Formal Analysis and Verification of Database Query Languages

Raju Halder

Indian Institute of Technology Patna
and
HASLab/INESC TEC & University of Minho

halder@iitp.ac.in, raju.halder@inesctec.pt

Outline

Research Contributions

Research Interests: Formal Methods (Abstract Interpretation, Model Checking, etc.), Program Analysis/Verification, Programming Languages, Databases, etc.

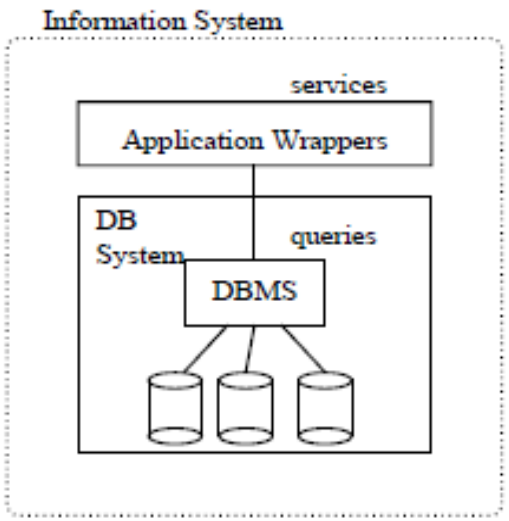
1. Formalized Concrete and Abstract Semantics of Database Query Language (SQL/HQL).
2. Language-based Information Flow Security Analysis of SQL/HQL.
3. Cooperative Query Answering/Observation-based FGAC/Persistent Watermarking/Data Cleaning, etc.
4. Verification of Hibernate Query Language (HQL) by Abstract Interpretation.
5. Semantic-based Refinement of Data/Control Dependences:
   - Dependence Graphs.
   - Slicing Algorithms.

New Applicative Scenarios: Information Systems

Definition (Information Systems)

Information Systems (IS) is a generic term referring to software and hardware systems that support data-intensive applications and provide information-based services.

- Information systems for most organizations are currently supported by databases.



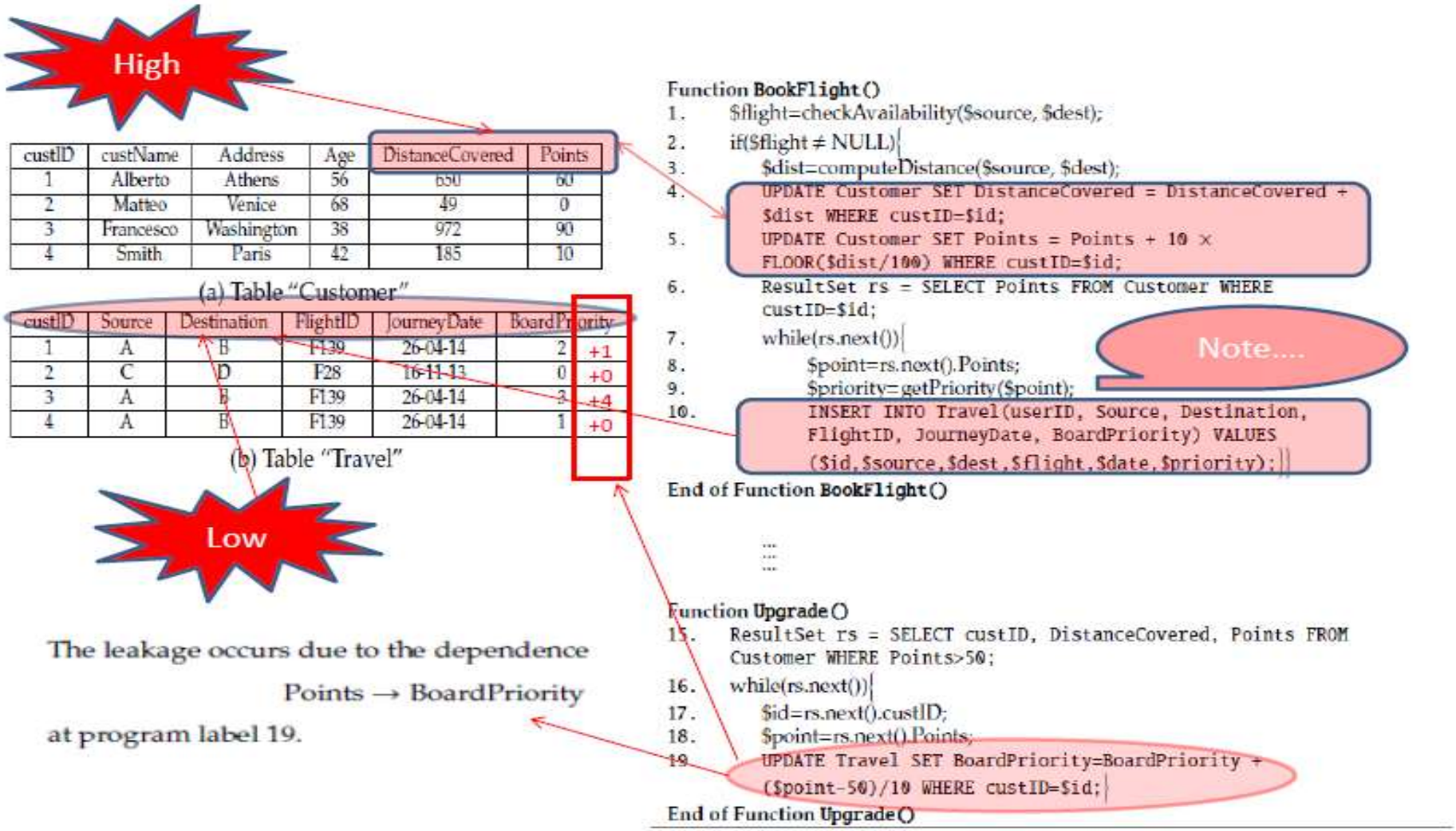
Information Systems: Challenging Issues

1. An exponentially increasing amount of data - finitely large/unbounded.
2. Dealing with dynamically changing content - hence behavior of the system.
3. Facing serious challenges while managing, processing, analyzing, or understanding large volume of data in restricted environments.
4. Optimization issues are really under big threat.
5. Advanced intelligent techniques must be exploited to
   - simplifying data processing, analyzing, etc....
   - reducing computational complexity..
   - addressing several key issues like soundness, correctness, scalability, etc....
   - . . . . . .

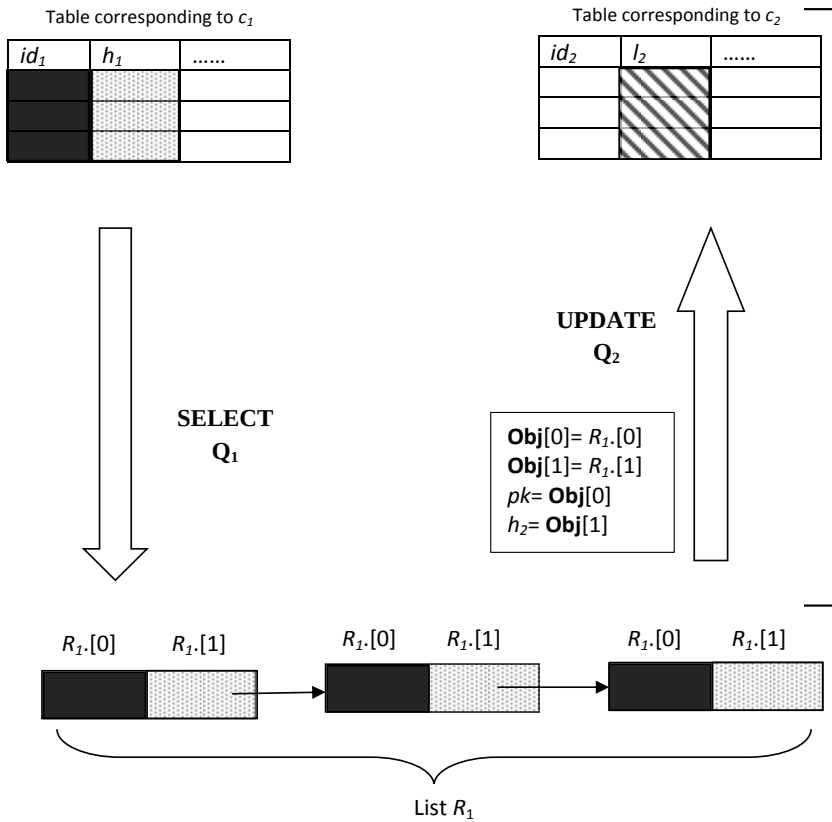
Extending Abstract Interpretation theory

- Reducing computational complexity
  - Dealing with finitely large/unbounded data sets.
  - Simplified approach to managing, processing, analyzing, or understanding data in very large databases.
- Collect dynamic behavioral properties of the systems during static time.
  - Dealing with dynamically changing data.
  - Semantics-based static analysis framework for applications accessing or manipulating databases, etc.
- Addressing security issues, like fine-grained access control, watermarking, information flow analysis, etc.
- Supporting Intelligent Answering System, such as Cooperative Query Answering, Intensional Query Answering, Approximate Query Answering.

