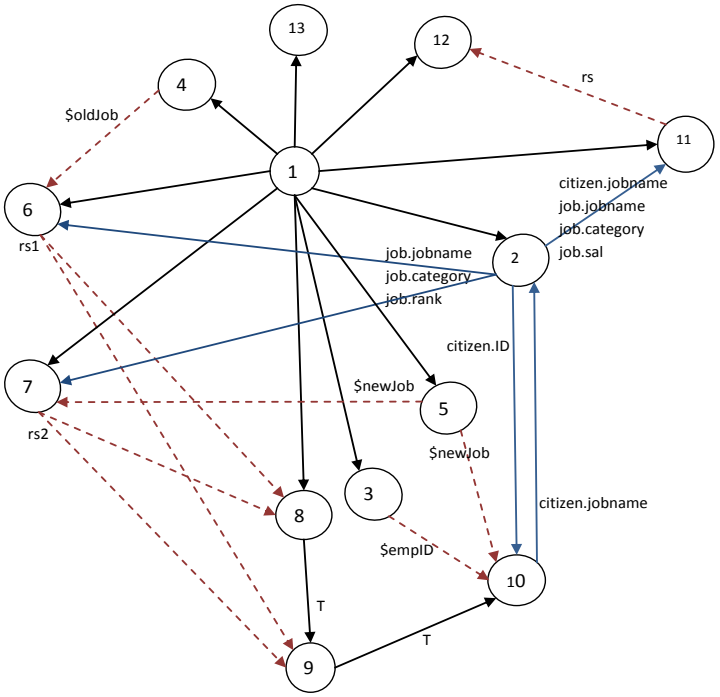
On the other hand, a SQL statement Q is called *PD-Dependent* on an imperative statement I for some variable x (denoted $I \xrightarrow{x} Q$) if Q uses x whose value is defined by I and there is an x-definition-free path from I to Q. In this case, Q can be either SELECT or UPDATE or INSERT or DELETE.

8.10.2.3 Constructing Concrete DOPDG

In previous sections, we discussed the extension of traditional Program Dependence Graphs (PDGs) into Database-Oriented Program Dependence Graphs (DOPDGs) by considering two additional types of data dependences: PD- and DD-Dependences that arise between SQL-SQL statements and SQL-imperative statements respectively. The computation of data dependences between two imperative statements and the computation of control dependences in DOPDGs are the same as in case of the traditional PDGs.

For instance, the syntactic DOPDG for the program P (Figure 8.20) is shown in Figure 8.25, where by the dotted and solid lines we denote the PD- and DD-Dependences respectively. Observe that when a SQL statement defines (*i.e.* updates or inserts or deletes) the database partially (*i.e.* a subset of database information corresponding to a subset of attributes or subset of tuples), we insert a DD-Dependence edge from the node corresponding to the defining statement to the node corresponding to the statement acting as a database source in the program. For instance, in Figure 8.25 we insert a DD-Dependence edge from node 10 to node 2, where the node 10 is an UPDATE statement (that defines the database partially) and the node 2 is treated as the source of the database as it makes a connection to the database. In contrast, when a SQL statement defines the whole database information, the defining statement will then be treated as the new source of the database information for the subsequent SQL statements.

Figure 8.25: DOPDG of the code *P*



8.10.3 Constructing Abstract DOPDG

In syntax-based DOPDGs, the PD-Dependences and the data dependences between imperative statements depend only on the syntactic presence of one variable in the definition of another variable or in a conditional expression. However, this is not true in case of DD-Dependences: DD-Dependences occur not only due to the syntactic presence of database attributes in SQL statements, but also when a part of the database information corresponding to these attributes defined by one SQL statement overlaps with the part of database information subsequently used by other SQL statement. We denoted this overlapping part by the notation Υ.

In order to compute abstract DD- and PD-Dependences in an abstract domain, we consider the notion of abstract databases, and we compute the part of the abstract database (denoted $\Upsilon^\sharp$) that is defined by one abstract SQL statement and subsequently used by other abstract SQL/imperative statement. This way we define abstract DOPDGs in an abstract domain of interest.

8.10.3.1 Abstract DD-Dependences

Let $Q^\sharp = \langle A^\sharp, \phi^\sharp \rangle$ be an abstract SQL statement with $target(Q^\sharp) = t^\sharp$ where $t^\sharp$ is an abstract table in an abstract domain. Suppose, according to the abstract semantics, $S^\sharp[\![Q^\sharp]\!](\rho_{t^\sharp}, \rho_{a^\sharp}) = (\rho_{t_1^\sharp}, \rho_{a_1^\sharp})$ where $S^\sharp$ is an abstract semantic function. We define the abstract functions $\mathfrak{A}^\sharp_{def}$ and $\mathfrak{A}^\sharp_{use}$ corresponding to $\mathfrak{A}_{def}$ and $\mathfrak{A}_{use}$ as follows:

$$\mathfrak{A}^\sharp_{use}(Q^\sharp, t^\sharp) = \rho_{t^\sharp \downarrow^\sharp_T \phi^\sharp}(\vec{a^\sharp}_{use}) \cup \rho_{t^\sharp \downarrow^\sharp_U \phi^\sharp}(\vec{a^\sharp}_{use}) \tag{8.11}$$

$$\mathfrak{A}^\sharp_{def}(Q^\sharp, t^\sharp) = \Delta^\sharp(\rho_{t_1^\sharp}(\vec{a^\sharp}_{def}), \rho_{t^\sharp}(\vec{a^\sharp}_{def})) \tag{8.12}$$

Observe that the notations $t^\sharp \downarrow^\sharp_T \phi^\sharp$ and $t^\sharp \downarrow^\sharp_U \phi^\sharp$ denote the set of abstract tuples in $t^\sharp$ for which $\phi^\sharp$ yields to "*true*" and "*unknown*" respectively in order to provide sound approximation in an abstract domain.

We now define the abstract version of the difference operation $\Delta^\sharp$ so as to provide a sound approximation of $\mathfrak{A}^\sharp_{def}$.

Abstract UPDATE Statement. Consider an abstract update statement

$$Q^\sharp_{update} = \langle update^\sharp(\vec{v^\sharp_d}, \vec{e^\sharp}), \phi^\sharp \rangle$$

Suppose, $target(Q^\sharp_{update}) = t^\sharp$ where $t^\sharp$ is an abstract table. According to the abstract semantics of $Q^\sharp_{update}$, we get

$$S^\sharp[\![Q^\sharp_{update}]\!](\rho_{t^\sharp}, \rho_{a^\sharp}) = S^\sharp[\![\langle update^\sharp(\vec{v^\sharp_d}, \vec{e^\sharp}), \phi^\sharp \rangle]\!](\rho_{t^\sharp}, \rho_{a^\sharp}) = (\rho_{t_1^\sharp}, \rho_{a^\sharp})$$

where,

$$\rho_{t_1^\sharp}(x^\sharp) = \begin{cases} \rho_{t^\sharp \downarrow^\sharp_T \phi^\sharp}(x^\sharp) \cup \rho_{t^\sharp \downarrow^\sharp_U \phi^\sharp}(x^\sharp) \cup \rho_{t^\sharp \downarrow^\sharp_F \phi^\sharp}(x^\sharp) & \text{if } x^\sharp \notin \vec{v^\sharp_d} \\ E^\sharp[\![e_i^\sharp]\!](\rho_{t^\sharp \downarrow^\sharp_T \phi^\sharp}, \rho_{a^\sharp}) \cup (\sqcup(E^\sharp[\![e_i^\sharp]\!](\rho_{t^\sharp \downarrow^\sharp_U \phi^\sharp}, \rho_{a^\sharp}), E^\sharp[\![x^\sharp]\!](\rho_{t^\sharp \downarrow^\sharp_U \phi^\sharp}))) \cup \rho_{t^\sharp \downarrow^\sharp_F \phi^\sharp}(x^\sharp) \\ \text{if } x^\sharp \text{ is the } i^{th} \text{ component of } \vec{v^\sharp_d} \text{ and } e_i^\sharp \text{ is the } i^{th} \text{ component of } \vec{e^\sharp} \end{cases}$$

By the notations $t^\sharp \downarrow^\sharp_T \phi^\sharp$, $t^\sharp \downarrow^\sharp_U \phi^\sharp$ and $t^\sharp \downarrow^\sharp_F \phi^\sharp$ we denote the set of abstract tuples in $t^\sharp$ for which $\phi^\sharp$ evaluates to *true*, *unknown* and *false* respectively. The operator $\sqcup$ stands for computing least upper bound component-wise, *i.e.* $\sqcup(X^\sharp, Y^\sharp) = \{lub(x_i^\sharp, y_i^\sharp) \mid x_i^\sharp \in$

$X^\sharp \wedge y_i^\sharp \in Y^\sharp\}$. Given an ordered set of abstract database variables $\vec{x^\sharp}$, we get

$$\rho_{t_1^\sharp}(\vec{x^\sharp}) = \biguplus_{\forall x_i^\sharp \in \vec{x^\sharp}} \rho_{t_1^\sharp}(x_i^\sharp)$$

where, the operator $\biguplus$ is defined in Equation 8.8.

The abstract difference operation $\Delta^\sharp$ in case of $Q^\sharp_{update}$ is defined as follows:

$$\begin{aligned}\mathfrak{A}^\sharp_{def}(Q^\sharp_{update}, t^\sharp) =& \Delta^\sharp(\rho_{t_1^\sharp}(\vec{v_d^\sharp}), \rho_{t^\sharp}(\vec{v_d^\sharp})) \\ =& E^\sharp[\![\vec{e^\sharp}]\!](\rho_{t^\sharp \downarrow^\sharp_T \phi^\sharp}, \rho_{a^\sharp}) \cup (\sqcup(E^\sharp[\![\vec{e^\sharp}]\!](\rho_{t^\sharp \downarrow^\sharp_U \phi^\sharp}, \rho_{a^\sharp}), E^\sharp[\![\vec{v_d^\sharp}]\!](\rho_{t^\sharp \downarrow^\sharp_U \phi^\sharp})))\end{aligned}$$

Abstract INSERT Statement. Given an abstract insert statement

$$Q^\sharp_{insert} = \langle insert^\sharp(\vec{v_d^\sharp}, \vec{e^\sharp}), \phi^\sharp \rangle$$

According to the abstract semantics of $Q^\sharp_{insert}$, we get

$$S^\sharp[\![Q^\sharp_{insert}]\!](\rho_{t^\sharp}, \rho_{a^\sharp}) = S^\sharp[\![\langle insert^\sharp(\vec{v_d^\sharp}, \vec{e^\sharp}), \phi^\sharp \rangle]\!](\rho_{t^\sharp}, \rho_{a^\sharp}) = (\rho_{t_1^\sharp}, \rho_{a^\sharp})$$

where,

$$\begin{aligned}&\text{let } \vec{v_d^\sharp} = \langle a_1^\sharp, a_2^\sharp, \ldots, a_n^\sharp \rangle = attr(t^\sharp), \text{ and } E^\sharp[\![\vec{e^\sharp}]\!](\rho_{a^\sharp}) = \vec{r^\sharp} = \langle r_1^\sharp, r_2^\sharp, \ldots, r_n^\sharp \rangle, \\ &\text{and } l^\sharp_{new} = \langle r_1^\sharp / a_1^\sharp, r_2^\sharp / a_2^\sharp, \ldots, r_n^\sharp / a_n^\sharp \rangle, \text{ and } \rho_{t_1^\sharp}(\vec{x^\sharp}) = \rho_{t^\sharp \cup l^\sharp_{new}}(\vec{x^\sharp})\end{aligned}$$

The abstract difference operation $\Delta^\sharp$ in case of $Q^\sharp_{insert}$ is defined as follows:

$$\mathfrak{A}^\sharp_{def}(Q^\sharp_{insert}, t^\sharp) = \Delta^\sharp(\rho_{t_1^\sharp}(\vec{v_d^\sharp}), \rho_{t^\sharp}(\vec{v_d^\sharp})) = \rho_{l^\sharp_{new}}(\vec{v_d^\sharp})$$

Abstract DELETE Statement. Given an abstract delete statement

$$Q^\sharp_{delete} = \langle delete^\sharp(\vec{v_d^\sharp}), \phi^\sharp \rangle$$

According to the abstract semantics of $Q^\sharp_{delete}$, we get

$$S^\sharp[\![Q^\sharp_{delete}]\!](\rho_{t^\sharp}, \rho_{a^\sharp}) = S^\sharp[\![\langle delete^\sharp(\vec{v_d^\sharp}), \phi^\sharp \rangle]\!](\rho_{t^\sharp}, \rho_{a^\sharp}) = (\rho_{t_1^\sharp}, \rho_{a^\sharp})$$

where, $\rho_{t_1^\sharp}(\vec{x^\sharp}) = \rho_{t^\sharp\downarrow^\sharp_{U'}\phi^\sharp}(\vec{x^\sharp}) \cup \rho_{t^\sharp\downarrow^\sharp_{F'}\phi^\sharp}(\vec{x^\sharp})$

The abstract difference operation $\Delta^\sharp$ in case of $Q^\sharp_{delete}$ is defined as follows:

$$\mathfrak{A}^\sharp_{def}(Q^\sharp_{delete}, t^\sharp) = \Delta^\sharp(\rho_{t_1^\sharp}(\vec{v_d^\sharp}), \rho_{t^\sharp}(\vec{v_d^\sharp})) = \rho_{t^\sharp\downarrow^\sharp_{T'}\phi^\sharp}(\vec{v_d^\sharp}) \cup \rho_{t^\sharp\downarrow^\sharp_{U'}\phi^\sharp}(\vec{v_d^\sharp})$$

Observe that we have $\mathfrak{A}^\sharp_{def}(Q^\sharp_{select}, t^\sharp) = \emptyset$, as there is no change in the abstract table $t^\sharp$ when $Q^\sharp_{select}$ executes on it. Lemma 9 depicts the soundness condition for SQL operations in abstract domains, according to definition 40.