Lang.-based Information Flow Analysis of SQL



Lang.-based Information Flow Analysis of HQL




```
Session session = getSessionFactory().openSession();
Transaction tx =session.beginTransaction();
Query Q1 = session.createQuery("SELECT id1 , h1 FROM c1 ");
List R1 = Q1.list();
for(Object[] obj:R1 ){
    pk=(Int) obj[0];
    h2=(Int) obj[1];
    Query Q2 = session.createQuery("UPDATE c2 SET l2 = l2 +1
                WHERE id2 = pk AND h2=1000");
    int result = Q2.executeUpdate();}
tx.commit();
session.close();
```

Cooperative Query Answering

- Example: Online Flight Reservation System

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

- Find `flights` from `''Rome Ciampino''` to `''Paris Orly''` that reach destination within `8:00 a.m.` and `11:00 a.m.`
  - **Traditional Query Answer:** No Flight Available....
  - **Cooperative query answer:** More Useful Information

Cooperative Answer

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

- Addressed Key Issues: Soundness, Relevancy, Optimality

Observation-based Fine Grained Access Control (OFGAC)

- Traditional FGAC provides two extreme views: either public or private
- What about partial sensitive data? *e.g.* Credit Card No. : **** **** **** 7432
- Introduced OFGAC based on the Abstract Interpretation framework
- Provides partial or relaxed view for partial sensitive information
- Traditional FGAC is a special case of OFGAC.

eID	*Name*	*Age*	*Dno*	*Sal*
1	Matteo (N)	30	2 (N)	2800 (N)
2	Pallab (N)	22	1 (N)	1500
3	Sarbani (N)	56 (N)	2 (N)	2300
4	Luca (N)	35	1 (N)	6700 (N)
5	Tanushree (N)	40	3 (N)	4900
6	Andrea (N)	52 (N)	1 (N)	7000 (N)
7	Alberto (N)	48	3 (N)	800
8	Mita (N)	29 (N)	2 (N)	4700 (N)

Table : Concrete Database

$eID^\sharp$	$Name^\sharp$	$Age^\sharp$	$Dno^\sharp$	$Sal^\sharp$
1	Male	30	$\langle \beta_1, [1,2] \rangle$	Medium
2	Male	22	$\langle \beta_2, [1,2] \rangle$	1500
3	Female	[50,59]	$\langle \beta_1, [1,2] \rangle$	2300
4	Male	35	$\langle \beta_2, [1,2] \rangle$	Very high
5	Female	40	$\langle \beta_3, [3,4] \rangle$	4900
6	Male	[50,59]	$\langle \beta_2, [1,2] \rangle$	Very high
7	Male	48	$\langle \beta_3, [3,4] \rangle$	800
8	Female	[20,29]	$\langle \beta_1, [1,2] \rangle$	High

Table : Abstract Database

- Collusion attacks in 3 different scenarios:
  - Multiple Policies/Single Level Abstraction
  - Single Policy/Multiple Level Abstraction
  - Multiple Policies /Multiple Level Abstractions

Abstract Fix-point Computation and Widening Operator

- Returns all the flight destinations reachable from the specified airport "Marcopolo (VCE)" along with the number of connections and the total cost to arrive at that final destination.

departure	arrival	connects	cost
Marcopolo (VCE)	Orly (ORY)	0	325.30
Marcopolo (VCE)	Zurich (ZRH)	1	525.30
Marcopolo (VCE)	Fiumicino (FCO)	1	753.58
Marcopolo (VCE)	Marcopolo (VCE)	2	775.45
Marcopolo (VCE)	Madrid-Barajas (MAD)	2	1164.15
Marcopolo (VCE)	Orly (ORY)	3	1100.75
Marcopolo (VCE)	Marcopolo (VCE)	3	1414.30

Concrete Result : infinite

- Defined Widening Operator for database domain to accelerate convergence in abstract domain

$\widetilde{departure}$	$\widetilde{arrival}$	$\widetilde{connects}$	$\widetilde{cost}$
Marcopolo (VCE)	Orly (ORY)	$[0, \infty]$	$[325.30, \infty]$
Marcopolo (VCE)	Zurich (ZRH)	$[1, \infty]$	$[525.30, \infty]$
Marcopolo (VCE)	Fiumicino (FCO)	$[1, \infty]$	$[753.58, \infty]$
Marcopolo (VCE)	Marcopolo (VCE)	$[2, \infty]$	$[775.45, \infty]$

Abstract Result : fixpoint solution using widening operator

Database Cleaning

Bertosi et al., Data cleaning and query answering with matching dependencies and matching functions. In Proc. of ICDT'11. ACM Press.

MD: ψ_1 ={Name, Address, Company}→{Company}

Dirty database:

Name	Address	Company
Caeser Doe	Main St Ottawa	IBM India
C Doe	Main St	IBM
Christian Doe	25 Main St	IBM International
Peter Christian	25 MG Road Patna	Samsung Electronics
Peter Christ	MG Road Patna	Samsung

Cleaned Database Instances:

Name	Address	Company
Caeser Doe	Main St Ottawa	IBM India
C Doe	Main St	IBM India
Christian Doe	25 Main St	IBM International
Peter Christian	25 MG Road Patna	Samsung Electronics
Peter Christ	MG Road Patna	Samsung Electronics

Name	Address	Company
Caeser Doe	Main St Ottawa	IBM India
C Doe	Main St	IBM International
Christian Doe	25 Main St	IBM International
Peter Christian	25 MG Road Patna	Samsung Electronics
Peter Christ	MG Road Patna	Samsung Electronics

Table : Concrete clean instances after applying the approach of **Bertosi et al.**

Database Cleaning

- Single abstract Clean Instance.
- Reducing time and space complexities.
- Certain and Possible Answers.

Abstract Cleaned Instance:

Name$^\sharp$	**Address**$^\sharp$	**Company**$^\sharp$
$[C]^{1,1}[aeser]^{0,1}[Doe]^{1,1}$	$[Main]^{1,1}[St]^{1,1}[Ottawa]^{0,1}$	$[IBM]^{1,1}[India, International]^{0,1}$
$[C]^{1,1}[aeser, hristian]^{0,1}[Doe]^{1,1}$	$[25]^{0,1}[Main]^{1,1}[St]^{1,1}[Ottawa]^{0,1}$	$[IBM]^{1,1}[India, International]^{0,1}$
$[C]^{1,1}[hristian]^{0,1}[Doe]^{1,1}$	$[25]^{0,1}[Main]^{1,1}[St]^{1,1}$	$[IBM]^{1,1}[India, International]^{0,1}$
$[Peter]^{1,1}[Christ]^{1,1}[ian]^{0,1}$	$[25]^{0,1}[MG]^{1,1}[Road]^{1,1}[Patna]^{1,1}$	$[Samsung]^{1,1}[Electronics]^{0,1}$
$[Peter]^{1,1}[Christ]^{1,1}[ian]^{0,1}$	$[25]^{0,1}[MG]^{1,1}[Road]^{1,1}[Patna]^{1,1}$	$[Samsung]^{1,1}[Electronics]^{0,1}$

Table : **Abstract clean instance after applying ψ_1**

Verification of HQL programs for Enterprise Policies defined on Persistent Objects

- **Examples of Enterprise Policies**

  Policy 1: *Employees age should be greater than or equal to 18 and less than or equal to 62.*
  Policy 2: *The salary of employees with age greater than 30 should be at least 1500 euro.*
  Policy 3: *Employees salary should not be more than three times of the lowest salary.*

Verification of HQL programs

```
1.  public class ExClass{
2.      public static void main(String[] args)
        { ...
    % Creating emp object and stores into database %
3.          emp obj=new emp( );
4.          obj.setId(4);
5.          obj.setage(32);
6.          obj.setdno(1);
7.          obj.setsalary(1000);
8.          ses.save( obj );

    % Updating persistent emp objects %
9.          Query q1 = ses.createQuery("
            UPDATE emp e SET e.age= e.age+1,
            e.sal= e.sal + :inc×2 WHERE
            e.sal > 1600");
10.         q1.setParameter("inc",100);
11.         int r1 = q1.executeUpdate();

    % Selecting from persistent emp objects %
12.         Query q2 = ses.createQuery("
            SELECT e.dno, MAX(e.sal), AVG(
            DISTINCT e.age) FROM emp e
            WHERE e.sal ≥ 1000 GROUP BY
            e.dno HAVING MAX (e.sal) < 4000
            ORDER BY e.dno");
13.         List r2 = q2.list();

    % Deleting persistent emp objects %
14.         Query q3 = ses.createQuery("
            DELETE FROM emp e WHERE
            e.age > 50");
15.         int r3 = q3.executeUpdate(); }}
```

tid	tage	tdno	tsal
1	35	3	1600
2	19	2	900
3	50	3	2550

• After executing statement 8 (save() method).

tid	tage	tdno	tsal
1	35	3	1600
2	19	2	900
3	50	3	2550
4	32	1	1000

• After executing statements 9-11.

tid	tage	tdno	tsal
1	35	3	1600
2	19	2	900
3	51	3	2750
4	32	1	1000

• After executing statements 12-13 (no change in database).

tid	tage	tdno	tsal
1	35	3	1600
2	19	2	900
3	51	3	2750
4	32	1	1000

• After executing statements 14-15.

tid	tage	tdno	tsal
1	35	3	1600
2	19	2	900
4	32	1	1000

Verification of HQL programs

- Domain of intervals: $\texttt{INT}$.
- Consider objects state and database state, we choose the abstract domain $\texttt{D}^\sharp = \texttt{INT} \times \texttt{INT}$.
- The set of abstract traces of `Session` object 'ses' in the program is

$$\mathcal{T}^\sharp = \{\ \tau^\sharp\ \} = \left\{\sigma_0^\sharp \xrightarrow{ses.\texttt{SAVE}^\sharp()} \sigma_1^\sharp \xrightarrow{ses.\texttt{UPD}^\sharp()} \sigma_2^\sharp \xrightarrow{ses.\texttt{SEL}^\sharp()} \sigma_3^\sharp \xrightarrow{ses.\texttt{DEL}^\sharp()} \sigma_4^\sharp\right\}$$

where

$$\sigma_1^\sharp = \langle [32,32], [19,50] \rangle$$
$$\sigma_2^\sharp = \langle [32,32], [19,51] \rangle$$
$$\sigma_3^\sharp = \langle [32,32], [19,51] \rangle$$
$$\sigma_4^\sharp = \langle [32,32], [19,50] \rangle$$

- According to the abstract projected collecting semantics, we get

$$\texttt{Projection}^\sharp[\![\mathcal{T}^\sharp]\!](\texttt{SAVE}^\sharp()) = \{\sigma_1^\sharp\}$$
$$\texttt{Projection}^\sharp[\![\mathcal{T}^\sharp]\!](\texttt{UPD}^\sharp()) = \{\sigma_2^\sharp\}$$
$$\texttt{Projection}^\sharp[\![\mathcal{T}^\sharp]\!](\texttt{SEL}^\sharp()) = \{\sigma_3^\sharp\}$$
$$\texttt{Projection}^\sharp[\![\mathcal{T}^\sharp]\!](\texttt{DEL}^\sharp()) = \{\sigma_4^\sharp\}$$

Applications:

- **Refinement of dependences by removing false alarms in Database Applications...**
  - Considering relational abstract domain.
  - Identifying *defined* and *used*-parts by database statements.
  - Refinement of Dependence Graphs.
  - Slicing of Database Programs.

Refinement of Abstract Program Slicing Techniques: Example

- Slicing Criteria: ⟨13, y, SIGN⟩

1.	start
2.	$i_1 = -2;$
3.	$x_1 = input;$
4.	$y_1 = input;$
5.	$w = input;$
6.	$if(x_1 \geq 0)$
7.	$y_2 = 4 \times (x_1)^3;$
ϕ_1	$y_3 = f(y_1, y_2);$
	$while($
ϕ_2	$(x_2, i_2) = f((x_1, i_1), (x_3, i_3));$
8.	$i_2 \leq 0$
	$)\{$
9.	$x_3 = x_2 \times 2;$
10.	$i_3 = i_2 + 1;\}$
11.	$if(x_2 \leq 0)$
12.	$y_4 = (x_2)^2 + 4w \ mod \ 2;$
ϕ_3	$y_5 = f(y_3, y_4);$
13.	$print(x_2, y_5);$
14.	stop

Table : Program P_{ssa}

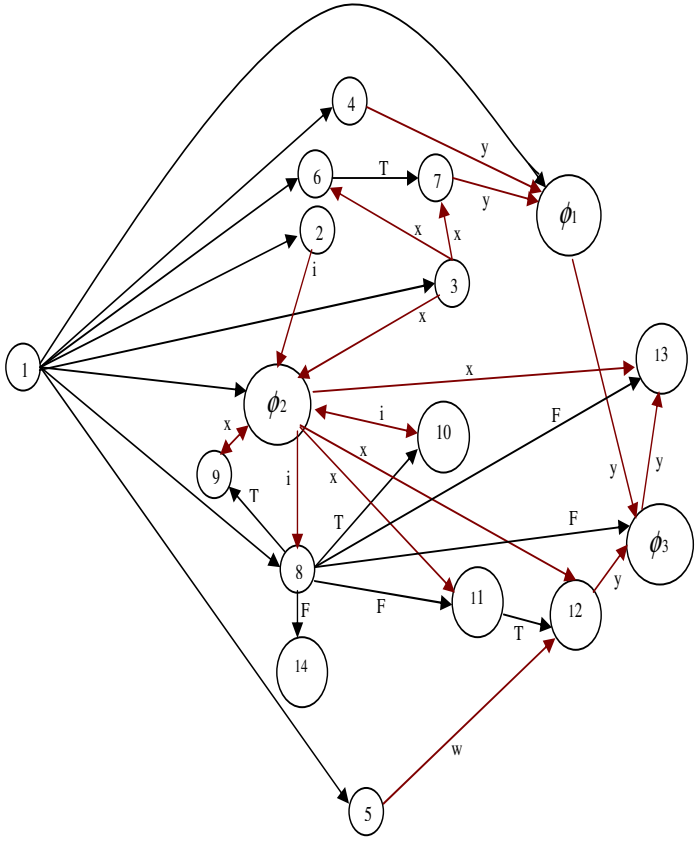


Figure : PDG of P_{ssa}

Refinement of Abstract Program Slicing Techniques: Example

- **Computation of Semantic Relevancy of Statements.**

1.	start
2.	$i_1 = -2;$
3.	$x_1 = input;$
4.	$y_1 = input;$
5.	$w = input;$
6.	$if(x_1 \geq 0)$
7.	$y_2 = 4 \times (x_1)^3;$
ϕ_1	$y_3 = f(y_1, y_2);$
	$while($
ϕ_2	$(x_2, i_2) = f((x_1, i_1), (x_3, i_3));$
8.	$i_2 \leq 0$
	$)\{$
9.	$x_3 = x_2 \times 2;$
10.	$i_3 = i_2 + 1;\}$
11.	$if(x_2 \leq 0)$
12.	$y_4 = (x_2)^2 + 4w \ mod \ 2;$
ϕ_3	$y_5 = f(y_3, y_4);$
13.	$print(x_2, y_5);$
14.	stop

Table : Program P_{ssa}

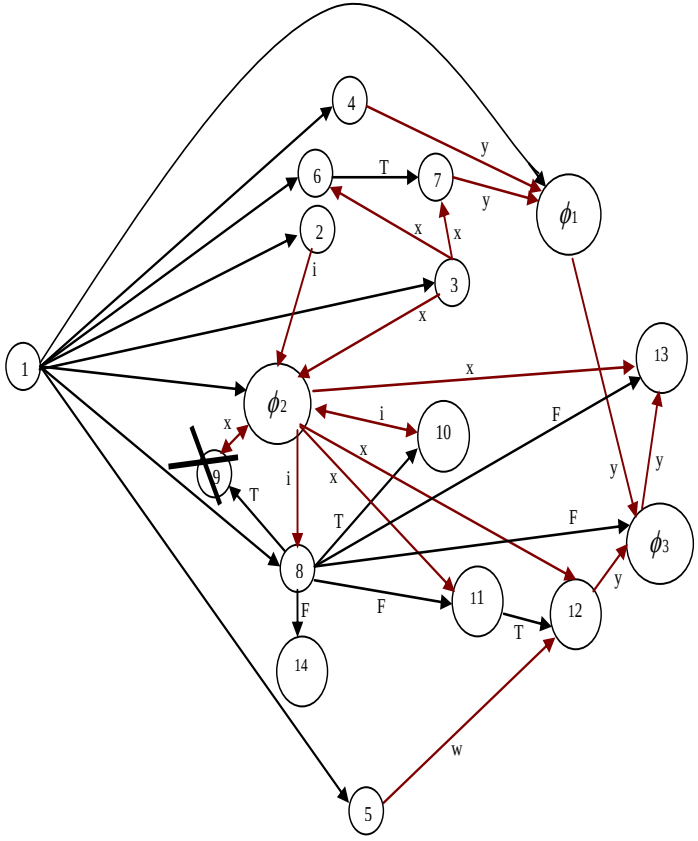


Figure : PDG after computing Semantic Relevancy w.r.t. SIGN

Refinement of Abstract Program Slicing Techniques: Example

- **Computation of Semantic Data Dependency.**

1.	start
2.	$i_1 = -2;$
3.	$x_1 = input;$
4.	$y_1 = input;$
5.	$w = input;$
6.	$if(x_1 \geq 0)$
7.	$y_2 = 4 \times (x_1)^3;$
ϕ_1	$y_3 = f(y_1, y_2);$
	$while($
ϕ_2	$(x_2, i_2) = f((x_1, i_1), (x_3, i_3));$
8.	$i_2 \leq 0$
	$)\{$
9.	$x_3 = x_2 \times 2;$
10.	$i_3 = i_2 + 1;\}$
11.	$if(x_2 \leq 0)$
12.	$y_4 = (x_2)^2 + 4w \ mod \ 2;$
ϕ_3	$y_5 = f(y_3, y_4);$
13.	$print(x_2, y_5);$
14.	stop

Table : Program P_{ssa}

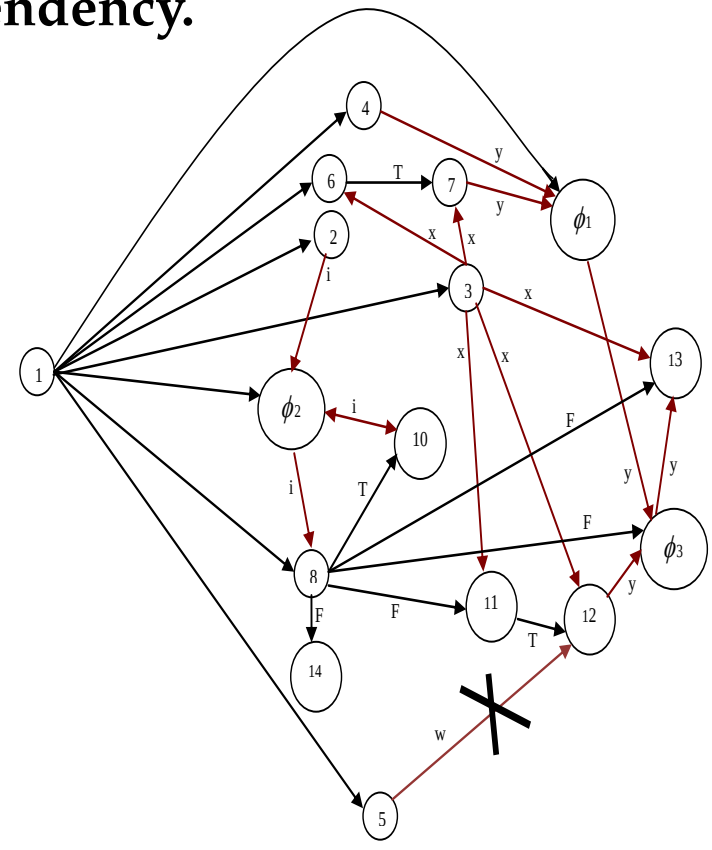
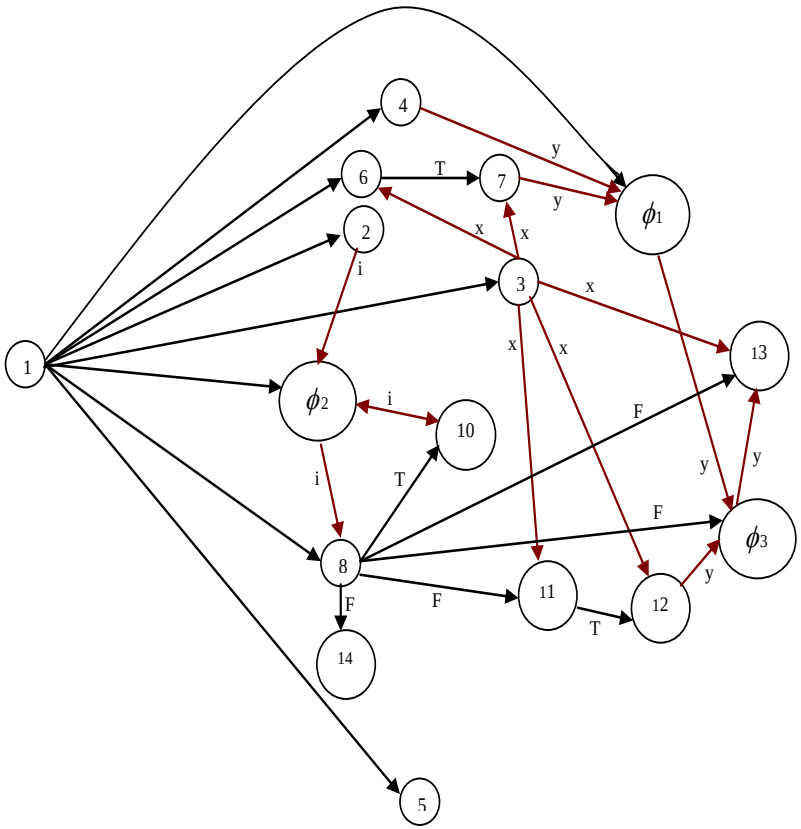