Definition 40 *Let ϱ be an operation on a database table t. Let (α, γ) be a Galois Connection. Let $t^\sharp$ be an abstract table corresponding to t. An abstract operation $\varrho^\sharp$ on $t^\sharp$ is sound w.r.t. ϱ if*

$$\varrho(t) \in \gamma(\varrho^\sharp(t^\sharp))$$

Lemma 9 *Given a database table t and a concrete SQL statement $Q = \langle A, \phi\rangle$ with $target(Q) = t$. Let (α, γ) be a Galois Connection. Let $Q^\sharp = \langle A^\sharp, \phi^\sharp\rangle$ be an abstract SQL statement corresponding to Q and $t^\sharp$ be an abstract table corresponding to t. The abstract SQL statement $Q^\sharp$ is sound w.r.t. Q if*

$$\mathfrak{A}_{use}(Q, t) \in \gamma(\mathfrak{A}^\sharp_{use}(Q^\sharp, t^\sharp)) \text{ and } \mathfrak{A}_{def}(Q, t) \in \gamma(\mathfrak{A}^\sharp_{def}(Q^\sharp, t^\sharp))$$

From the abstract definition of $\mathfrak{A}^\sharp_{use}$ and $\mathfrak{A}^\sharp_{def}$, we can express the result into two distinct parts: *yes*-part and *may*-part, as follows:

$$\mathfrak{A}^\sharp_{use}(Q^\sharp, t^\sharp) = \xi^{use} = \langle \xi^{use}_{yes}, \xi^{use}_{may}\rangle$$

$$\mathfrak{A}^\sharp_{def}(Q^\sharp, t^\sharp) = \xi^{def} = \langle \xi^{def}_{yes}, \xi^{def}_{may}\rangle$$

where $\xi^{use}_{yes} \cap \xi^{use}_{may} = \emptyset$ and $\xi^{def}_{yes} \cap \xi^{def}_{may} = \emptyset$. The *yes*-part defines the portion for which $\phi^\sharp$ evaluates to "*true*", whereas the *may*-part defines the portions for which $\phi^\sharp$ evaluates to "*unknown*". Observe that in case of $Q^\sharp_{insert}$, the *may*-part $\xi^{def}_{may} = \emptyset$.

Given two abstract SQL statements $Q^\sharp_1$ and $Q^\sharp_2$ where $target(Q^\sharp_1) = t^\sharp$ and $S^\sharp[\![Q^\sharp_1]\!](\rho_{t^\sharp}, \rho_{a^\sharp}) = (\rho_{t_1^\sharp}, \rho_{a^\sharp})$ such that $target(Q^\sharp_2) = t^\sharp_1$. The part of the abstract database information $\Upsilon^\sharp$ defined by $Q^\sharp_1$ and subsequently used by $Q^\sharp_2$ is computed as follows:

$$\begin{aligned}
\Upsilon^\sharp =& \mathfrak{A}^\sharp_{use}(Q^\sharp_2, t^\sharp_1) \ \cap^\sharp \ \mathfrak{A}^\sharp_{def}(Q^\sharp_1, t^\sharp) \\
=& \xi^{use} \ \cap^\sharp \ \xi^{def} \\
=& \langle \xi^{use}_{yes}, \xi^{use}_{may}\rangle \ \cap^\sharp \ \langle \xi^{def}_{yes}, \xi^{def}_{may}\rangle \\
=& \langle (\xi^{use}_{yes} \cap \xi^{def}_{yes}), ((\xi^{use}_{may} \cap \xi^{def}) \cup (\xi^{def}_{may} \cap \xi^{use}))\rangle
\end{aligned}$$

Lemma 10 *Given two abstract SQL statements $Q_1^\sharp$ and $Q_2^\sharp$ that are sound w.r.t. their concrete counterpart Q_1 and Q_2 respectively. Suppose, $target(Q_1^\sharp) = t^\sharp$ where $t^\sharp$ is an abstract table and $S^\sharp[\![Q_1^\sharp]\!](\rho_{t^\sharp}, \rho_{a^\sharp}) = (\rho_{t_1^\sharp}, \rho_{a^\sharp})$ such that $target(Q_2^\sharp) = t_1^\sharp$. The combination $Q_2^\sharp(Q_1^\sharp(t^\sharp))$ is sound w.r.t. $Q_2(Q_1(t))$ if*

$$\mathfrak{A}_{use}(Q_2, t_1) \cap \mathfrak{A}_{def}(Q_1, t) \in \gamma(\mathfrak{A}_{use}^\sharp(Q_2^\sharp, t_1^\sharp) \ \cap^\sharp \ \mathfrak{A}_{def}^\sharp(Q_1^\sharp, t^\sharp))$$

where γ is the concretization function and $S[\![Q_1]\!](\rho_t, \rho_a) = (\rho_{t_1}, \rho_a)$.

The soundness of $Q_2^\sharp(Q_1^\sharp(t^\sharp))$ (Lemma 10) guarantees the soundness of abstract DD-Dependences in an abstract domain of interest. The three conditions for full, partial and non DD-dependences for $Q_1^\sharp \in \{Q_{update}^\sharp, Q_{insert}^\sharp, Q_{delete}^\sharp\}$ and $Q_2^\sharp \in \{Q_{select}^\sharp, Q_{update}^\sharp, Q_{insert}^\sharp, Q_{delete}^\sharp\}$ can be expressed similarly as defined in the concrete domain.

8.10.3.2 Abstract PD-Dependences

An abstract PD-Dependence is denoted by either $I^\sharp \xrightarrow{x^\sharp} Q^\sharp$ or $Q_{select}^\sharp \xrightarrow{x^\sharp} I^\sharp$, where $I^\sharp$, $Q^\sharp$ and $Q_{select}^\sharp$ represent abstract imperative statement, abstract SQL statement and abstract SELECT statement respectively.

Given an abstract domain, the abstract select statements $Q_{select}^\sharp$ is always sound *w.r.t.* its concrete counter-part Q_{select} (see, chapter 3). The soundness of $Q_{select}^\sharp$ guarantees the soundness of abstract PD-Dependences.

8.10.3.3 Semantics-based Dependences Computation

We already introduced the notion of semantic relevancy of imperative statements *w.r.t.* a property of interest, and we combined with it the notion of semantic data dependences and conditional dependences in order to provide a semantics-based computation to data and control dependences. In this subsection, we extend and integrate these notions to the context of programs embedding SQL statements.

Semantic Relevancy of SQL statements. When the execution of a statement does not change a property of the states possibly occurring at that statement, the statement is referred to as semantically irrelevant *w.r.t.* that property. The semantic relevancy of SQL statements can be defined with respect to concrete as well as abstract property, depicted in Definition 41 and 42 respectively.

Definition 41 *(Concrete Semantic Relevancy of SQL statements)*
Consider a concrete property $\omega = \langle \omega_{db}, \omega_{ap} \rangle$, where ω_{db} and ω_{ap} are the concrete properties on

the database variables and application variables respectively. A SQL statement $Q = \langle A, \phi \rangle$ at program point p is called semantically irrelevant w.r.t. ω, if for all states $\nu = (\rho_d, \rho_a)$ possibly appearing at p the execution of Q on ν results into a state $\nu' = (\rho_{d'}, \rho_{a'})$ such that ν and ν' are are equivalent w.r.t. ω, i.e.

$$\omega(\nu) \equiv \omega(\nu') \text{ or,}$$

$$\omega(\rho_d, \rho_a) \equiv \omega(\rho_{d'}, \rho_{a'}) \text{ or,}$$

$$\omega_{db}(\rho_d) \equiv \omega_{db}(\rho_{d'}) \wedge \omega_{ap}(\rho_a) \equiv \omega_{ap}(\rho_{a'})$$

Example 33 *Given a database containing salary information of the employees. Consider a concrete database property ω_{db} that expresses a constraint of gross salary on basic salary, defined as*

$$\textit{gross salary} = \frac{(165 \times \textit{basic salary})}{100} + 200$$

Let the initial database state satisfies ω_{db}. Let Q_1 and Q_2 be two SQL statements in a program. Suppose Q_1 updates basic salary, whereas Q_2 follows the above equation and updates the gross salary of employees. We say that Q_1 is semantically relevant w.r.t. ω_{db} as it changes the initial database state into a new state that does not satisfy ω_{db}. On the other hand, the entire block $\{Q_1; Q_2\}$ is semantically irrelevant w.r.t. ω_{db}, because its execution results into a state that preserves ω_{db}.

Definition 42 *(Abstract Semantic Relevancy of SQL statements)*
Given an abstract domain ϑ[1]. A SQL statement $\widetilde{Q} = \langle \widetilde{A}, \widetilde{\phi} \rangle$ with $target(\widetilde{Q}) = \widetilde{t}$ at program point p is called semantically irrelevant w.r.t. ϑ, if for all abstract states $\epsilon = (\rho_{\tilde{t}}, \rho_{\tilde{a}})$ possibly appearing at p the execution of $\widetilde{Q}$ on ϵ results into an abstract state $\epsilon' = (\rho_{\tilde{t}'}, \rho_{\tilde{a}'})$ such that they are equivalent i.e. $\epsilon \equiv \epsilon'$ w.r.t. ϑ.

Example 34 *Figure 8.3 depicts an abstract database where some of the numeric values are represented by the domain of intervals and jobs are represented by the abstract domain $ABSJOB = \{\bot, TechStaff, AdminStaff, \top\}$. Since the execution of the statement 10 in P (Figure 8.20) does not change the property of job information, we say that statement 10 is semantically irrelevant w.r.t. ABSJOB.*

The notion of semantic relevancy of statements allows us to refine syntactic DOPDGs into more precise semantics-based DOPDGs by removing the nodes corresponding to the irrelevant statements. For instance, removal of node 10 from the DOPDG of Figure 8.25 produces a refined semantics-based abstract DOPDG.

[1]For the sake of simplicity, we assume here that the attributes are of the same type, and that the property ϑ is the same for all attributes. The definition can easily be generalized to the case of lists of properties related to corresponding attributes.

Semantic Data Dependences. We can apply the notion of semantic data dependences to the case of programs embedding SQL statements to determine whether the presence of variables in an expression semantically affect the evaluation of the expression. Observe that expressions in SQL statements contain application variables as well as database attributes. The semantic data dependence computation for application variables appearing in the expressions has already been described in (140). However, we can extend this to the case of database attributes appearing in the expressions, resulting into the removal of corresponding PD- or DD-Dependences as well as the database information corresponding to these irrelevant attributes from the sliced database.

Example 35 *Consider the following code fragment, where statement 5 adds a new column to a table t and sets its default value to 0. Observe that the expression "oldcol+newcol" in the select statement at program point 6 does not semantically depend on the database variable "newcol", as the variable "newcol" does not affect the result of the expression for any states possibly reaching program point 6. Therefore, we can remove the corresponding DD-Dependence edge* $5 \xrightarrow{newcol} 6$ *from the syntactic DOPDG. Also the database information corresponding to the attribute "newcol" does not appear in the resultant sliced database.*

```
4.  ....
5.  ALTER TABLE t ADD COLUMN newcol int(10) NOT NULL DEFAULT 0;
6.  ResultSet rs = myStmt.executeQuery("SELECT oldcol+newcol FROM t");
7.  while ( rs.next() ) {
8.        System.out.println(rs.getString(1));}
9.  ....
```

Conditional Dependences. Given an abstract Database-Oriented Program Dependence Graph (DOPDG), we can convert it into an abstract Database-Oriented Dependence Condition Graph (DODCG) by computing annotations e^b over all dependence edges e, by following similar steps as in the case of PDG-to-DCG conversion.

The semantically unrealizable dependence paths (see Definition 43) in an abstract DODCG under the abstract trace semantics of the program removes some false dependences from the abstract DODCG, resulting into more precise semantics-based abstract DODCG.

Definition 43 *(Semantically Unrealizable Dependence Path)*
Given a DODCG G_{dodcg} *and an abstract domain* ϑ*. A dependence path* $\eta \in G_{dodcg}$ *is called semantically unrealizable in the abstract domain* ϑ *if* $\forall \psi$*:* $\psi \not\vdash^{\vartheta} \eta$*, where* ψ *is an abstract execution trace.*

As in the case of imperative programs, the combination of these three notions adopted for programs embedding SQL code can also lead to a slicing refinement algorithm where we use DOPDGs instead of PDGs. The cost of our proposal in case of programs embedding SQL code is $O(n^3h\gamma)$, where γ is the number of tuples in the database, n is the number of statements in the program, h is the height of the context lattice. Observe that the worst complexity is comparable with the complexity of usual dataflow analysis (like, liveness etc.), and that also in this case, in the practice we can get a quadratic complexity which is acceptable for its actual usability.

Chapter 9

Tukra: A Semantics-based Abstract Program Slicing Tool

In this section, we describe a program slicing tool, called **Tukra**[1], based on our proposal discussed in chapter 8. **Tukra** is implemented in Java language and can perform syntactic as well as semantics-based abstract slicing of programs in imperative languages *w.r.t.* a given criterion in an abstract domain of interest, by combining the notions of statements relevancy, semantic data dependences and conditional dependences. We execute it on a PC running with 2.27GHz Processor, Windows 7 Professional 64-bit Operating System and 4 GB RAM.

Tukra accepts the following inputs from the users:

1. **Program *P* to be sliced: Tukra** is able to perform slicing of programs in imperative languages. However, we have some assumptions on the syntax of the input programs as below:
   - All control blocks should be enclosed with "{}" irrespective of the number of statements in it.
   - Empty control block must have "skip;" statement in it.
   - Run-time Input for any statement is denoted by "?".
2. **Abstract Domain of Interest:** We provided two abstract domains in **Tukra** at this preliminary stage of implementation. The first one is "SIGN"=$\langle \top, +, -, 0, \bot \rangle$ that represents the sign property of variables and the second one is "PAR"=$\langle \top,$

[1] "Tukra" is a hindi word which means "Slice"

odd, even, $\perp$ $\rangle$ that represents the parity property of the variables. However, we, the programmer, can easily integrate additional abstract domains as required, by implementing the corresponding interfaces depicted in Figures 9.2, 9.3 and 9.4.

3. **Types of Semantic Computations: Tukra** can perform three types of semantic computations - statements relevancy, semantic data dependences and conditional dependences in the abstract domain chosen before. Users are provided options to choose either single or combination of multiple types of semantic computations. Moreover, the user can also perform syntax-driven slicing of the input programs.

4. **Slicing Criterion** C**:** A slicing criterion is composed of two components - a program variable v and a program point p where v is used or defined.

The main modules of **Tukra** are:

1. **ExtractInfo:** The module "ExtractInfo" extracts detail information about the input programs, *i.e*. the type of program statements, the controlling statements, the defined variables, the used variables etc for all statements in the program and store them in a file as an intermediate representation.

2. **FormMatrix:** The module "FormMatrix" generates incidence matrix for Control Flow Graphs (CFGs) and Program Dependence Graphs (PDGs) of the input programs based on the information extracted by the module "ExtractInfo".

3. **GenCollectingSemantics** and **GenTraceSemantics:** Given an abstract domain, these modules compute the abstract collecting Semantics and abstract trace semantics of the input programs based on the information extracted by "ExtractInfo" and the CFG generated by "FormMatrix".

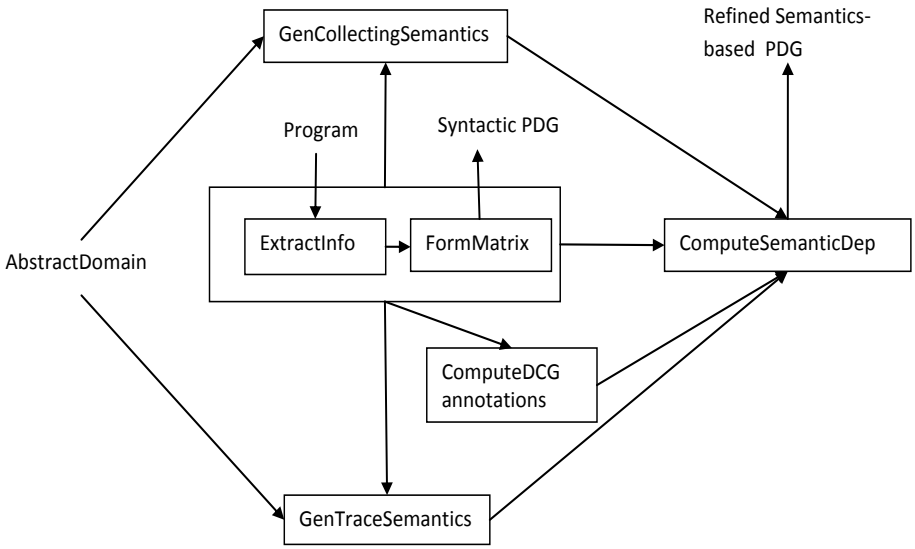
4. **ComputeDCGannotations:** This module computes the DCG annotations (Reach Sequences and Avoid Sequences) for all CDG and DDG edges of the PDG based on the information extracted by "ExtractInfo" and the PDG generated by "FormMatrix".

5. **ComputeSemanticDep:** It computes statements relevancy, semantic data dependences and conditional dependences of the programs under all possible states reaching each program points of the program (collecting semantics) and by computing the satisfiability of DCG annotations under its abstract trace semantics.

The interaction between different modules above is depicted in Figure 9.1. Observe that "FormMatrix" generates a syntactic PDG based on which we can perform syntax-driven slicing *w.r.t*. a criterion, whereas "ComputeSemanticDep" refines the syntactic

PDG into a semantics based abstract PDG by computing statements relevancy and semantic data dependences under its abstract collecting semantics and conditional dependences based on the satisfiability of DCG annotations under its abstract trace semantics.

Figure 9.1: Interaction between various modules

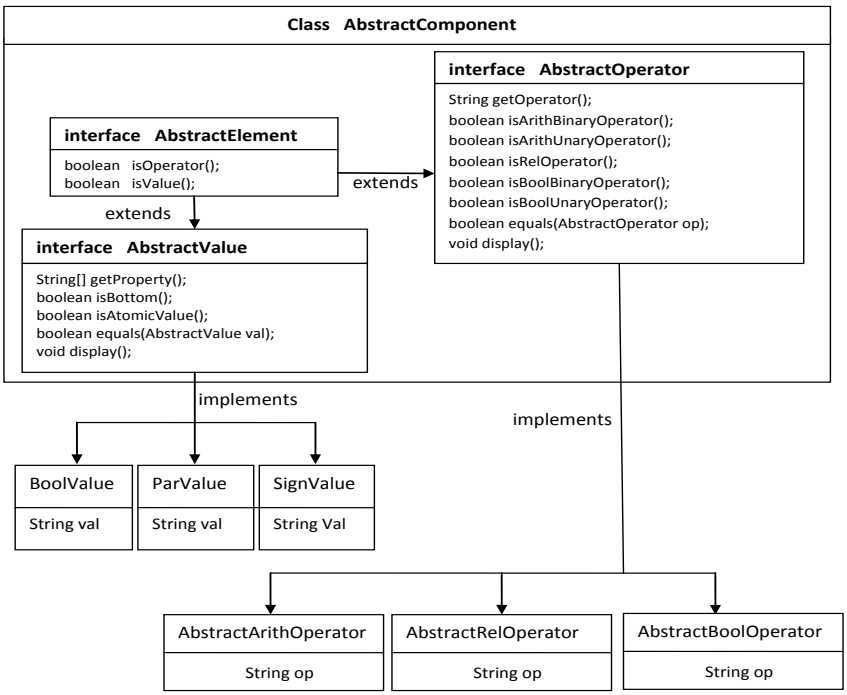


The design of the abstract domains is depicted in Figure 9.2, 9.3 and 9.4 respectively. Any abstract value such as sign value, parity value, boolean value, and any abstract operator such as arithmetic, relational, boolean operator implement the specifications represented by the interfaces "AbstractElement", "AbstractValue" and "AbstractOperator". The abstract domain such as sign domain or parity domain implements the specifications represented by the interface "AbstractDomain". The abstract states at each program point in the program is defined by the abstract environment associated with the corresponding program point. The abstract environment is defined by the class "AbstractEnvironment".

We now show how **Tukra** actually works by providing some snapshots of it.

1. **Graphical Interface 1 (Accepting inputs from users):** This is the main and starting interface window depicted in Figure 9.5. With this interface, we can browse the input program file, or we can write the source code in the text area provided on the right side of the screen, or we can open nodepad software by clicking on "Open NotePad" button to write and save the program code if does

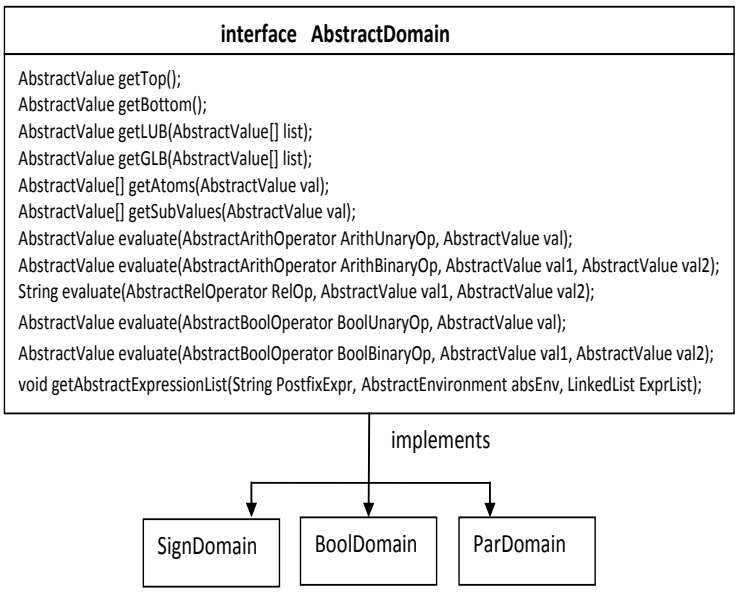
Figure 9.2: Designing Abstract Values and Abstract Operators



not exist. On clicking the "Next" button we can go to the next window.

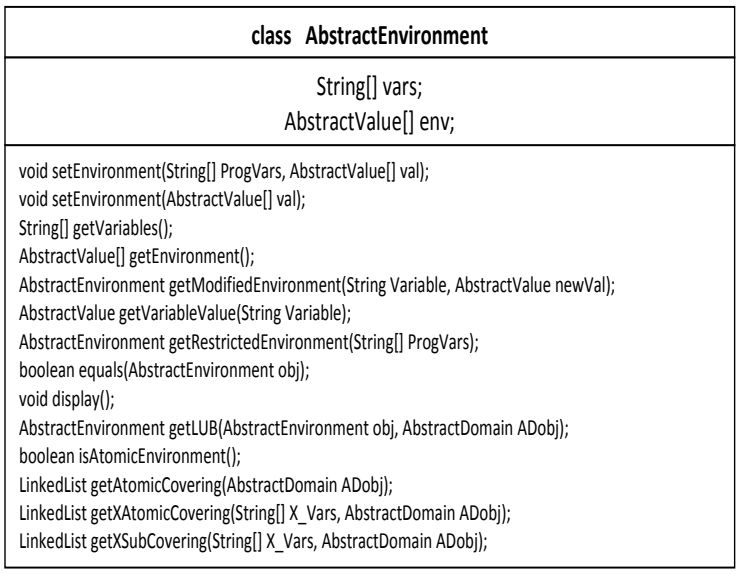
2. **Graphical Interface 2 (Syntactic slicing and semantic computations):** With this interface, depicted in Figure 9.6, users can perform the followings: (*i*) generating CDG and PDG of the input programs, (*ii*) performing syntax-based slicing, (*iii*) the options to choose the type of abstract semantic computations users want to perform on the input programs. A window showing the PDG of the program is obtained by clicking on the button "Show PDG", depicted in Figure 9.7 . Similarly, the "Show CDG" button is used to display the CDG of the input program. When users click on "Slice" button, it asks for slicing criterion: a program point and a list of variables (separated by comma) used/defined at that program point. It then computes and shows the syntactic slice of the input program *w.r.t.* the given criterion, depicted in Figure 9.8 and 9.9 respectively. On the right side of the screen, the tool displays the options for two types of abstract computation: abstract semantic relevancy of statements and abstract semantic data dependence computation, as check-boxes. Users can choose any one or both of them. However, in doing so, users must choose an abstract domain of interest shown as radio buttons (we provided here only two: SIGN and PAR domain, but we can easily add more). The "Go" button moves the tool to the next window.

Figure 9.3: Designing Abstract Domains



3. **Graphical Interface 3 (Semantics-based abstract program slicing):** This is a child window displaying over the main window depicted in Figure 9.10. Users can see in the preview area of this interface the refined program where irrelevant statements and/or irrelevant variables in the expressions are disregarded (marked in red color), depending on the options they choose on the previous interface. This interface provides two options: (*i*) generate CDG or PDG of this refined program and perform slicing on it (upper right part of the screen), and (*ii*) generate its DCG, refine it into more precise one by computing unrealizable paths based on the satisfiability of DCG annotations against their trace semantics and perform slicing on it (lower right part of the screen). The button "Show DCG" displays the SSA form of the program and the DCG annotations over all the edges of the dependence graph of the program in SSA form (depicted in Figure 9.11). The "Refinement" button shows a message with a list of refinement happens based on the DCG annotations (depicted in Figure 9.12). Observe that the checking for unrealizable paths is performed against the traces containing only atomic abstract values for the variables. Since **Tukra** is in a preliminary stage now, we are facing a memory limitation in generating the number of such traces containing atomic states only. Therefore, at this stage, we are not able to find all possible DCG-based refinement for the programs with large number of variables or in an abstract domain with large number of atomic abstract values.

Figure 9.4: Designing Abstract Environments



The slicing based on this refined semantics-based abstract DCG can be performed by supplying slicing criterion in the text areas provided on the lower right area of the screen, as depicted in Figure 9.13.

We test **Tukra** with some example programs, one of which is provided here: Figure 9.1(a) depicts a test program. Figures 9.1(b), 9.1(c) and 9.1(d) depict three types of slicing of the program *w.r.t.* ⟨L11, *y*⟩: syntax-based, semantic data dependence-based (Mastroeni and Zanadini's approach) and the proposed approach. It shows clearly that our approach generates more precise slice *w.r.t.* the existing ones.

The sound, efficient and effective theoretical support behind "Tukra" may make it more attractive to the practical field, as it is able to generate more precise slice w.r.t. the literature. Tukra is now in a preliminary stage of implementation, but there are a lot of scopes to improve it in terms of algorithmic efficiency and to generalize it in order to accept more programming language constructs on which slicing is performed.