Figure : Computing Semantic Relevancy first, and then Semantic Data Dependency w.r.t. SIGN

Refinement of Abstract Program Slicing Techniques: Example

- **Computation of Conditional Dependency: compute DCG annotations**



e	e^R	e^A
$4 \xrightarrow{y} \phi_1$	$\emptyset$	$1 \xrightarrow{true} 6 \xrightarrow{true} 7$
$7 \xrightarrow{y} \phi_1$	$\emptyset$	$\emptyset$
$3 \xrightarrow{x} 6$	$\emptyset$	$\emptyset$
$3 \xrightarrow{x} 7$	$1 \xrightarrow{true} 6 \xrightarrow{true} 7$	$\emptyset$
$3 \xrightarrow{x} 11$	$1 \xrightarrow{true} 8 \xrightarrow{false} 11$	$\emptyset$
$3 \xrightarrow{x} 12$	$1 \xrightarrow{true} 8 \xrightarrow{false} 11 \xrightarrow{true} 12$	$\emptyset$
$3 \xrightarrow{x} 13$	$1 \xrightarrow{true} 8 \xrightarrow{false} 13$	$\emptyset$
$12 \xrightarrow{y} \phi_3$	$\emptyset$	$\emptyset$
$\phi_1 \xrightarrow{y} \phi_3$	$1 \xrightarrow{true} 8 \xrightarrow{false} \phi_3$	$1 \xrightarrow{true} 8 \xrightarrow{false} 11 \xrightarrow{true} 12$
$\phi_3 \xrightarrow{y} 13$	$\emptyset$	$\emptyset$
$2 \xrightarrow{i} \phi_2$	$\emptyset$	$\emptyset$
$\phi_2 \xrightarrow{i} 8$	$\emptyset$	$\emptyset$
$10 \xrightarrow{i} \phi_2$	$\emptyset$	$\emptyset$
$\phi_2 \xrightarrow{i} 10$	$1 \xrightarrow{true} 8 \xrightarrow{true} 10$	$\emptyset$