Figure 9.5: Graphical Interface 1 (accepting inputs from users)

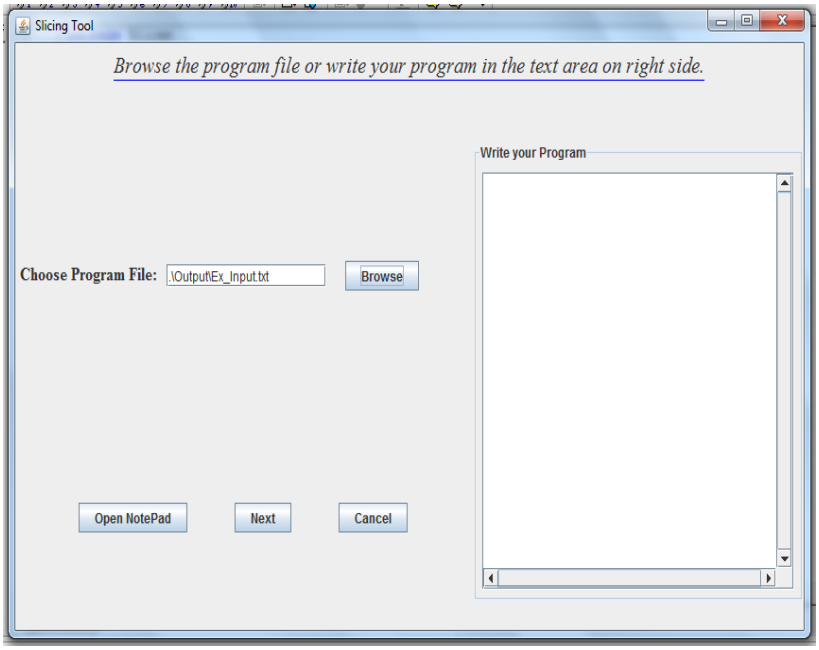


Figure 9.6: Graphical Interface 2a (syntactic slicing and semantic computation)

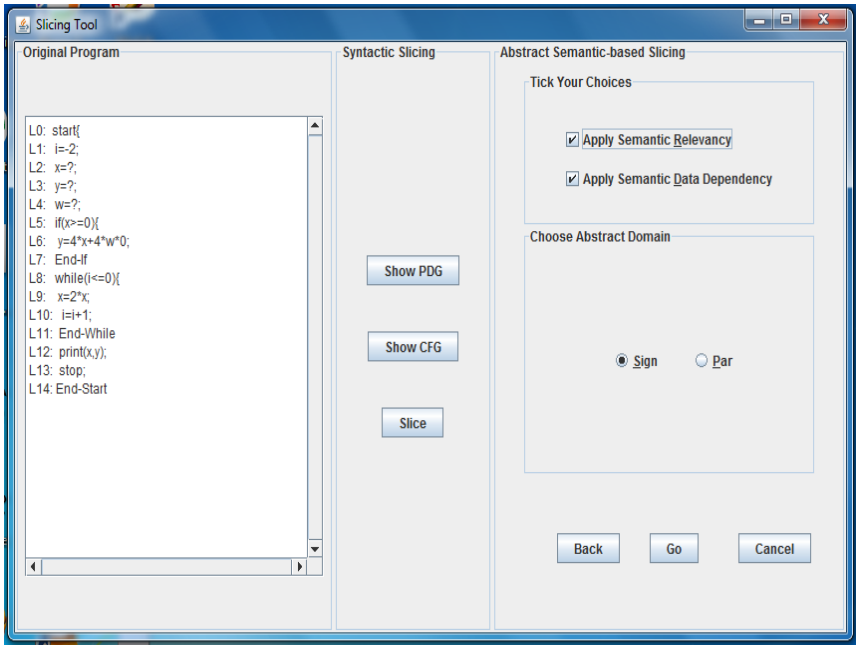


Figure 9.7: Graphical Interface 2b (showing PDG)

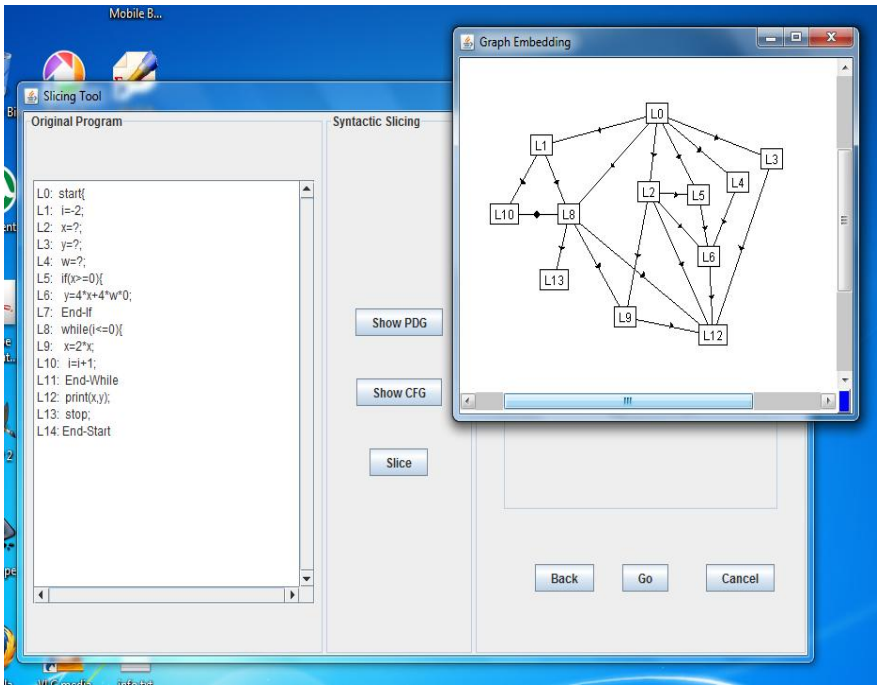


Figure 9.8: Graphical Interface 2c (accepting slicing criterion)

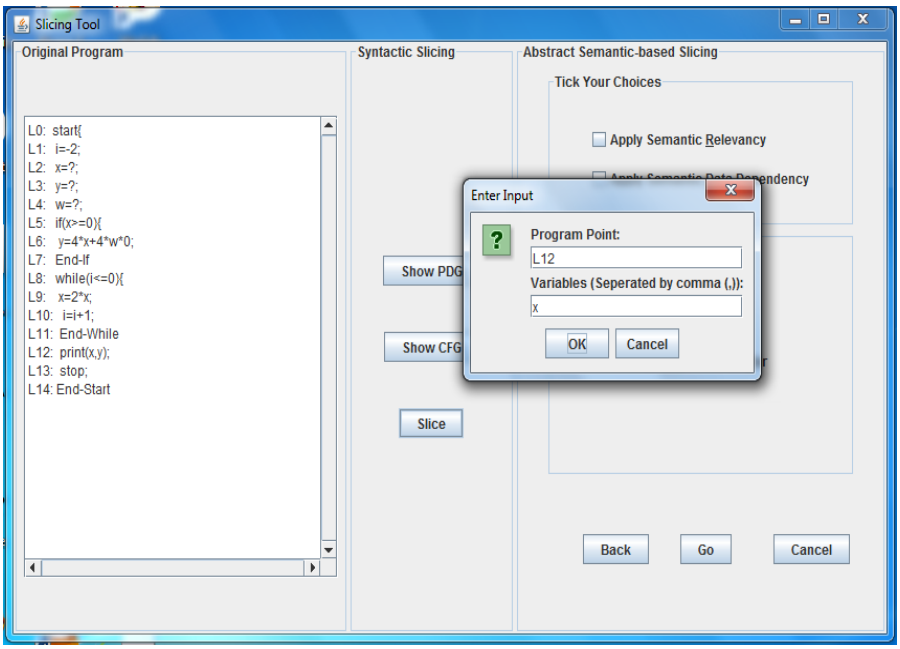


Figure 9.9: Graphical Interface 2d (showing slice *w.r.t.* criterion)

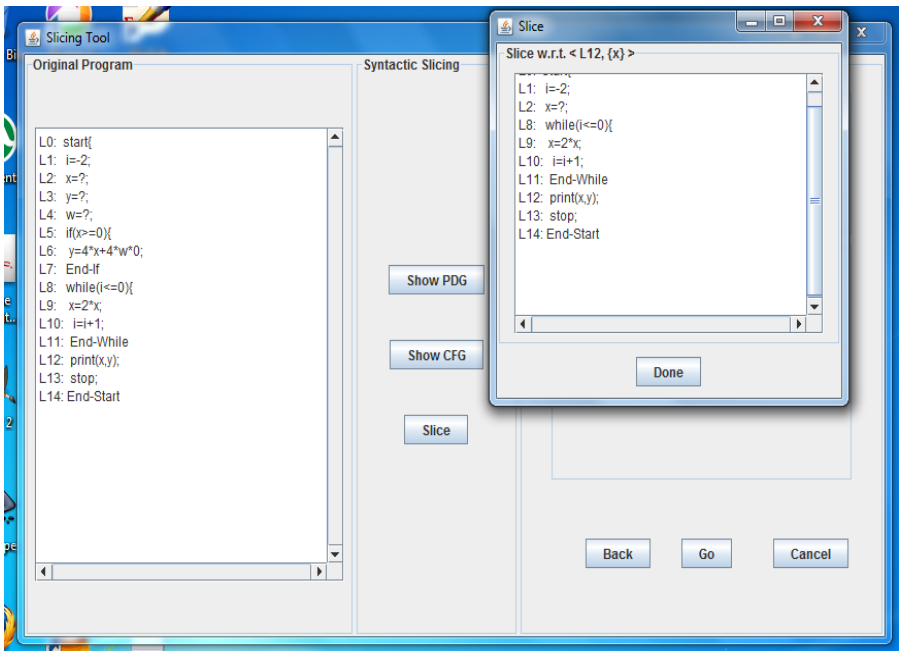


Figure 9.10: Graphical Interface 3a (semantics-based abstract program slicing)

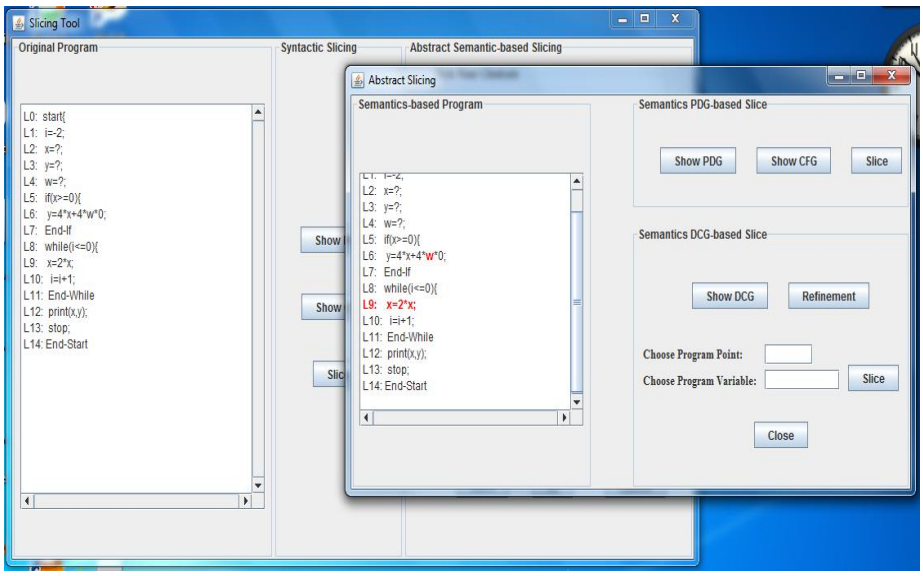


Figure 9.11: Graphical Interface 3b (showing program's SSA form and DCG annotations)

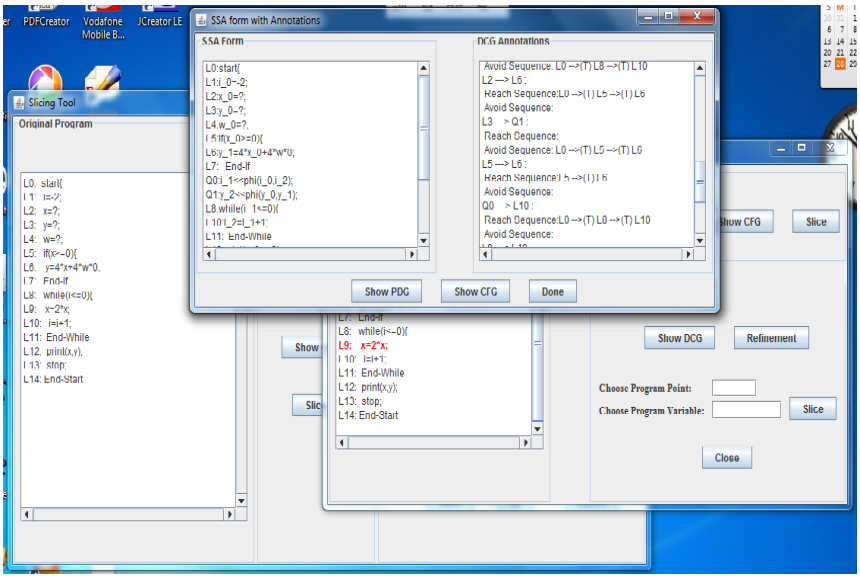


Figure 9.12: Graphical Interface 3c (showing list of DCG-based refinement)

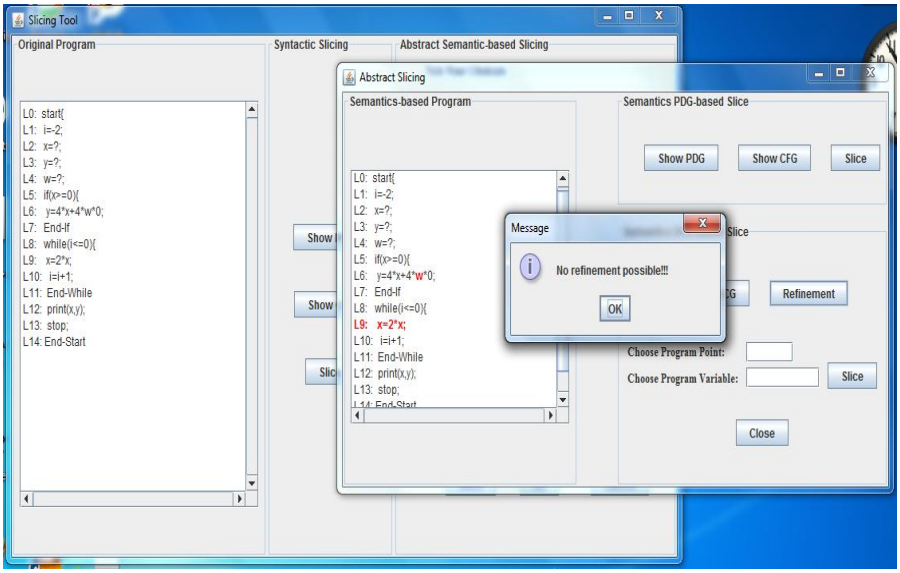


Figure 9.13: Graphical Interface 3d (DCG-based slicing)

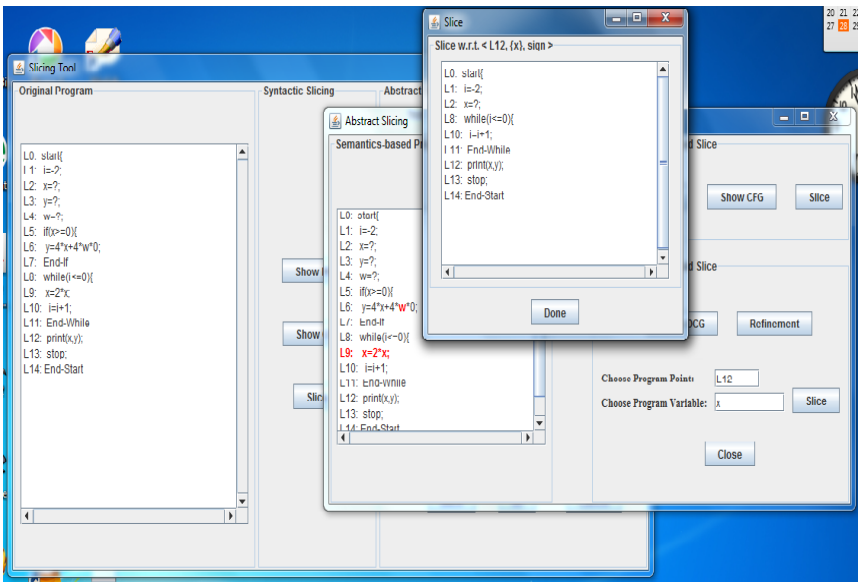


Table 9.1: A test program and its various slicing

(a) A test program

```
L1    i = −2;
L2    x =?;
L3    y =?;
L4    w =?;
L5    if(x ≥ 0){
L6          x = x + w;
L7          y = 4 ∗ w ∗ 0;}
L8    while(i ≤ 0){
L9          y = y ∗ 2;
L10         i = i + 1;}
L11   print(x, y);
```

(b) Syntax-based slicing *w.r.t.* ⟨L11, y⟩

```
L1    i = −2;
L2    x =?;
L3    y =?;
L4    w =?;
L5    if(x ≥ 0){
L7          y = 4 ∗ w ∗ 0;}
L8    while(i ≤ 0){
L9          y = y ∗ 2;
L10         i = i + 1;}
L11   print(x, y);
```

(c) Semantic data dependence-based slicing *w.r.t.* ⟨L11, y⟩

```
L1    i = −2;
L2    x =?;
L3    y =?;
L5    if(x ≥ 0){
L7          y = 4 ∗ w ∗ 0;}
L8    while(i ≤ 0){
L9          y = y ∗ 2;
L10         i = i + 1;}
L11   print(x, y);
```

(d) Proposed slicing *w.r.t.* ⟨L11, y⟩

```
L1    i = −2;
L2    x =?;
L3    y =?;
L5    if(x ≥ 0){
L7          y = 4 ∗ w ∗ 0;}
L8    while(i ≤ 0){
L10         i = i + 1;}
L11   print(x, y);
```

Chapter 10

Conclusions

In this thesis, we addressed the issue of extending the Abstract Interpretation framework to new scenarios, that may be particularly interesting from an application perspective. At the end of each chapter, we already drew some conclusive remarks on each of the subjects.

As an overall conclusions, it seems just important to notice that the field of information systems may really deserve to be further studied using semantics-based formal methods based on the data/operation abstractions, as this approach combines the soundness requirement with the efficiency and effectiveness requirements that are very demanding in any applicative scenarios, and there is still a big amount of challenging research work to be done in this area.

References

[1] Amr T. Abdel-Hamid, Sofiéne Tahar, and El Mostapha Aboulhamid. **A Survey on IP Watermarking Techniques**. Design Automation for Embedded Systems, **9**(3):211–227, 2004. 77

[2] Hiralal Agrawal and Joseph R. Horgan. **Dynamic program slicing**. In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '90), pages 246–256, White Plains, New York, June 20–22 1990. ACM Press. 6, 176, 179, 184

[3] Rakesh Agrawal, Paul Bird, Tyrone Grandison, Jerry Kiernan, Scott Logan, and Walid Rjaibi. **Extending Relational Database Systems to Automatically Enforce Privacy Policies**. In Proceedings of the 21st International Conference on Data Engineering (ICDE '05), pages 1013–1022, Tokyo, Japan, April 5–8 2005. IEEE Computer Society. 125

[4] Rakesh Agrawal, Peter J. Haas, and Jerry Kiernan. **A system for watermarking relational databases**. In Proceedings of the 2003 ACM SIGMOD international conference on Management of data (SIGMOD '03), pages 674–674, San Diego, California, June 9–12 2003. ACM Press. 86, 87, 101

[5] Rakesh Agrawal, Peter J. Haas, and Jerry Kiernan. **Watermarking relational data: framework, algorithms and analysis**. The VLDB Journal, **12**(2):157–169, 2003. 79, 86, 87, 101, 103, 109, 119

[6] Rakesh Agrawal and Jerry Kiernan. **Watermarking Relational Databases**. In Proceedings of the 28th international conference on Very Large Data Bases (VLDB '02), pages 155–166, Hong Kong, China, August 20–23 2002. VLDB Endowment. 78, 86, 87, 88, 99, 101, 103

[7] Rakesh Agrawal and Ramakrishnan Srikant. **Privacy-preserving data mining**. ACM SIGMOD Record, **29**(2):439–450, 2000. 78

[8] Ali Al-Haj and Ashraf Odeh. **Robust and Blind Watermarking of Relational Database Systems**. Journal of Computer Science, **4**(12):1024–1029, 2008. 94, 101

[9] Elena Baralis and Jennifer Widom. **An Algebraic Approach to Rule Analysis in Expert Database Systems**. In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB' 94), pages 475–486, Santiago de Chile, Chile, September 12–15 1994. Morgan Kaufmann. 180, 222, 225

[10] Véronique Benzaken and Xavier Schaefer. **Ensuring Efficiently the Integrity of Persistent Object Systems via Abstract Interpretation**. In Proceedings of the 7th Workshop on Persistent Object Systems (POS '96), pages 72–87, Cape May, New Jersey, USA, May 29–31 1996. Morgan Kaufmann Publishers Inc. 21

[11] Véronique Benzaken and Xavier Schaefer. **Static Integrity Constraint Management in Object-Oriented Database Programming Languages via Predicate Transformers**. In Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97), pages 60–84, Finland, June 9–13 1997. Springer-Verlag LNCS, Volume 1241. 21

[12] Véronique Benzaken and Xavier Schaefer. **Static Management of Integrity in Object-Oriented Databases: Design and Implementation**. In Proceedings of the 6th International Conference on Extending Database Technology (EDBT '98), pages 311–325, Valencia, Spain, March 23–27 1998. Springer LNCS, Volume 1377. 21

[13] Jean-Francois Bergeretti and Bernard A. Carré. **Information-flow and data-flow analysis of while-programs**. ACM Transactions on Programming Languages and Systems, **7**(1):37–61, 1985. 183

[14] Elisa Bertino and Elena Ferrari. **Secure and selective dissemination of XML documents**. ACM Transactions on Information and System Security, **5**(3):290–331, 2002. 126

[15] Elisa Bertino, Sushil Jajodia, and Pierangela Samarati. **A flexible authorization mechanism for relational data management systems**. ACM Transactions on Information Systems, **17**(2):101–140, 1999. 121

[16] Elisa Bertino, Ashish Kamra, and James P. Early. **Profiling Database Applications to Detect SQL Injection Attacks**. In Proceedings of the 26th IEEE International Performance Computing and Communications Conference (IPCCC '07), pages 449 – 458, New Orleans, Louisiana, USA, April 11–13 2007. IEEE Computer Society. 150

[17] Elisa Bertino, Beng Chin Ooi, Yanjiang Yang, and Robert H. Deng. **Privacy and Ownership Preserving of Outsourced Medical Data**. In Proceedings of the 21st International Conference on Data Engineering (ICDE '05), pages 521–532, Tokyo, Japan, April 05–08 2005. IEEE Computer Society. 94

[18] Sukriti Bhattacharya. Property Driven Program Slicing and Watermarking in the Abstract Interpretation Framework. PhD thesis, Università Ca' Foscari Venezia, 2011. 177, 179

[19] Sukriti Bhattacharya and Agostino Cortesi. **A Distortion Free Watermark Framework For Relational Databases**. In Proceedings of the 4th International Conference on Software and Data Technologies (ICSOFT '09), pages 229–234, Sofia, Bulgaria, July 26–29 2009. INSTICC Press. 96, 102

[20] Sukriti Bhattacharya and Agostino Cortesi. **A Generic Distortion Free Watermarking Technique for Relational Databases**. In Proceedings of the 5th International Conference on Information Systems Security (ICISS '09), pages 252–264, Kolkata, India, December 14–18 2009. Springer LNCS, Volume 5905. 79, 98, 102, 109, 119

[21] Sukriti Bhattacharya and Agostino Cortesi. **Database Authentication by Distortion-Free Watermarking**. In Proceedings of the 5th International Conference on Software and Data Technologies (ICSOFT '10), pages 219–226, Athens, Greece, July 22–24 2010. INSTICC Press. 98, 102

[22] D. Binkley, S. Danicic, T. Gyimóthy, M. Harman, Á. Kiss, and B. Korel. **A formalisation of the relationship between forms of program slicing**. Science of Computer Programming, **62**(3):228–252, 2006. 177, 179

[23] Stefan Bottcher, Rita Hartel, and Matthias Kirschner. **Detecting Suspicious Relational Database Queries**. In Proceedings of the 3rd International Conference on Availability, Reliability and Security (ARES '08), pages 771–778, Barcelona, Spain, March 4–7 2008. IEEE Computer Society. 125

REFERENCES

[24] Luc Bouganim, François Dang Ngoc, and Philippe Pucheral. **Client-based access control management for XML documents**. In Proceedings of the 13th International Conference on Very Large Data Bases (VLDB '04), pages 84–95, Toronto, Canada, August 31–September 3 2004. Morgan Kaufmann Publishers Inc. 126, 127

[25] Stephen W. Boyd and Angelos D. Keromytis. **SQLrand: Preventing SQL Injection Attacks**. In Proceedings of the 2nd International Conference on Applied Cryptography and Network Security Conference (ACNS '04), pages 292–302, Yellow Mountain, China, June 8–11 2004. Springer LNCS, Volume 3089. 149, 162

[26] José Luís Braga, Alberto H. F. Laender, and Claudiney Vander Ramos. **Cooperative Relational Database Querying Using Multiple Knowledge Bases**. In Proceedings of the 12th International Florida Artificial Intelligence Research Society Conference (FLAIRS '99), pages 95–99, Orlando, Florida, USA, May 1–5 1999. AAAI Press. 166

[27] Gregory Buehrer, Bruce W. Weide, and Paolo A. G. Sivilotti. **Using Parse Tree Validation to Prevent SQL Injection Attacks**. In International Workshop on Software Engineering and Middleware (SEM '05), pages 106–113, Lisbon, Portugal, September 5–6 2005. ACM Press. 148, 161, 162

[28] Gunter V. Bultzingwloewen. **Translating and optimizing SQL queries having aggregates**. In Proceedings of the 13th International Conference on Very Large Data Bases (VLDB '87), pages 235–243, Brighton, England, September 1–4 1987. Morgan Kaufmann Publishers Inc. 2, 18, 20

[29] Robert Cartwright and Mattias Felleisen. **The semantics of program dependence**. ACM SIGPLAN Notices, **24**(7):13–27, 1989. 176, 179

[30] Stefano Ceri and Georg Gottlob. **Translating SQL into relational algebra: Optimization, Semantics, and Equivalence of SQL Queries**. IEEE Transactions on Software Engineering, **11**(4):324–345, 1985. 2, 18, 20

[31] Liqian Chen, Antoine Miné, and Patrick Cousot. **A Sound Floating-Point Polyhedra Abstract Domain**. In Proceedings of the 6th Asian Symposium on Programming Languages and Systems (APLAS '08), pages 3–18, Bangalore, India, December 9–11 2008. Springer LNCS, Volume 5356. 106, 117, 173

[32] Liqian Chen, Antoine Miné, Ji Wang, and Patrick Cousot. **Interval Polyhedra: An Abstract Domain to Infer Interval Linear Relationships**. In Proceedings of the 16th International Static Analysis Symposium (SAS '09), pages 309–325, Los Angeles, CA, USA, August 9–11 2009. Springer LNCS, Volume 5673. 106, 173

[33] James Cheney. **Program Slicing and Data Provenance**. IEEE Data Engineering Bulletin, **30**:22–28, 2007. 222

[34] Wesley W. Chu and Qiming Chen. **Neighborhood and associative query answering**. Journal of Intelligent Information Systems, **1**(3–4):355–382, 1992. 165, 167

[35] Wesley W. Chu and Qiming Chen. **A Structured Approach for Cooperative Query Answering**. IEEE Transactions on Knowledge and Data Engineering, **6**(5):738–749, 1994. 167

[36] Wesley W. Chu, H. Yang, K. Chiang, M. Minock, G. Chow, and C. Larson. **CoBase: a scalable and extensible cooperative information system**. Journal of Intelligent Information Systems, **6**(2–3):223–259, 1996. 165, 167

[37] Aniello Cimitile, Andrea De Lucia, and Malcolm Munro. **Identifying reusable functions using specification driven program-slicing: a case study**. In Proceedings of the 11th International Conference on Software Maintenance (ICSM '95), pages 124–133, Opio (Nice), France, October 17–20 1995. IEEE Computer Society. 175

[38] Anthony Cleve. **Program analysis and transformation for data-intensive system evolution**. In Proceedings of the 26th IEEE International Conference on Software Maintenance (ICSM '10), pages 1–6, Timisoara, Romania, September 12–18 2010. IEEE Computer Society. 181

[39] E. F. Codd. **A DataBase Sublanguage Founded on the Relational Calculus**. In Proceedings of 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, pages 35–68, San Diego Caifornia, November 11–12 1971. ACM Press. 20

[40] E. F. Codd. **Relational Completeness of DataBase Subanguages**. In Database Systems, pages 65–98, San Jose, California, March 6 1972. Prentice Hall and IBM Research Report RJ 987. 19, 20

[41] E. F. Codd. **A Relational Model of Data for Large Shared Data Banks**. Communications of the ACM 25th Anniversary Issue, **26**(1):64–69, 1983. 19

[42] Agostino Cortesi, Agostino Dovier, Elisa Quintarelli, and Letizia Tanca. **Operational and abstract semantics of the query language G-Log**. Theoretical Computer Science, **275**(1–2):521–560, 2002. 18

[43] Agostino Cortesi and Raju Halder. **An Abstract Interpretation Framework for Structured Query Languages**. In Proceedings of the 21st Nordic Workshop on Programming Theory (NWPT '09), pages 41–43, Lyngby, Denmark, 14–16 October 2009. DTU Informatics. 3, 17

[44] Agostino Cortesi and Raju Halder. **Dependence Condition Graph for Semantics-based Abstract Program Slicing**. In Proceedings of the 10th International Workshop on Language Descriptions Tools and Applications (LDTA '10), pages 4:1–4:6, Paphos, Cyprus, March 27–28 2010. ACM Press. 6, 175