Table : *DependenceConditionGraphs(DCG)*

Refinement of Abstract Program Slicing Techniques: Example

- **Computation of Conditional Dependency: remove unrealizable dependences**

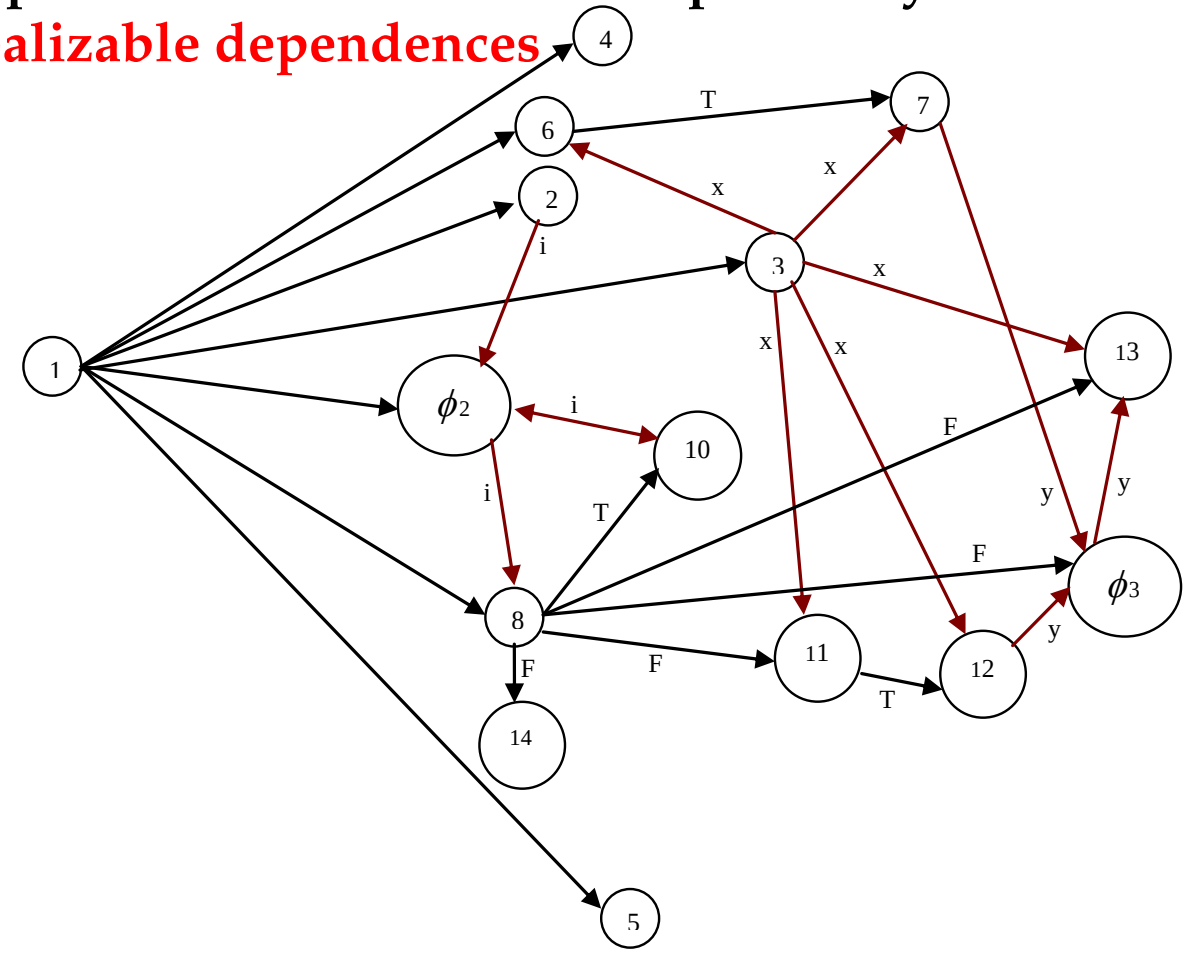


Figure : Semantics-based abstract DCG

Refinement of Abstract Program Slicing Techniques: Example

- **Slicing w.r.t. ⟨13, y, SIGN⟩.**

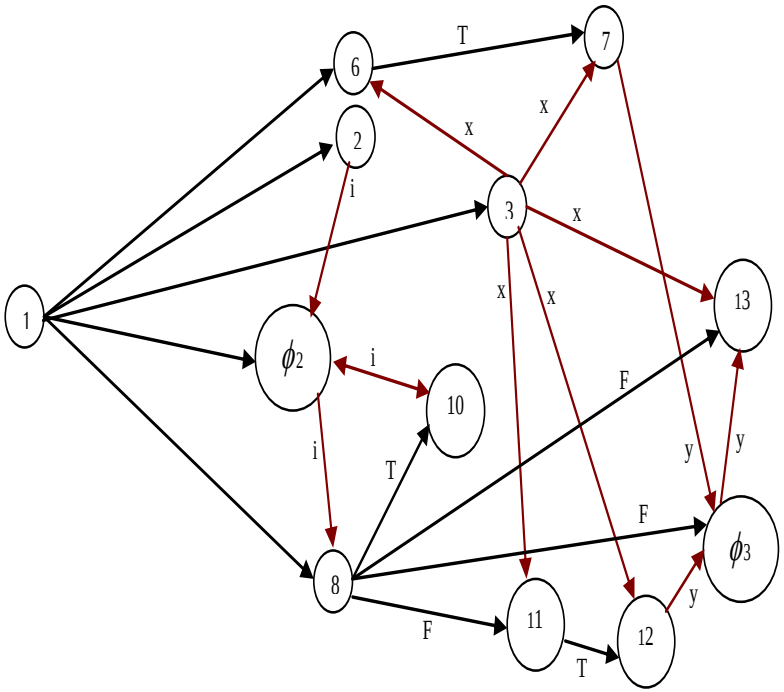


Figure : sub-DCG after traversing backward *w.r.t.* ⟨13, *y*⟩

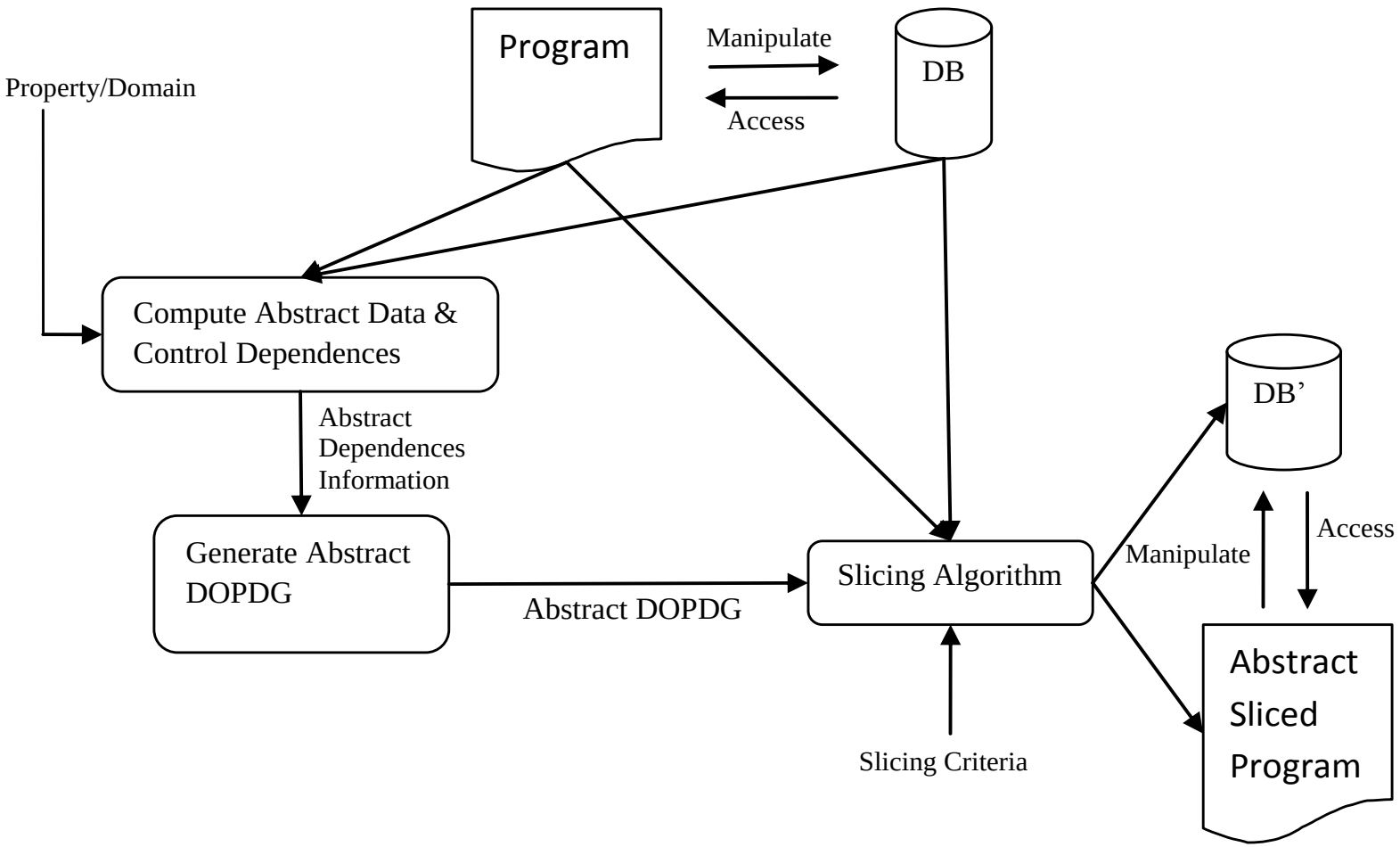
1.	start
2.	$i = -2;$
3.	$x = input;$
6.	$if(x \geq 0)$
7.	$y = 4x^3;$
8.	while$(i \leq 0)\{$
10.	$i = i + 1;\}$
11.	$if(x \leq 0)$
12.	$y = x^2 + 4w \ mod \ 2;$

Table : Slice w.r.t. ⟨13, *y*, *SIGN*⟩

Slicing of Database Query Languages

- Extended the proposed slicing refinement to the applications accessing/manipulating databases.
- Database-Oriented Program Dependence Graphs (DOPDGs)
  - Database-Database Dependences (DD-Dependences)
  - Program-Database Dependences (PD-Dependences)

Slicing of Database Query Languages



Slicing of Database Query Languages: example

Abstract domain $ABSJOB = \{TechStaff, AdminStaff\}$

$\gamma(TechStaff)=\{Programmer, Analyst, Project\ Manager\}$

$\gamma(AdminStaff)=\{Receptionist, Secretary, HR\}$

Table : Database dB

ID	name	age	locID	jobname
1	Alberto	28	2	Programmer
2	Matteo	30	1	HR
3	Francesco	35	3	Analyst

(a) Table "citizen"

locID	locname
1	Venice
2	Marghera
3	Mestre

(b) Table "location"

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

(c) Table "job"

Table : Abstract Database $dB^\sharp$

ID	name	age	locID	jobname
1	Alberto	28	2	TechStaff
2	Matteo	30	1	AdminStaff
3	Francesco	35	3	TechStaff

(a) Table "citizen$^\sharp$"

locID	locname
1	Venice
2	Marghera
3	Mestre

(b) Table "location$^\sharp$"

jobname	category	rank	sal
TechStaff	A	[1, 3]	[1500, 2100]
AdminStaff	B	[1, 3]	[1000, 2000]

(c) Table "job$^\sharp$"

Slicing of Database Query Languages: example

Figure : Program P

```
...
6.  ResultSet rs1=myStmt.executeQuery("SELECT category, rank FROM job WHERE jobname=$oldJob");
7.  ResultSet rs2=myStmt.executeQuery("SELECT category, rank FROM job WHERE jobname=$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);
...
```

Slicing of Database Query Languages: example

Abstract Slicing Criteria: $\langle 12, rs, Ival \rangle$

where *"Ival"* includes the abstract domain *"ABSJOB"* and *"domain of Intervals"*

```
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);
```

(a) abstract slice *w.r.t.* $\langle 12, rs, Ival \rangle$ of P

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

(b) Part of the database relevant to the slice

Slicing of Database Query Languages: example

- Similarly, we apply the notions of semantic data dependences and conditional dependences.

```
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.    ....
```

- Here, the expression *"oldcol+newcol"* is not semantically data dependent on the attribute *"newcol"* at all.

Database Applications: Introduction

- Two sets of variables:
  - Application variables ($\mathbb{V}_a$)
  - Database Variables ($\mathbb{V}_d$)
- We denote any SQL statement $C_{sql} = \langle A_{sql}, \phi \rangle$
- We call the first component A_{sql} the *operation* and the second component ϕ the *condition* of C_{sql}.
- The condition ϕ in SQL statements is a well-formed formula of first-order logic.
- In an abstract sense, C_{sql} first identifies a data set from the database using the condition ϕ and then performs the appropriate operations on that data set using A_{sql}.

Abstract Syntax of SQL

Constants:			
k	$\in$	$\mathbb{K}$	Set of Constants
k	::=	$n \mid s$	
		where $n \in$ Integers and $s \in$ Strings	
Variables:			
v_a	$\in$	$\mathbb{V}_a$	Set of Application Variables
v_a	::=	$x \mid y \mid z \mid \ldots$	
v_d	$\in$	$\mathbb{V}_d$	Set of Database Attributes
v_d	::=	$a_1 \mid a_2 \mid a_3 \mid \ldots$	
Expressions:			
e	$\in$	$\mathbb{E}$	Set of Arithmetic Expressions
e	::=	$k \mid v_d \mid v_a \mid op_u\ e \mid e_1\ op_b\ e_2$	
		where $op_u \in \{+, -\}$ and $op_b \in \{+, -, *, /\}$	
b	$\in$	$\mathbb{B}$	Set of Boolean Expressions
b	::=	true $\mid$ false $\mid e_1\ op_r\ e_2 \mid \neg b \mid b_1 \oplus b_2$	
		where $op_r \in \{\leq, \geq, ==, >, \neq, \ldots\}$ and $\oplus \in \{\vee, \wedge\}$	
SQL conditions:			
τ	$\in$	$\mathbb{T}$	Set of Terms
τ	::=	$k \mid v_a \mid v_d \mid f_n(\tau_1, \tau_2, ..., \tau_n)$	
		where f_n is an n-ary function.	
a_f	$\in$	$\mathbb{A}_f$	Set of Atomic Formulas
a_f	::=	$R_n(\tau_1, \tau_2, ..., \tau_n) \mid \tau_1 == \tau_2$	
		where $R_n(\tau_1, \tau_2, ..., \tau_n) \in \{true, false\}$	
ϕ	$\in$	$\mathbb{W}$	Set of conditions
ϕ	::=	$a_f \mid \neg\phi \mid \phi_1 \oplus \phi_2 \mid \otimes v\ \phi$	
		where $\oplus \in \{\vee, \wedge\}$ and $\otimes \in \{\forall, \exists\}$	

Abstract Syntax of SQL

SQL Functions:

$g(\vec{e})$	::=	`GROUP BY`$(\vec{e})$ | id	where $\vec{e} = \langle e_1, ..., e_n \mid e_i \in \mathbb{E} \rangle$
r	::=	`DISTINCT` | `ALL`	
s	::=	`AVG` | `SUM` | `MAX` | `MIN` | `COUNT`	
$h(e)$	::=	$s \circ r(e)$ | `DISTINCT`(e) | id	
$h(*)$	::=	`COUNT(*)`	where * represents a list of database attributes denoted by $\vec{v_d}$
$\vec{h}(\vec{x})$	::=	$\langle h_1(x_1), ..., h_n(x_n) \rangle$	where $\vec{h} = \langle h_1, ..., h_n \rangle$ and $\vec{x} = \langle x_1, ..., x_n \mid x_i = e \vee x_i = * \rangle$
$f(\vec{e})$	::=	`ORDER BY ASC`$(\vec{e})$ | `ORDER BY DESC`$(\vec{e})$ | id	

Labeled Commands:

ℓ	$\in$	$\mathbb{L}$	Set of Labels
Q	$\in$	$\mathbb{Q}$	Set of Labeled SQL Statements
Q	::=	SELECT | UPDATE | INSERT | DELETE	
SELECT	::=	$\langle {}^{\ell_5} assign(v_a),\ {}^{\ell_4} f(\vec{e'}),\ {}^{\ell_3} r(\vec{h}(\vec{x})),\ {}^{\ell_2}\phi',\ {}^{\ell_1} g(\vec{e}),\ {}^{\ell_0}\phi \rangle$	
UPDATE	::=	$\langle {}^{\ell'} \vec{v_d} \overset{upd}{=} \vec{e},\ {}^{\ell}\phi \rangle$	
INSERT	::=	$\langle {}^{\ell'} \vec{v_d} \overset{new}{=} \vec{e},\ {}^{\ell} true \rangle$	
DELETE	::=	$\langle {}^{\ell'} del(\vec{v_d}),\ {}^{\ell}\phi \rangle$	
c	$\in$	$\mathbb{C}$	Set of Labeled Commands
c	::=	${}^{\ell} skip \mid {}^{\ell} v_a = e \mid Q \mid c_1; c_2$	
	|	$if\ {}^{\ell} b\ then\ c_1\ else\ c_2\ {}^{\ell'}\ endif$	
	|	$while\ {}^{\ell} b\ do\ c\ {}^{\ell'}\ done$	
P	::=	c^{ℓ}	Program that ends with label ℓ.

Environment and States

Definition (Application Environment)

An application environment ρ_a is a partial function $\rho_a : \mathbb{V}_a \longmapsto \mathfrak{D}_{\mathfrak{V}}$. We denote by $\mathfrak{E}_a$ the set of all application environments.

Definition (Database Environment)

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

Definition (Table Environment)

A table environment ρ_t for a table $t \in DB$ is defined as a function such that for any attribute $a_i \in attr(t)$, $\rho_t(a_i) = \langle \pi_i(l_j) \mid l_j \in t \rangle$.

Environments and States

- The set of states $\mathfrak{E} \triangleq \mathfrak{E}_d \times \mathfrak{E}_a$.
  $\mathfrak{E}_d$= set of all database environments;
  $\mathfrak{E}_a$ = set of all application environments.

- A state $\sigma \in \mathfrak{E}$ is a pair (ρ_d, ρ_a)
  ρ_d= database environment;
  ρ_a = application environment.

- Semantics is described as a partial functions on states which specify how expressions are evaluated and instructions are executed.

Semantics of arithmetic expressions

1. $E[\![c]\!](\rho_d, \rho_a) = c$
2. $E[\![v_a]\!](\rho_d, \rho_a) = \rho_a(v_a)$
3. *let* $\exists t \in dom(\rho_d) : \; v_d = a_i \in attr(t)$ *in*

$$E[\![v_d]\!](\rho_d, \rho_a) = E[\![v_d]\!](\rho_t, \rho_a) = \rho_t(a_i)$$

5. *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*

$$\begin{aligned} & E[\![v_d \; op \; v_a]\!](\rho_d, \rho_a) \\ =& E[\![v_d \; op \; v_a]\!](\rho_t, \rho_a) \\ =& \langle (m \; op \; n) \in D_k \mid m \in \rho_t(a_i) \wedge \rho_a(v_a) = n \wedge a_i \in D_i \wedge v_a \in D_j \rangle \end{aligned}$$

6. *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*

$$\begin{aligned} & E[\![v_{d_1} \; op \; v_{d_2}]\!](\rho_d, \rho_a) \\ =& E[\![v_{d_1} \; op \; v_{d_2}]\!](\rho_t, \rho_a) \\ =& \langle m_r \in D_k \mid m_r = \pi_i(l_r) \; op \; \pi_j(l_r) \; where \; l_r \; is \; the \; r^{th} \; row \; of \; t \rangle \end{aligned}$$

Semantics of boolean expressions

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. $B[\![b_1 \wedge b_2]\!](\rho_d, \rho_a) = B[\![b_1]\!](\rho_d, \rho_a) \wedge B[\![b_2]\!](\rho_d, \rho_a)$

Semantics of Statements: UPDATE

Step 1: Absorbing ϕ:

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

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

Step 2: Update:

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

$let\ \vec{v_d} = \vec{a} \subseteq attr(t)\ and\ \vec{e} = \langle e_1, e_2, ..., e_h\rangle\ and\ E[\![\vec{e}]\!](\rho_t, \rho_a) = \langle \vec{m_i} \mid i = 1, ..., h$

$let\ m_i^j\ be\ the\ j^{th}\ element\ of\ the\ sequence\ \vec{m_i}\ and\ a_i\ be\ the\ i^{th}\ element$
$of\ the\ sequence\ \vec{a}.$

$$t' \triangleq \langle\ l_j[m_i^j/a_i] \mid l_j \in t\ \rangle$$

UPDATE: Illustration with example

Consider the following database table `t`

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

Table `t`

- Consider the following update statement

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

- The abstract syntax is denoted by $\langle \texttt{UPDATE}(\vec{v_d}, \vec{e}),\ \phi \rangle$.

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

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

UPDATE: Illustration with example

The semantics of Q_{upd} is:

$$
\begin{aligned}
& S[\![Q_{upd}]\!](\rho_d, \rho_a) \\
=& S[\![\texttt{UPDATE}(\langle sal\rangle, \langle sal + 100\rangle),\ (age \geqslant 35)]\!](\rho_d, \rho_a) \\
=& S[\![\texttt{UPDATE}(\langle sal\rangle, \langle sal + 100\rangle),\ (age \geqslant 35)]\!](\rho_t, \rho_a) \quad \text{Since, target}(Q_{upd})=\{t\}. \\
=& S[\![\texttt{UPDATE}(\langle sal\rangle, \langle sal + 100\rangle),\ true]\!](\rho_{t\downarrow(age\geqslant 35)}, \rho_a) \qquad \text{Absorbing } \phi. \\
& \qquad \sqcup \\
& S[\![\texttt{UPDATE}(\langle sal\rangle, \langle sal + 100\rangle),\ false]\!](\rho_{t\downarrow\neg(age\geqslant 35)}, \rho_a) \\
=& (\rho_{t\downarrow(age\geqslant 35)} \left[sal \leftarrow E[\![sal + 100]\!](\rho_{t\downarrow(age\geqslant 35)}, \rho_a)\right], \rho_a) \quad \text{Performing update action.} \\
& \qquad \sqcup \\
& (\rho_{t\downarrow\neg(age\geqslant 35)}, \rho_a) \\
=& (\rho_{t\downarrow(age\geqslant 35)} [sal \leftarrow \langle 1600, 2600, 3100\rangle], \rho_a) \sqcup (\rho_{t\downarrow\neg(age\geqslant 35)}, \rho_a) \\
=& (\rho_{t'}, \rho_a) \sqcup (\rho_{t\downarrow\neg(age\geqslant 35)}, \rho_a) \\
=& (\rho_{t'} \sqcup \rho_{t\downarrow\neg(age\geqslant 35)},\ \rho_a \sqcup \rho_a) = (\rho_{t''}, \rho_a)
\end{aligned}
$$

Concrete Semantics of Database Applications

Transition Semantics:

$$\mathscr{T}[\![c]\!] = \{\sigma \xrightarrow{c} \sigma' \mid \sigma' \in S[\![c]\!]\sigma \wedge \sigma, \sigma' \in \Sigma[\![c]\!]\}$$

Fix-point Trace Semantics:

$$\mathbb{T}[\![P]\!](\mathcal{I}_0) = \mathrm{lfp}^{\subseteq}_{\emptyset}\mathcal{F}(\mathcal{I}_0) = \bigcup_{i \leq \omega} \mathcal{F}^i(\mathcal{I}_0)$$

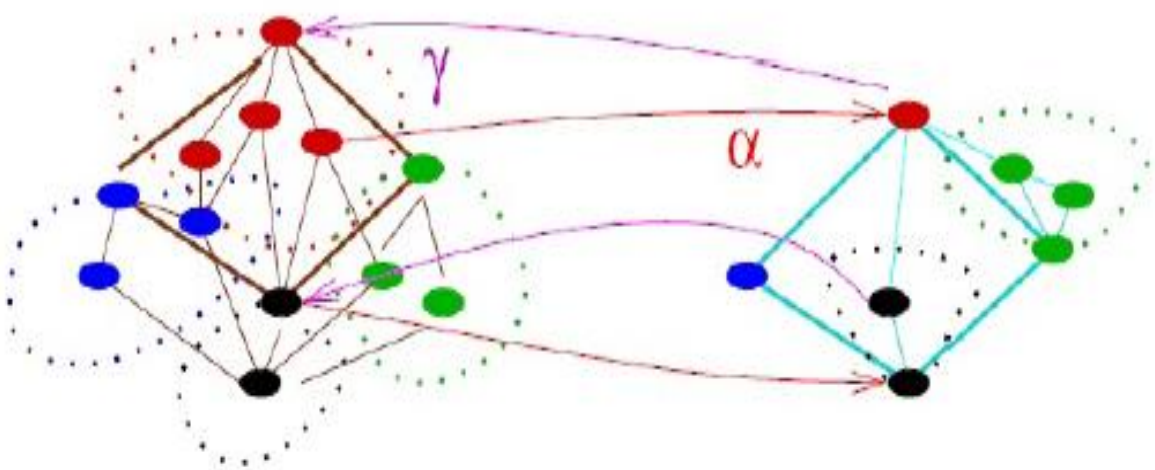
$$\text{where } \mathcal{F}(\mathcal{I}) = \quad \lambda\mathcal{T}.\, \mathcal{I} \cup \left\{\sigma_0 \xrightarrow{c_0} \ldots \xrightarrow{c_{n-1}} \sigma_n \xrightarrow{c_n} \sigma_{n+1} \mid \sigma_0 \xrightarrow{c_0} \ldots \xrightarrow{c_{n-1}} \sigma_n \in \mathcal{T} \wedge \sigma_n \xrightarrow{c_n} \sigma_{n+1} \in \mathscr{T}[\![P]\!]\right\}$$