[45] Agostino Cortesi and Matteo Zanioli. **Widening and narrowing operators for abstract interpretation**. Computer Languages, Systems & Structures, **37**(1):24–42, 2011. 218

[46] Patrik Cousot. **Abstract Interpretation Based Formal Methods and Future Challenges**. In Informatics - 10 Years Back. 10 Years Ahead., pages 138–156. Springer LNCS, Volume 2000, 2001. 1

[47] Patrik Cousot and Radhia Cousot. **Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints**. In Proceedings of the 4th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '77), pages 238–252, Los Angeles, CA, USA, January 17–19 1977. ACM Press. 1, 9, 14, 18, 137, 138, 187

[48] Patrik Cousot and Radhia Cousot. **Systematic Design of Program analysis Frameworks**. In Proceedings of the 6th Annual ACM SIGACT-SIGPLAN symposium on Principles of programming languages (POPL '79), pages 269–282, San Antonio, Texas, January 29–31 1979. ACM Press. 9, 14, 18

[49] Patrik Cousot and Radhia Cousot. **Systematic Design of Program Transformation Frameworks by Abstract Interpretation**. In Proceedings of the 29th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '02), pages 178–190, Portland, OR USA, January 16–18 2002. ACM Press. 9, 14, 18

[50] Xinchun CUI, Xiaolin QIN, and GANG Sheng. **A Weighted Algorithm for Watermarking Relational Databases**. Wuhan University Journal of Natural Science, **12**(1):79–82, 2007. 93

[51] Xinchun Cui, Xiaolin Qin, Gang Sheng, and Jiping Zheng. **A Robust Algorithm for Watermark Numeric Relational Databases**. In Proceedings of the 2010 International conference on Intelligent computing (ICIC '06), pages 810–815, Kunming, China, August 16–19 2006. Springer Lecture Notes in Control and Information Sciences. 93

[52] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. **Efficiently computing static single assignment form and the control dependence graph**. ACM Transactions on Programming Languages and Systems, **13**(4):451–490, 1991. 181

[53] Ernesto Damiani, Sabrina de Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. **Design and implementation of an access control processor for XML documents**. Journal of computer and telecommunications netowrking, **33**(1–6):59–75, 2000. 126

[54] Ernesto Damiani, Sabrina de Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. **A fine-grained access control system for XML documents**. ACM Transactions on Information and System Security, **5**(2):169–202, 2002. 4, 121, 126, 139, 143

[55] C. J. Date. An introduction to DataBase Systems. Addison-Wesley Longman Publishing Co., Inc., 8th edition, 2003. 9, 11

[56] Ramez Elmasri and Shamkant B. Navathe. Fundamentals of DATABASE SYSTEMS. Addison-Wesley Longman Publishing Co., Inc., 4th edition, 2003. ix, 9, 11, 12

[57] Chuhong Fei, Deepa Kundurb, and Raymond Kwonga. **Analysis and Design of Authentication Watermarking**. In Proceedings of the Security, Steganography, and Watermarking of Multimedia Contents IV (SSWMC '04), pages 760–771, San Jose, California, USA, January 18–22 2004. SPIE, Volume 5306. 79

[58] Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. **The program dependence graph and its use in optimization**. ACM Transactions on Programming Languages and Systems, **9**(3):319–349, 1987. 175, 182

[59] Logozzo Francesco. **Practical Verification for the Working Programmer with CodeContracts and Abstract Interpretation (Invited Talk)**. In Proceedings of the 12th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI '11), pages 19–22, Austin, TX, USA, January 23–25 2011. Springer LNCS, Volume 6538. 1

[60] Piero Fraternali and Letizia Tanca. **A structured approach for the definition of the semantics of active databases**. ACM Transactions on Database Systems, **20**(4):414–471, 1995. 21

[61] Terry Gaasterland. **Cooperative Answering through Controlled Query Relaxation**. IEEE Expert: Intelligent Systems and Their Applications, **12**(5):48–59, 1997. 165, 166

[62] Keith Brian Gallagher and James R. Lyle. **Using Program Slicing in Software Maintenance**. IEEE Transactions on Software Engineering, **17**(8):751–761, 1991. 175

[63] Richard Gerber and Seongsoo Hong. **Slicing real-time programs for enhanced schedulability**. ACM Transactions on Programming Languages and Systems, **19**(3):525–555, 1997. 175

[64] Roberto Giacobazzi, Francesco Ranzato, and Francesca Scozzari. **Making abstract interpretations complete**. Journal of the ACM, **47**(2):361–416, 2000. 9, 14, 15, 18, 169

[65] Dina Goldin, Srinath Srinivasa, and Vijaya Srikanti. **Active Databases as Information Systems**. In Proceedings of the 8th International Database Engineering and Applications Symposium (IDEAS '04), pages 123–130, Coimbra, Portugal, July 7–9 2004. IEEE Computer Society. 9, 10

[66] Dina Goldin, Srinath Srinivasa, and Bernhard Thalheim. **IS=DBS+Interaction: Towards Principles of Information System Design**. In Proceedings of the 19th International Conference on Conceptual Modeling (ER '00), pages 140–153, Salt Lake City, Utah, USA, October 9–12 2000. Springer LNCS, volume 1920. ix, 9, 10

[67] Derek Goldrei. Propositional and Predicate Calculus: A Model of Argument. Springer, 1st edition, 2005. 21

[68] Diganta Goswami and Rajib Mall. **An efficient method for computing dynamic program slices**. Information Processing Letters, **81**(2):111–117, 2002. 6, 176, 179, 184

[69] Carl Gould, Zhendong Su, and Premkumar T. Devanbu. **Static Checking of Dynamically Generated Queries in Database Applications**. In Proceedings of the 26th International Conference on Software Engineering (ICSE '04), pages 645–654, Edinburgh, United Kingdom, May 23–28 2004. IEEE Computer Society. 150

[70] Patricia P. Griffiths and Bradford W. Wade. **An authorization mechanism for a relational database system**. ACM Transactions on Database Systems, **1**(3):242–255, 1976. 121

[71] Fei Guo, Jianmin Wang, and Deyi Li. **Fingerprinting relational databases**. In Proceedings of the 2006 ACM symposium on Applied computing (SAC '06), pages 487–492, Dijon, France, April 23–27 2006. ACM Press. 99, 101

[72] Fei Guo, Jianmin Wang, Zhihao Zhang, Xiaojun Ye, and Deyi Li. **An Improved Algorithm to Watermark Numeric Relational Data**. In Proceedings of the 6th International Workshop on Information Security applications (WISA '05), pages 138–149, Jeju Island, Korea, August 22–24 2005. Springer LNCS, Volume 3786. 93, 101

[73] Huiping Guo, Yingjiu Li, Anyi Liu, and Sushil Jajodia. **A fragile watermarking scheme for detecting malicious modifications of database relations**. Information Sciences, **176**(10):1350–1378, 2006. 79, 91, 101, 119

[74] Gupta Gupta and Josef Pieprzyk. **Database Relation Watermarking Resilient against Secondary Watermarking Attacks**. In Proceedings of the 5th International Conference on Information Systems Security (ICISS '09), pages 222–236, Kolkata, India, December 14–18 2009. Springer LNCS, Volume 5905. 88, 101

[75] Rajiv Gupta, Mary Jean Harrold, and Mary Lou Soffa. **An approach to regression testing using slicing**. In Proceedings of International Conference on Software Maintenance (ICSM '92), pages 299–308, Orlando, FL, USA, November 9–12 1992. IEEE Computer Society. 175

[76] Narjes Hachani and Habib Ounelli. **A Knowledge-Based Approach For Database Flexible Querying**. In Proceedings of the 17th International Workshop on Database and Expert Systems Applications (DEXA '06), pages 420–424, Krakow, Poland, September 4–8 2006. IEEE Computer Society. 167

[77] Hakan Hacigumus, Bala Iyer, and Sharad Mehrotra. **Providing Database as a Service**. In Proceedings of the 18th International Conference on Data Engineering (ICDE'02), pages 29–38, San Jose, CA, USA, 26 February–1 March 2002. IEEE Computer Society. 78

[78] Vivek Haldar, Deepak Chandra, and Michael Franz. **Dynamic Taint Propagation for Java**. In Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC '05), pages 303–311, Tucson, AZ, USA, December 5–9 2005. IEEE Computer Society. 150

[79] Raju Halder and Agostino Cortesi. **Abstract Interpretation for Sound Approximation of Database Query Languages**. In Proceedings of the IEEE 7th International Conference on Informatics and Systems (INFOS '10), Advances in Data Engineering and Management Track, pages 53–59, Cairo, Egypt, 28–30 March 2010. IEEE Press. IEEE Catalog Number: IEEE CFP1006J-CDR. 3, 17

[80] Raju Halder and Agostino Cortesi. **Obfuscation-based Analysis of SQL Injection Attacks**. In Proceedings of the 15th IEEE Symposium on Computers and Communications (ISCC '10), pages 931–938, Riccione, Italy, 22–25 June 2010. IEEE Press. 5, 147

[81] Raju Halder and Agostino Cortesi. **Observation-based Fine Grained Access Control for Relational Databases**. In Proceedings of the 5th International Conference on Software and Data Technologies (ICSOFT '10), pages 254–265, Athens, Greece, July 22–24 2010. INSTICC Press. 4, 121

[82] Raju Halder and Agostino Cortesi. **A Persistent Public Watermarking of Relational Databases**. In Somesh Jha and Anish Mathuria, editors, Proceedings of the 6th International Conference on Information Systems Security (ICISS '10), pages 216–230, Gujrat, India, December 15–19 2010. Springer LNCS, Volume 6503. 3, 77

[83] Raju Halder and Agostino Cortesi. **Persistent Watermarking of Relational Databases**. In Proceedings of the 1st IEEE International Conference on Advances in Communication, Network, and Computing (CNC '10), pages 46–52, Kerala, India, October 4–5 2010. IEEE Computer Society. 3, 77

[84] Raju Halder and Agostino Cortesi. **Cooperative Query Answering by Abstract Interpretation**. In Proceedings of the 37th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '11), pages 284–296,, Novỳ Smokovec, Slovakia, January 22–28 2011. Springer LNCS, Volume 6543. 5, 165

[85] Raju Halder and Agostino Cortesi. **Observation-based Fine Grained Access Control for XML Documents**. In Proceedings of the 10th International Conference on Computer Information Systems and Industrial Management Applications (CISIM '11), pages 267–276, Kolkata, India, December 14–16 2011. Springer CCIS, Volume 245. 4, 121

[86] Raju Halder and Agostino Cortesi. **Fine Grained Access Control for Relational Databases by Abstract Interpretation**. In Josè Cordeiro, Maria Virvou, and Boris Shishkov, editors, Software and Data Technologies, pages 235–249. Springer CCIS, Volume 170, 2012. (Selected Revised Papers). 4, 121

[87] Raju Halder and Agostino Cortesi. **Abstract Interpretation of Database Query Languages**. Computer Languages, Systems and Structures, **38**(2), 2012. (To apear). 3, 17

[88] Raju Halder and Agostino Cortesi. **Abstract Program Slicing on Dependence Condition Graph**. Science of Computer Programming, (Accepted, under final revision). 6, 175

[89] Raju Halder, Partha Sarathi Dasgupta, Saptarshi Naskar, and Samar Sen Sarma. **An Internet-based IP Protection Scheme for Circuit Designs using Linear Feedback Shift Register (LFSR)-based Locking**. In Proceedings of the 22nd Annual Symposium on Integrated Circuits and System Design(SBCCI '09), pages 15:1–15:6, Natal, Brazil, August 31st–September 3rd 2009. ACM Press. 87, 97

[90] Raju Halder, Partha Sarathi Dasgupta, Saptarshi Naskar, and Samar Sen Sarma. **An Internet-based IP Protection Scheme for Circuit Designs using Linear Feedback Shift Register-based Locking**. Engineering Letters, **19**(2):84–94, 2011. 87, 97

[91] Raju Halder, Shantanu Pal, and Agostino Cortesi. **Watermarking Techniques for Relational Databases: Survey, Classification and Comparison**. Journal of Universal Computer Science, **16**(21):3164–3190, 2010. 3, 77

[92] William G. J. Halfond and Alessandro Orso. **AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks**. In Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE '05), pages 174–183, Long Beach, California, USA, November 7–11 2005. ACM Press. 148, 161, 162

[93] William G.J. Halfond, Jeremy Viegas, and Alessandro Orso. **A Classication of SQL-Injection Attacks and Countermeasures**. In Proceedings of the IEEE International Symposium on Secure Software Engineering (ISSSE '06), Arlington, VA, USA, March 13–15 2006. IEEE. 148, 162

[94] Rebecca Hasti and Susan Horwitz. **Using static single assignment form to improve flow-insensitive pointer analysis**. ACM SIGPLAN Notices, **33**(5):97–105, 1998. 182

[95] Susan Horwitz, Jan Prins, and Thomas Reps. **On the adequacy of program dependence graphs for representing programs**. In Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '88), pages 146–157, San Diego, California, United States, January 1988. ACM Press. 176, 179

[96] Susan Horwitz, Thomas Reps, and David Binkley. **Interprocedural slicing using dependence graphs**. ACM Transactions on Programming Languages and Systems, **12**(1):26–60, 1990. 6, 176, 179

[97] Tsan-sheng Hsu, Churn-Jung Liau, Da-Wei Wang, and Jeremy K.-P. Chen. **Quantifying Privacy Leakage through Answering Database Queries**. In Proceedings of the 5th International Conference on Information Security (ISC '02), pages 162–176, London, UK, September 30–October 2 2002. Springer LNCS, Volume 2433. 126

[98] Jinmin Hu and Paul Grefen. **Component Based System Framework for Dynamic B2B Interaction**. In Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment (COMPSAC '02), pages 557–562, Oxford, England, August 26–29 2002. IEEE Computer Society. 78

[99] Tianlei Hu, Gang Chen, Ke Chen, and Jinxiang Dong. **GARWM: Towards a Generalized and Adaptive Watermark Scheme for Relational Data**. In Proceedings of the 6th International Conference in Advances in Web-Age Information Management (WAIM '05), pages 380–391, Hangzhou, China, October 11–13 2005. Springer LNCS, Volume 3739. 92, 101

[100] Zhongyan Hu, Zaihui Cao, and Jianhua Sun. **An Image Based Algorithm for Watermarking Relational Databases**. In Proceedings of the 2009 International Conference on Measuring Technology and Mechatronics Automation (ICMTMA '09), pages 425–428, Zhangjiajie, Hunan, China, April 11–12 2009. IEEE Computer Society. 89, 90, 91, 93

[101] Kaiyin Huang, Min Yue, Pengfei Chen, Yanshan He, and Xiaoyun Chen. **A Cluster-Based Watermarking Technique for Relational Database**. In Proceedings of the 1st International Workshop on Database Technology and Applications (DBTA '09), pages 107–110, Wuhan, China, April 25–26 2009. IEEE Press. 92

[102] Min Huang, Jiaheng Cao, Zhiyong Peng, and Ying Fang. **A New Watermark Mechanism for Relational Data**. In Proceedings of the 4th International Conference on Computer and Information Technology (CIT '04), pages 946–950, Wuhan, China, September 14–16 2004. IEEE Computer Society. 92, 101

[103] Yao-Wen Huang, Shih-Kun Huang, Tsung-Po Lin, and Chung-Hung Tsai. **Web Application Security Assessment by Fault Injection and Behavior Monitoring**. In Proceedings of the 11th International World Wide Web Conference (WWW '03), pages 148–159, Budapest, Hungary, May 20–24 2003. ACM Press. 150

[104] Yao-Wen Huang, Fang Yu, Christian Hang, Chung-Hung Tsai, Der-Tsai Lee, and Sy-Yen Kuo. **Securing Web Application Code by Static Analysis and Runtime Protection**. In Proceedings of the 13th International World Wide Web Conference (WWW '04), pages 40–52, New York, USA, May 17–20 2004. ACM Press. 150

[105] Soon-Young Huh and Kae-Hyun Moon. **Approximate query answering approach based on data abstraction and fuzzy relation**. In Proceedings of INFORMS-KORMS, pages 2057–2065, Seoul, Korea, June 18–21 2000. The Korean Operations Research and Management Science Society. 167

[106] Tadao Ichikawa and Masahito Hirakawa. **ARES: a relational database with the capability of performing flexible interpretation of queries**. IEEE Transactions on Software Engineering, **12**(5):624–634, 1986. 167

[107] AMERICAN NATIONAL STANDARD INSTITUTE. **Information technology-Database languages-SQL-Part 2: Foundation (SQL/Foundation)**. In ISO/IEC 9075-2:2008. http://www.iso.org/iso/iso_catalogue/catalogue_tc/ catalogue_detail.htm? csnumber = 38640. 41

[108] Yannis E. Ioannidis and Viswanath Poosala. **Histogram-Based Approximation of Set-Valued Query Answers**. In Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99), pages 174–185, Edinburgh, Scotland, UK, September 7–10 1999. Morgan Kaufmann Publishers Inc. 17

[109] Daniel Jackson and Eugene J. Rollins. **A new model of program dependences for reverse engineering**. ACM SIGSOFT Software Engineering Notes, **19**(5):2–10, 1994. 175, 183, 184

[110] Sushil Jajodia, Pierangela Samarati, V. S. Subrahmanian, and Eliza Bertino. **A unified framework for enforcing multiple access control policies**. SIGMOD Record, **26**(2):474–485, 1997. 121

[111] Matthias Jarke and Jurgen Koch. **Query Optimization in Database Systems**. ACM Computing Surveys, **16**(2):111–152, 1984. 20, 21

[112] Govind Kabra, Ravishankar Ramamurthy, and S. Sudarshan. **Redundancy and information leakage in fine-grained access control**. In Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '06), pages 133–144, Chicago, IL, USA, June 27–29 2006. ACM Press. 121, 125, 126

[113] Ibrahim Kamel. **A schema for protecting the integrity of databases**. Computers & Security, **28**(7):698–709, 2009. 98, 102

[114] Auguste Kerckhoffs. **La cryptographie militaire**. Journal des Sciences Militaires, **9**:5–38, 1983. 85

[115] Myung Keun Shin, Soon-Young Huh, and Wookey Lee. **Providing ranked cooperative query answers using the metricized knowledge abstraction hierarchy**. Expert Systems with Applications, **32**(2):469–484, 2007. 165, 167

[116] Sanjeev Khanna and Francis Zane. **Watermarking maps: hiding information in structured data**. In Proceedings of the 11th annual ACM-SIAM symposium on Discrete algorithms (SODA '00), pages 596–605, San Francisco, California, United States, January 9–11 2000. Society for Industrial and Applied Mathematics. 78

[117] Anthony Klug. **Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions**. Journal of the ACM, **29**(3):699–717, 1982. 18, 20

[118] Sastry Konduri, Brajendra Panda, and Wing-Ning Li. **Monitoring Information Leakage During Query Aggregation**. In Proceedings of the 4th International Conference in Distributed Computing and Internet Technology (ICDCIT'07), pages 89–96, Bangalore, India, December 17–20 2007. Springer LNCS, Volume 4882. 126

[119] Bogdan Korel and Janusz Laski. **Dynamic program slicing**. Information Processing Letters, **29**(3):155–163, 1988. 175, 179

[120] Bogdan Korel and Jurgen Rilling. **Dynamic Program Slicing in Understanding of Program Execution**. In Proceedings of the 5th International Workshop on Program Comprehension (WPC '97), pages 80–89, Dearborn, MI, USA, May 28–30 1997. IEEE Computer Society. 175

[121] Lazaros Koromilas, George Chinis, Irini Fundulaki, and Sotiris Ioannidis. **Controlling Access to XML Documents over XML Native and Relational Databases**. In Proceedings of the 6th VLDB Workshop on Secure Data Management (SDM '09), pages 122–141, Lyon, France, August 28 2009. Springer LNCS, Volume 5776. 121, 126, 127, 143

[122] Yuji Kosuga, Kono Kenji, Miyuki Hanaoka, Miho Hishiyama, and Yu Takahama. **Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection**. In Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC '07), pages 107–117, Miami Beach, Florida, USA, December 10–14 2007. IEEE Computer Society. 150

[123] Jens Krinke. **Static slicing of threaded programs**. ACM SIGPLAN Notices, **33**(7):35–42, 1998. 175

[124] D. J. Kuck, R. H. Kuhn, D. A. Padua, B. Leasure, and M. Wolfe. **Dependence graphs and compiler optimizations**. In Proceedings of the 8th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '81), pages 207–218, Williamsburg, Virginia, January 1981. ACM Press. 175, 182

[125] J. L. Kuhns. **Answering Questions by Computer: A Logical Study**. In Report RM-5428-PR, page 137, Santa Monica, California, December 1967. The Rand Corporation. 19

[126] Julien Lafaye. **An Analysis of Database Watermarking Security**. In Proceedings of the 3rd International Symposium on Information Assurance and Security (IAS '07), pages 462–467, Manchester, United Kingdom, August 29–31 2007. IEEE Computer Society. 87

[127] Filippo Lanubile and Giuseppe Visaggio. **Extracting Reusable Functions by Flow Graph-Based Program Slicing**. IEEE Transactions on Software Engineering, **23**(4):246–259, 1997. 175

[128] Dongwon Lee, Wang-Chien Lee, and Peng Liu. **Supporting XML Security Models Using Relational Databases: A Vision**. In Proceedings of the 1st International XML Database Symposium (Xsym '03), pages 267–281, Berlin, Germany, September 8 2003. Springer LNCS, Volume 2824. 126, 127

[129] Sin-Joo Lee and Sung-Hwan Jung. **A survey of watermarking techniques applied to multimedia**. In Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE '01), pages 272–277, Pusan, South Korea, June 12–18 2001. IEEE Press. 77

REFERENCES

[130] Kristen LeFevre, Rakesh Agrawal, Vuk Ercegovac, Raghu Ramakrishnan, Yirong Xu, and David DeWitt. **Limiting disclosure in hippocratic databases**. In Proceedings of the 30th international conference on Very Large Data Bases (VLDB '04), pages 108–119, Toronto, Canada, August 31–September 3 2004. Morgan Kaufmann Publishers Inc. 125, 128, 131

[131] Yingjiu Li. Database Watermarking: A Systematic View. Springer Verlag, 2007. 96

[132] Yingjiu Li and Robert Huijie Deng. **Publicly verifiable ownership protection for relational databases**. In Proceedings of the 2006 ACM Symposium on Information, computer and communications security (ASIACCS '06), pages 78–89, Taipei, Taiwan, March 21–24 2006. ACM Press. 79, 97, 98, 102, 109, 117, 119

[133] Yingjiu Li, Huiping Guo, and Sushil Jajodia. **Tamper detection and localization for categorical data using fragile watermarks**. In Proceedings of the 4th ACM workshop on Digital rights management (DRM '04), pages 73–82, Washington, DC, USA, Oct 25 2004. ACM Press. 79, 96, 102, 103, 119

[134] Yingjiu Li, Vipin Swarup, and Sushil Jajodia. **Constructing a virtual primary key for fingerprinting relational data**. In Proceedings of the 3rd ACM workshop on Digital rights management (DRM '03), pages 133–141, Washington, DC, USA, October 27 2003. ACM Press. 87, 99, 101, 111

[135] Yingjiu Li, Vipin Swarup, and Sushil Jajodia. **Fingerprinting relational databases: schemes and specialties**. IEEE Transactions on Dependable and Secure Computing, **2**(1):34–45, 2005. 99, 101

[136] Jin-Cherng Lin, Jan-Min Chen, and Cheng-Hsiung Liu. **An Automatic Mechanism for Adjusting Validation Function**. In Proceedings of the 22nd International Conference on Advanced Information Networking and Applications (AINA '08), pages 602–607, Okinawa, Japan, March 25–28 2008. IEEE Computer Society. 150

[137] Siyuan Liu, Shuhong Wang, Robert H. Deng, and Weizhong Shao. **A Block Oriented Fingerprinting Scheme in Relational Database**. In Proceedings of the 7th International Conference in Information Security and Cryptology (ICISC '04), pages 455–466, Springer LNCS, Volume 3506, December 2–3 2004. Seoul, Korea. 99, 101

[138] Bo Luo, Dongwon Lee, Wang-Chien Lee, and Peng Liu. **QFilter: fine-grained run-time XML access control via NFA-based query rewriting**. In Proceedings of the 13th ACM International Conference on Information and knowledge management (CIKM '04), pages 543–552, Washington D.C., USA, November 8–13 2004. ACM Press. 121, 126, 127

[139] Isabella Mastroeni and Durica Nikolic. **Abstract Program Slicing: From Theory towards an Implementation**. In Proceedings of the 12th International Conference on Formal Engineering Methods (ICFEM '10), pages 452–467, Shanghai, China, November 17–19 2010. Springer LNCS, Volume 6447. 177, 179

[140] Isabella Mastroeni and Damiano Zanardini. **Data dependencies and program slicing: from syntax to abstract semantics**. In Proceedings of the ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation (PEPM '08), pages 125–134, San Francisco, California, USA, January 7–8, 2008. ACM Press. 6, 177, 178, 179, 180, 184, 193, 209, 211, 213, 215, 218, 220, 240

[141] Mirjana Mazuran, Elisa Quintarelli, and Letizia Tanca. **Data Mining for XML Query-Answering Support**. IEEE Transactions on Knowledge and Data Engineering, **99**(PrePrints):1041–4347, 2011. 174

[142] Russell A. McClure and Ingolf H. Kruger. **SQL DOM: Compile Time Checking of Dynamic SQL Statements**. In Proceedings of the 27th International Conference on Software Engineering (ICSE '05), pages 88–96, St. Louis, Missouri, USA, May 15–21 2005. ACM Press. 149

[143] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot. Handbook of Applied Cryptography. CRC Press, Inc., 1996. 110

[144] Mailing Meng, Xinchun Cui, and Haiting Cui. **The Approach for Optimization in Watermark Signal of Relational Databases by using Genetic Algorithms**. In Proceedings of the 2008 International Conference on Computer Science and Information Technology (ICCSIT '08), pages 448–452, Singapore, August 29–September 2 2008. IEEE Computer Society. 90

[145] Antoine Miné. **A New Numerical Abstract Domain Based on Difference-Bound Matrices**. In Proceedings of the 2nd Symposium on Programs as Data Objects (PADO '01), pages 155–172, Aarhus, Denmark, May 21–23 2001. Springer LNCS, Volume 2053. 106, 174

[146] Antoine Miné. **The octagon abstract domain**. Higher-Order and Symbolic Computation, **19**(1):31–100, 2006. 106

[147] Mod-Security. In http://www.modsecurity.org. 162

[148] Amihai Motro. **Intensional Answers to Database Queries**. IEEE Transactions on Knowledge and Data Engineering, **6**(3):444–454, 1994. 174

[149] Markus Muller-Olm and Helmut Seidl. **On optimal slicing of parallel programs**. In Proceedings of the 33rd annual ACM symposium on Theory of computing (STOC '01), pages 647–656, Hersonissos, Greece, July 6–8 2001. ACM press. 175

[150] G. B. Mund and Rajib Mall. **An efficient interprocedural dynamic slicing method**. The Journal of Systems and Software, **79**(6):791–806, 2006. 176, 179, 184

[151] Makoto Murata, Akihiko Tozawa, Michiharu Kudo, and Satoshi Hada. **XML access control using static analysis**. ACM Transactions on Information and System Security, **9**(3):292–324, 2006. 126, 127

[152] M. Muthuprasanna, Ke Wei, and Suraj Kothari. **Eliminating SQL Injection Attacks - A Transparent Defence Mechanism**. In Proceedings of the 8th IEEE International Symposium on Web Site Evolution (WSE '06), pages 22–32, Philadelphia, Pennsylvania, September 23–24 2006. IEEE Computer Society. 148, 149, 161, 162

[153] Ryohei Nakano. **Translation with Optimization from Relational Calculus to Relational Algebra Having Aggregate Functions**. ACM Transactions on Database Systems, **15**(4):518–557, 1990. 18, 20

[154] M. Negri, G. Pelagatti, and L. Sbattella. **Formal Semantics of SQL Queries**. ACM Transactions on DataBase System, **17**(3):513–534, 1991. 2, 18, 20