Abstract Interpretation Theory

Galois Connection

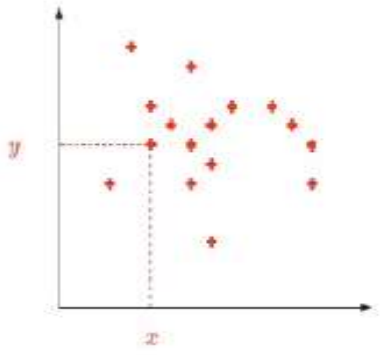
- A Galois Connection $(\langle D, \subseteq \rangle, \alpha, \gamma, \langle D^\sharp, \sqsubseteq \rangle)$ between a concrete poset $\langle D, \subseteq \rangle$ and an abstract poset $\langle D^\sharp, \sqsubseteq \rangle$ arises when:
  - α, γ are monotone: $\forall x, y \in D : x \subseteq y \implies \alpha(x) \sqsubseteq \alpha(y)$,
  - $\gamma \circ \alpha$ is extensive: $\forall x \in D : x \subseteq \gamma(\alpha(x))$,
  - $\alpha \circ \gamma$ is reductive: $\forall x^\sharp \in D^\sharp : \alpha(\gamma(x^\sharp)) \sqsubseteq x^\sharp$.

`That is, iff` $\forall x \in D, x^\sharp \in D^\sharp : \alpha(x) \sqsubseteq x^\sharp \iff x \subseteq \gamma(x^\sharp)$.



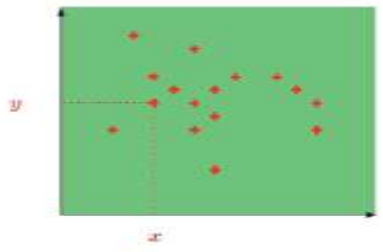
Abstract Interpretation Theory

Abstract domains for approximating an (infinite) set of values of two variables.



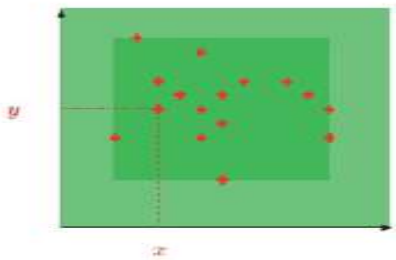
$\{\ldots, \langle 19, 77\rangle, \ldots, \langle 20, 03\rangle, \ldots\}$

The Domain of Signs



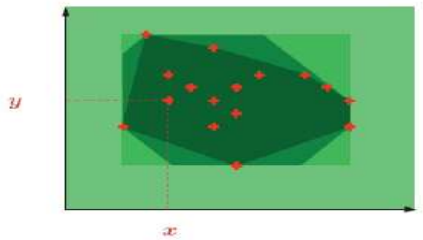
$\begin{cases} x \geq 0 \\ y \geq 0 \end{cases}$

The Domain of Intervals



$\begin{cases} x \in [19,\ 77] \\ y \in [20,\ 03] \end{cases}$

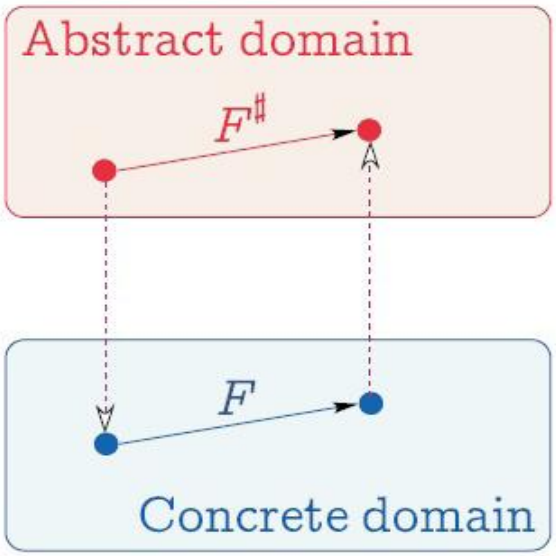
The Domain of Polyhedra

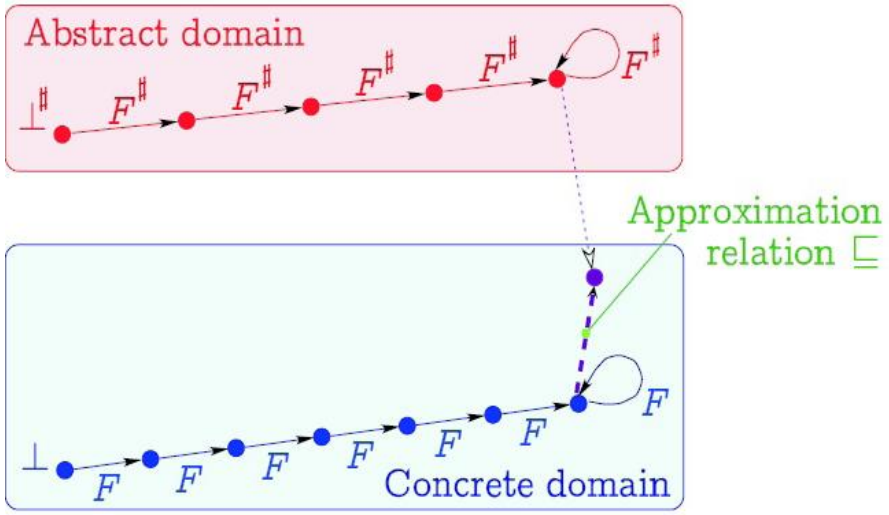


$\begin{cases} 19x + 77y \leq 2004 \\ 20x + 03y \geq 0 \end{cases}$

Abstract Interpretation Theory

Abstract Functions





$\alpha(\mathsf{lfp}\, F) \sqsubseteq \mathsf{lfp}\, F^\sharp$

There is always an optimal abstraction of the concrete function F, defined by

$$F^\sharp = \alpha \circ F \circ \gamma$$

However, for the correctness of the analysis it is sufficient that

$$\forall x^\sharp \in D^\sharp : \ \gamma(F^\sharp(x^\sharp)) \supseteq F(\gamma(x^\sharp))$$

AI of DB Systems: Data and Operations Abstraction

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	Sanu	14	2	3	4000	3

Table : Concrete Table: t_{emp}

$eID^\sharp$	$Name^\sharp$	$Age^\sharp$	$Dno^\sharp$	$Pno^\sharp$	$Sal^\sharp$	$Child-no^\sharp$
1	Matteo	[25,59]	2	1	[1500,2499]	many
2	Alice	[12,24]	1	2	[1500,2499]	Medium
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]	Medium
7	Alberto	[12,24]	3	4	[500,1499]	Few
8	Sanu	[12,24]	2	3	[2500,10000]	Medium

Table : Abstract table: $t^\sharp_{emp}$

$$\alpha_{age}(X) \triangleq \begin{cases} \bot & \text{if } X = \emptyset \\ [5,11] & \text{if } \forall x \in X:\ 5 \le age \le 11 \\ [12,24] & \text{if } \forall x \in X:\ 12 \le age \le 24 \\ [25,59] & \text{if } \forall x \in X:\ 25 \le age \le 59 \\ [60,100] & \text{if } \forall x \in X:\ 60 \le age \le 100 \\ \top & \text{otherwise} \end{cases}$$

$$\alpha_{sal}(X) \triangleq \begin{cases} \bot & \text{if } X = \emptyset \\ [500,1499] & \text{if } \forall x \in X:\ 500 \le x \le 1499 \\ [1500,2499] & \text{if } \forall x \in X:\ 1500 \le x \le 2499 \\ [2500,10000] & \text{if } \forall x \in X:\ 2500 \le x \le 10000 \\ \top & \text{otherwise} \end{cases}$$

$$\alpha_{ChildNo}(X) \triangleq \begin{cases} \bot & \text{if } X = \emptyset \\ \text{Zero} & \text{if } X = \{0\} \\ \text{Few} & \text{if } \forall x \in X:\ 1 \le x \le 2 \\ \text{Medium} & \text{if } \forall x \in X:\ 3 \le x \le 4 \\ \text{many} & \text{if } \forall x \in X:\ 5 \le x \le 10 \\ \top & \text{otherwise} \end{cases}$$

- Define Abstract Operations, *e.g.* abstract aggregate functions ($\texttt{SUM}^\sharp$, $\texttt{AVG}^\sharp$, $\texttt{COUNT}^\sharp$, $\texttt{MAX}^\sharp$, $\texttt{MIN}^\sharp$) and abstract set operations ($\texttt{MINUS}^\sharp$, $\texttt{INTERSECT}^\sharp$, $\texttt{UNION}^\sharp$), so as to maintain soundness.