[155] Anh Nguyen-Tuong, Salvatore Guarnieri, Doug Greene, Jeff Shirley, and David Evans. **Automatically Hardening Web Applications Using Precise Tainting Information**. In Proceedings of the 20th IFIP International Information Security Conference (SEC '05), pages 295–308, Chiba, Japan, May 30–June 1 2005. Springer. 150

[156] Karl J. Ottenstein and Linda M. Ottenstein. **The program dependence graph in a software development environment**. ACM SIGPLAN Notices, **19**(5):177–184, 1984. 175, 182, 183, 207, 211, 219

[157] Themistoklis Palpanas and Nick Koudas. **Entropy Based Approximate Querying and Exploration of Datacubes**. In Proceedings of the 13th International Conference on Scientific and Statistical Database Management (SSDBM '01), pages 81–90, Fairfax, VA, USA, July 18–20 2001. IEEE Computer Society. 167

[158] Tadeusz Pietraszek and Chris Vanden Berghe. **Defending Against Injection Attacks through Context-Sensitive String Evaluation**. In Proceedings of Recent Advances in Intrusion Detection (RAID '05), pages 124–145, Seattle, Washington, USA, September 7–9 2005. Springer LNCS, Volume 3858. 150, 153

[159] Alain Pirotte, Dominique Roelants, and Esteban Zimányi. **Controlled Generation of Intensional Answers**. IEEE Transactions on Knowledge and Data Engineering, **3**(2):221–236, 1991. 174

[160] Andy Podgurski and Lori A. Clarke. **A Formal Model of Program Dependences and its Implications for Software Testing, Debugging, and Maintenance**. IEEE Transactions on Software Engineering, **16**(9):965–979, 1990. 6, 175, 176, 179

[161] Vidyasagar Potdar, Song Han, and Elizabeth Chang. **A survey of digital image watermarking techniques**. In Proceedings of the 3rd IEEE International Conference on Industrial Informatics (INDIN '05), pages 709–716, Peth, Australia, August 10–12 2005. IEEE Press. 77

[162] Vahab Pournaghshband. **A new watermarking approach for relational data**. In Proceedings of the 46th Annual Southeast Regional Conference on XX (ACM-SE '08), pages 127–131, Auburn, Alabama, March 28–29 2008. ACM Press. 95, 101

[163] Vahab Prasannakumari. **A Robust Tamperproof Watermarking for Data Integrity in Relational Databases**. Research Journal of Information Technology, **1**(3):115–121, 2009. 95, 101

[164] ZHU Qin, YANG Ying, LE Jia-jin, and LUO Yi-shu. **Watermark Based Copyright Protection of Outsourced Database**. In Proceedings of the 10th International Database Engineering and Applications Symposium (IDEAS'06), pages 301–308, Delhi, India, December 11–14 2006. IEEE Computer Society. 87

[165] Thomas Reps and Genevieve Rosay. **Precise interprocedural chopping**. In Proceedings of the 3rd ACM SIGSOFT Symposium on Foundations of Software Engineering (SIGSOFT '95), pages 41–52, Washington, DC, USA, October 10–13 1995. ACM Press. 183, 184

[166] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. **On Data Banks and Privacy Homomorphisms**. In Foundations of Secure Computation, pages 169–180., New York, 1978. Academic Press. 116, 118

[167] Shariq Rizvi, Alberto Mendelzon, S. Sudarshan, and Prasan Roy. **Extending query rewriting techniques for fine-grained access control**. In Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '04), pages 551–562, Paris, France, June 13–18 2004. ACM Press. 121, 125, 126

[168] Andrei Sabelfeld and Andrew C. Myers. **Language-Based Information-Flow Security**. IEEE Journal on selected areas in Communications, **21**(1):5–19, 2003. 122

[169] Diptikalyan Saha, Mangala Gowri Nanda, Pankaj Dhoolia, V. Krishna Nandivada, Vibha Sinha, and Satish Chandra. **Fault localization for data-centric programs**. In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (ESEC/FSE '11), pages 157–167, Szeged, Hungary, September 5–9 2011. ACM Press. 181

[170] Vivek Sarkar. **Automatic partitioning of a program dependence graph into parallel tasks**. IBM Journal of Research and Development, **35**(5–6):779–804, 1991. 176, 179

[171] David Scott and Richard Sharp. **Abstracting Application-level Web Security**. In Proceedings of the 11th International Conference on the World Wide Web (WWW '02), pages 396–407, Honolulu, Hawaii, USA, May 7–11 2002. ACM Press. 150

[172] Hyoung Seok Hong, Insup Lee, and Oleg Sokolsky. **Abstract Slicing: A New Approach to Program Slicing Based on Abstract Interpretation and Model Checking**. In Proceedings of the 5th IEEE International Workshop on Source Code Analysis and Manipulation (SCAM '05), pages 25–34, Budapest, Hungary, 30 September–1 October 2005. IEEE Computer Society. 177, 179

[173] Hossain Shahriar and Mohammad Zulkernine. **MUSIC: Mutation-based SQL Injection Vulnerability Checking**. In Proceedings of the 8th International Conference on Quality Software (QSIC '08), pages 77–86, Oxford, UK, August 12–13 2008. IEEE Computer Society. 150

[174] Jie Shi and Hong Zhu. **A fine-grained access control model for relational databases**. Journal of Zhejiang University - Science C, **11**(8):575–586, 2010. Zhejiang University Press, co-published with Springer. 121

[175] Jie Shi, Hong Zhu, Ge Fu, and Tao Jiang. **On the Soundness Property for SQL Queries of Fine-grained Access Control in DBMSs**. In Proceedings of the 8th IEEE/ACIS International Conference on Computer and Information Science (ICIS '09), pages 469–474, Shanghai, China, June 1–3 2009. IEEE Compueter Society. 125

[176] David P. Silberberg and Glenn E. Mitzel. **Information Systems Engineering**. Johns Hopkins APL Technical Digest, **26**(4):343–349, 2005. 9, 10

[177] Saurabh Sinha, Mary Jean Harrold, and Gregg Rothermel. **System-dependence-graph-based slicing of programs with arbitrary interprocedural control flow**. In Proceedings of the 21st International Conference on Software Engineering (ICSE '99), pages 432–441, Los Angeles, CA, USA, May 16–22 1999. ACM Press. 176, 179

[178] Radu Sion. **Proving Ownership over Categorical Data**. In Proceedings of the 20th International Conference on Data Engineering (ICDE 04), pages 584–595, Boston, MA, USA, March 30–April 2 2004. IEEE Computer Society. 93, 101

[179] Radu Sion, Mikhail Atallah, and Sunil Prabhakar. **Rights Protection for Categorical Data**. IEEE Transactions on Knowledge and Data Engineering, **17**(7):912–926, 2005. 93, 101, 103

[180] Yoga Sivagurunathan, Mark Harman, and Sebastian Danicic. **Slicing, I/O and the Implicit State**. In Proceedings of the Third International Workshop on Automatic Debugging (AADEBUG '97), pages 59–68, Linkoping, Sweden, May 26–27 1997. Linkoping Electronic Articles in Computer and Information Science. 180

[181] Zhendong Su and Gary Wassermann. **The Essence of Command Injection Attacks in Web Applications**. In Proceedings of the 33rd Annual Symposium on Principles of Programming Languages (POPL '06), pages 372–382, Charleston, South California, USA, January 11–13 2006. ACM Press. 148

[182] Srihari Sukumaran, Ashok Sreenivas, and Ravindra Metta. **The dependence condition graph: Precise conditions for dependence between program points**. Computer Languages, Systems & Structures, **36**(1):96–121, 2010. 6, 178, 179, 180, 184, 193, 195, 198, 219

REFERENCES

[183] Hee Beng Kuan Tan and Tok Wang Ling. **Correct Program Slicing of Database Operations**. IEEE Software, **15**:105–112, 1998. 180, 222

[184] Kian-Lee Tan, Mong Li Lee, and Wang Wang. **Access Control of XML Documents in Relational Database Systems**. In Proceedings of the International Conference on Internet Computing (IC '01), pages 185–191, Las Vegas, Nevada, USA, June 25–28 2001. CSREA Press. 126, 127

[185] Reps Thomas and Yang Wuu. **The semantics of program slicing**. Technical report, University of Wisconsin, 1988. 182

[186] Meng-Hsiun Tsai, Fang-Yu Hsu, Jun-Dong Chang, and Hsien-Chu Wu. **Fragile Database Watermarking for Malicious Tamper Detection Using Support Vector Regression**. In Proceedings of the 3rd International Conference on International Information Hiding and Multimedia Signal Processing (IIH-MSP '07), pages 493–496, Splendor Kaohsiung, Taiwan, November 26–28 2007. IEEE Computer Society. 90, 101

[187] Meng-Hsiun Tsai, Hsiao-Yun Tseng, and Chen-Ying Lai. **A Database Watermarking Technique for Temper Detection**. In Proceedings of the 2006 Joint Conference on Information Sciences (JCIS '06), Kaohsiung, Taiwan, October 8–11 2006. Atlantis Press. 79, 96, 102, 119

[188] Fredrik Valeur, Darren Mutz, and Giovanni Vigna. **A Learning-Based Approach to the Detection of SQL Attacks**. In Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA '05), pages 123–140, Vienna, Austria, July 7–8 2005. Springer LNCS, Volume 3548. 149

[189] G. A. Venkatesh. **The semantic approach to program slicing**. ACM SIGPLAN Notices, **26**(6):107–119, 1991. 179

[190] Chaokun Wang, Jianmin Wang, Ming Zhou, Guisheng Chen, and Deyi Li. **ATBaM: An Arnold Transform Based Method on Watermarking Relational Data**. In Proceedings of the 2008 International Conference on Multimedia and Ubiquitous Engineering (MUE '08), pages 263–270, Beijing, China, May 07 2008. IEEE Computer Society. 89, 101

[191] Haiqing Wang, Xinchun Cui, and Zaihui Cao. **A Speech Based Algorithm for Watermarking Relational Databases**. In Proceedings of the 2008 International Symposiums on Information Processing (ISIP '08), pages 603–606, Moscow, Russia, May 23–25 2008. IEEE Computer Society. 90, 101

[192] Qihua Wang, Ting Yu, Ninghui Li, Jorge Lobo, Elisa Bertino, Keith Irwin, and Ji-Won Byun. **On the correctness criteria of fine-grained access control in relational databases**. In Proceedings of the 33rd international conference on Very large data bases (VLDB '07), pages 555–566, Vienna, Austria, September 23–27 2007. VLDB Endowment. 4, 121, 125, 128, 131, 135

[193] Mark Weiser. **Program slicing**. IEEE Transactions on Software Engineering, **SE-10**(4):352–357, 1984. 175, 179, 183

[194] David Willmor, Suzanne M. Embury, and Jianhua Shao. **Program Slicing in the Presence of a Database State**. In Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM '04), pages 448–452, Chicago, IL, USA, September 11–17 2004. IEEE Computer Society. 176, 180, 182, 221, 222

[195] Glynn Winskel. The Formal Semantics of Programming Languages: An Introduction. The MIT Press, 1993. 185

[196] Jing Wu, Jennifer Seberry, Yi Mu, and Chun Ruan. **Delegatable Access Control for Fine-Grained XML**. In Proceedings of the 11th International Conference on Parallel and Distributed Systems (ICPADS '05), pages 270–274, Fuduoka, Japan, July 20–22 2005. IEEE Computer Society. 127

[197] Xiangrong Xiao, Xingming Sun, and Minggang Chen. **Second-LSB-Dependent Robust Watermarking for Relational Database**. In Proceedings of the 3rd International Symposium on Information Assurance and Security (IAS '07), pages 292–300, Manchester, United Kingdom, August 29–31 2007. IEEE Computer Society. 88

[198] Suk-Chung Yoon and E. K. Park. **An Approach to Intensional Query Answering at Multiple Abstraction Levels Using Data Mining Approaches**. In Proceedings of the 32 Annual Hawaii International Conference on System Sciences (HICSS '99), pages 1–9, Maui, Hawaii, USA, January 5–8 1999. IEEE Computer Society. 174

[199] Ting Yu, Divesh Srivastava, Laks V. S. Lakshmanan, and H. V. Jagadish. **Compressed accessibility map: efficient access control for XML**. In Proceedings of the 28th international conference on Very Large Data Bases (VLDB '02), pages 478–489, Hong Kong, China, August 20–23 2002. VLDB Endowment. 127

[200] Damiano Zanardini. **The Semantics of Abstract Program Slicing**. In Proceedings of the IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM '08), pages 89–100, Beijing, China, September 2008. IEEE Press. 177, 179

[201] Yong ZHANG, Xiamu NIU, and Dongning ZHAO. **A Method of Protecting Relational Databases Copyright with Cloud Watermark**. International Journal of Information and Communication Engineering, **1**(7):337–341, 2005. 91, 101

[202] Yong Zhang, Xiamu Niu, Dongning Zhao, Juncao Li, and Siming Liu. **Relational Databases Watermark Technique Based on Content Characteristic**. In Proceedings of the 1st International Conference on Innovative Computing, Information and Control (ICICIC '06), pages 677–680, Beijing, China, August 30–September 1 2006. IEEE Computer Society. 79, 91, 101, 109, 119

[203] Xiang Zhou, Min Huang, and Zhiyong Peng. **An additive-attack-proof watermarking mechanism for databases' copyrights protection using image**. In Proceedings of the 2007 ACM symposium on Applied computing (SAC '07), pages 254–258, Seoul, Korea, March 11–15 2007. ACM Press. 89, 101

[204] Hong Zhu and Kevin Lu. **Fine-Grained Access Control for Database Management Systems**. In Proceedings of the 24th British National Conference on Databases, pages 215–223, Glasgow, UK, July 3–5 2007. Springer LNCS, Volume 4587. 121, 125

[205] Hong Zhu, Jie Shi, Yuanzhen Wang, and Yucai Feng. **Controlling Information Leakage of Fine-Grained Access Model in DBMSs**. In Proceedings of the 9th International Conference on Web-Age Information Management (WAIM '08), pages 583–590, Zhangjiajie, China, July 20–22 2008. IEEE Computer Society. 121, 125