AI of Database Systems: Example

SELECT $MAX(Sal), AVG(Age), COUNT(*), SUM(Child - no)$
FROM t_{emp}
WHERE $sal > 1600$
GROUP BY (Dno, Pno)
HAVING $MAX(sal) < 4000$

$MAX(Sal)$	$AVG(Age)$	$COUNT(*)$	$SUM(Child - no)$
2000	30	1	4
3000	40	1	5
1900	40	2	3

Table : Resulting table: t_{res} (concrete)

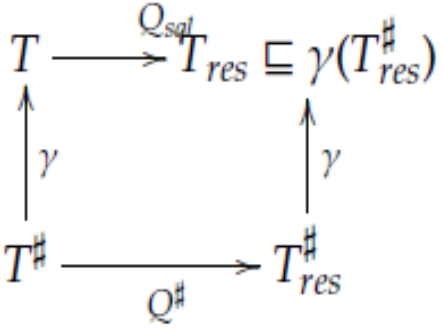
SELECT$^\sharp$ $MAX^\sharp(sal^\sharp)$, $AVG^\sharp(age^\sharp)$, $COUNT^\sharp(*)$, $SUM^\sharp(child - no^\sharp)$
FROM $t^\sharp_{emp}$
WHERE $sal^\sharp >^\sharp [1500, 1499]$
GROUP BY$^\sharp$ $(Dno^\sharp, Pno^\sharp)$
HAVING $MAX^\sharp(sal^\sharp) <^\sharp [2500, 10000]$

$MAX^\sharp(Sal^\sharp)$	$AVG^\sharp(Age^\sharp)$	$COUNT^\sharp(*)$	$SUM^\sharp(Child - no^\sharp)$
[1500,2499]	[25,59]	[0, 1]	many
[2500,10000]	[25,59]	[1, 1]	many
[1500,2499]	[5,100]	[0, 3]	$\top$

Table : Resulting table: $t^\sharp_{res}$ (abstract)

Oh! yes, the above abstraction is sound, *i.e.* $t_{res} \in \gamma(t^\sharp_{res})$

Soundness Condition:



Abstract Transition Semantics: Relational Abs. Dom.

The abstract transition relation: $\mathscr{T}_{sql} : \mathbb{Q} \times \mathfrak{p} \to \wp(\mathfrak{p})$.

UPDATE: $\mathscr{T}_{sql}[\![\langle \texttt{UPDATE}(\vec{v_d}, \vec{e}),\ \phi\rangle]\!]\mathtt{P} = \{\mathtt{P}'_T\ ,\ \mathtt{P}_F\}$

where

$$\mathtt{P}_T = (\mathtt{P} \sqcap \phi).$$
$$\mathtt{P}'_T = \mathscr{T}_{sql}[\![\texttt{UPDATE}(\vec{v_d}, \vec{e})]\!]\mathtt{P}_T = \mathscr{T}_{sql}[\![\vec{v_d} := \vec{e}]\!]\mathtt{P}_T.$$
$$\mathtt{P}_F = (\mathtt{P} \sqcap \neg\phi).$$

DELETE: $\mathscr{T}_{sql}[\![\langle \texttt{DELETE}(\vec{v_d}),\ \phi\rangle]\!]\mathtt{P} = \{(\mathtt{P} \sqcap \neg\phi)\}$

INSERT: $\mathscr{T}_{sql}[\![\langle \texttt{INSERT}(\vec{v_d}, \vec{e}),\ \phi\rangle]\!]\mathtt{P} = \{\mathtt{P} \sqcup \mathtt{P}_{new}\} = \{\mathtt{P}'\}$

where $\mathtt{P}_{new}$ represents a polyhedron corresponding to the new inserted tuple values.

SELECT: The select operation does not modify any information in a polyhedron.

$$\begin{aligned}
&\mathscr{T}_{sql}[\![\langle \texttt{SELECT}\big(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2,\ g(\vec{e})\big),\ \phi_1\rangle]\!]\mathtt{P} \\
&= \mathscr{T}_{sql}[\![\langle \texttt{SELECT}\big(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2,\ g(\vec{e})\big),\ true\rangle]\!]\mathtt{P}_T \ \bigcup \\
&\quad \mathscr{T}_{sql}[\![\langle \texttt{SELECT}\big(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2,\ g(\vec{e})\big),\ false\rangle]\!]\mathtt{P}_F \\
&= \{\mathtt{P}\}
\end{aligned}$$

Challenges in case of HQL

- High-level Object-Oriented Constructs
- Session Methods (Hibernate API)
- Mapping Between HQL variables and Database Attributes
- Persistent Vs. Transient Objects

HQL: Syntax

Set of Classes

$$c \quad \in \quad \texttt{Class}$$

$$c \quad ::= \quad \langle \texttt{init}, \texttt{F}, \texttt{M} \rangle$$
where `init` is the constructor, $\texttt{F} \subseteq \texttt{Var}$ is the set of fields, and `M` is the set of methods.

`Session` methods

$$m_{ses} \quad \in \quad \texttt{M}_{ses}$$

$$m_{ses} \quad ::= \quad \langle \texttt{C}, \phi, \texttt{OP} \rangle$$
where $\texttt{C} \subseteq \texttt{Class}$ and ϕ represents '`WHERE` clause'

$$\texttt{OP} \quad ::= \quad \texttt{SEL}\Big(f(\vec{e'}), r(\vec{h}(\vec{x})), \phi, g(\vec{e})\Big) \mid \texttt{UPD}(\vec{v}, \vec{e}) \mid \texttt{SAVE}(\texttt{obj}) \mid \texttt{DEL}()$$
where ϕ represents '`HAVING`' clause and `obj` denotes an instance of a class.

HQL Programs

$$p \quad \in \quad \mathbb{P}$$

$$p \quad ::= \quad \langle c_{main}, \texttt{L} \rangle$$
where $c_{main} \in \texttt{Class}$ is the main class and $\texttt{L} \subset \texttt{Class}$

HQL: Environments, Stores and States

- `Env` : `Var` $\longrightarrow$ `Loc`.
- `Store` : `Loc` $\longrightarrow$ `Val`.
- $\mathfrak{E}_d$: Set of database environments.
  - Let `DB` $= \{t_i \mid i \in I_x\}$.
  - Database environment: $\rho_d(i) = t_i$ for $i \in I_x$.
  - Table Environment: $\rho_t(a_i) = \langle \pi_i(l_j) \mid l_j \in t \rangle$.

Definition (Interaction States of `Session` Objects)

The set of interaction states of `Session` objects is defined by $\Sigma =$ `Env` $\times$ `Store` $\times \mathfrak{E}_d$. Therefore, an interaction state of a `Session` object is a triplet $\langle e, s, \rho_d \rangle$ where $e \in$ `Env`, $s \in$ `Store` and $\rho_d \in \mathfrak{E}_d$.

HQL: Concrete Semantics

- `Session` Method $\left\langle \{c\}, \mathit{false}, \texttt{SAVE(obj)} \right\rangle$:

$$\mathbf{S}_{hql}[\![\langle \{c\}, \mathit{false}, \texttt{SAVE(obj)} \rangle]\!] =$$
$$\lambda(e, s, \rho_t).\ \text{let } c = \langle \texttt{init}, \texttt{F}, \texttt{M} \rangle \text{ such that } \mathit{map}(\texttt{F}) = \mathit{attr}(t) = \vec{a},$$
$$\text{and let } s(e(\texttt{obj})) = e' \text{ such that } s(e'(\texttt{F})) = \vec{val},\ \text{in}$$
$$\left\{ \langle e, s, \rho_{t'} \rangle \mid \rho_{t'} \in \mathbf{S}_{sql}[\![\left\langle \texttt{INSERT}(\vec{a}, \vec{val}), \mathit{false} \right\rangle]\!](\rho_t) \right\}.$$

- Similarly for other session methods: $\left\langle \texttt{C}, \phi, \texttt{UPD}(\vec{v}, \vec{exp}) \right\rangle$, $\left\langle \{c\}, \phi, \texttt{DEL}() \right\rangle$, $\left\langle \texttt{C}, \phi', \texttt{SEL}\left(f(\vec{exp'}),\ r(\vec{h}(\vec{x})),\ \phi,\ g(\vec{exp})\right) \right\rangle$.
- Abstraction in relational/non-relational abstract domains.

Conclusions and Future Aims

- We addressed the issue of extending the Abstract Interpretation framework to new scenarios, that may be particularly interesting from an application perspective.
- Building an automatic analysis/verification tool of database applications based on our theoretical foundations.
  - Considering Complex, Nested Queries....
  - Various suitable abstract domains....
  - Efficiency, Preciseness...
- Looking for new application areas and moving towards web-based information systems.
- Application of Formal Methods to Cloud Computing, Big-Data Analytics...

References


Raju Halder and Agostino Cortesi. Abstract Program Slicing on Dependence Condition Graph. *Science of Computer Programming, Elsevier Ed.*, 78(9): 1240-1263. Elsevier Ed., 2013.


Raju Halder and Agostino Cortesi. Abstract Interpretation of Database Query Languages. *Computer Languages, Systems and Structures, Elsevier Ed.*, 38(2): 123-157, Elsevier Ed., 2012.


Raju Halder, Angshuman Jana, Agostino Cortesi. Data Leakage Analysis of the Hibernate Query Language on a Propositional Formulae Domain. *LNCS Transactions on Large-Scale Data and Knowledge-Centered Systems, Springer*, Vol 23: 23-44, Springer, 2016.


R. Halder, M. Zanioli and A. Cortesi. Information Leakage Analysis of Database Query Languages. *Proc. of the 29th Symposium on Applied Computing (SAC 2014)*, Korea. ACM Press.


Raju Halder and Agostino Cortesi. Abstract program slicing of database query languages. *Proc. of the 28th Symposium on Applied Computing (SAC 2013)*, Portugal. ACM Press.


A. Cortesi and R. Halder. Abstract Interpretation of Recursive Queries. *Proc. of the 9th Int. Conf. on Distributed Computing and Internet Technologies (ICDCIT '13)*, India. Springer LNCS 7753.


Raju Halder and Agostino Cortesi. Fine Grained Access Control for Relational Databases by Abstract Interpretation. *Software and Data Technologies, Springer CCIS 170, 2013.*


Raju Halder and Agostino Cortesi. Cooperative Query Answering by Abstract Interpretation. In *Proceedings of the SOFSEM 2011*, Springer LNCS 6543.

Thank You !