SemDDA: A Semantics-based Database Dependency Analyzer

Angshuman Jana, Raju Halder
Indian Institute of Technology Patna
{ajana.pcs, halder}@iitp.ac.in

Introduction

• Dependency among program statements has been widely and successfully used in many software-engineering activities, *e.g.* Maintenance, Safety Verification, Optimization, etc [1].

• One most suitable representation of these dependences (data- and control-dependences) is in the form of Dependency Graph.

• Various forms of dependency graph are: Program Dependence Graph (PDG), System Dependence Graph (SDG), Class Dependence Graph (ClDG), Database-Oriented Program Dependence Graph (DOPDG), etc [2].

• The syntactic construction of dependency graph is based on the syntactic presence of variables (either defined or used) in program statements.

Motivations

• Syntactic dependency computation may produce false alarms. Therefore, it may fail to compute an optimal set of dependences.

• For instance, expression "$e = x^2 + 4w \bmod 2 + z$" syntactically depends on w, semantically there is no dependency as the evaluation of "$4w \bmod 2$" is always zero.

• In [3] authors introduced the notion of semantic-data dependency which focuses on the actual values of variable rather than their syntactic presence.

Objective

• We present SemDDA, a novel Semantics-based Database Dependency Analyzer.

• This tool computes semantic-based dependences in database applications.

• For example, consider the following SQL statements Q_1 and Q_2:

Q_1 :UPDATEempSETsal:=sal+100 WHERE sal⩽2000

Q_2 :SELECT AVG(sal) FROMemp WHERE sal⩾3000

• The SQL statement Q_2 is syntactically Dependent on Q_1 but there exist no semantics-based dependence between Q_1 and Q_2.

Syntax-based DOPDG

• The DOPDG [2] is an extension of PDG with two additional dependences.

(i) Program-Database Dependences (PD-Dependences).

(ii) Database-Database Dependences (DD-dependences).

Semantic-based Dependency

• The SQL statement $Q = \langle A, \phi \rangle$ where A represents an action-part and ϕ represents a conditional-part.

• Let $\rho_{db} \in \Sigma_{db}$ be a database state and $\mathbf{S} : Q \times \Sigma_{db} \rightarrow \Sigma_{db}$ be a semantic function.

• Functions $\mathfrak{A}_{use}$, $\mathfrak{A}_{def}$ compute used- and defined-part of target table t by Q as bellow:

$$\mathfrak{A}_{use}(Q,t) = \rho_{t\downarrow\phi}(\vec{x}) \cup \rho_{t\downarrow\phi}(\vec{y}),$$
$$\mathfrak{A}_{def}(Q,t) = \Delta(\rho_{t'}(\vec{z}), \rho_t(\vec{z}))$$

where $\vec{x}$ = USE(A) and $\vec{y}$ = USE(ϕ) are a list of *used*-variable in A and ϕ. $\vec{z}$ = DEF(Q) is a list of *defined*-variable in Q and $(t \downarrow \phi)$ denotes the set of tuples in t for which ϕ holds "$true$".

• Formally, the SQL statement $Q_2 = \langle A_2, \phi_2 \rangle$ with $target(Q_2) = t'$ is DD-Dependent on another SQL statement Q_1 for Υ ($Q_1 \xrightarrow{\Upsilon} Q_2$) iff $Q_1 \in \{Q_{upd}, Q_{ins}, Q_{del}\}$ and the overlapping-part $\Upsilon = \mathfrak{A}_{use}(Q_2, t') \cap \mathfrak{A}_{def}(Q_1, t) \neq \emptyset$.

Abstract Interpretation

• Abstract Interpretation [4] is a semantics-based static analysis framework.

• It provides a sound approximation of program semantics focusing on a particular property.

Abstract Semantics

• Let $\mathtt{P} \in \mathfrak{p}$ be a polyhedra and $\mathtt{P}_T$, $\mathtt{P}_F$ be the polyhedral form of the *true*- and *false*-part of the database. The abstract transition semantics is defined as $\bar{\mathbf{S}}$: $\mathbb{C} \times \mathfrak{p} \rightarrow \wp(\mathfrak{p})$ where $\mathbb{C}$ is the set of database statements and $\mathfrak{p}$ is the set of all polyhedra. The computation of $\mathfrak{A}_{use}$ and $\mathfrak{A}_{def}$ are defined *w.r.t.* abstract semantics as below:

Assignment: $\bar{\mathbf{S}}[\![x_j := e]\!](\mathtt{P}) = \{\mathtt{P}'\}$

Test: $\bar{\mathbf{S}}[\![\beta]\!]\mathtt{P} = \{\mathtt{P}_T,\ \mathtt{P}_F\}$ where $\mathtt{P}_T = (\mathtt{P} \sqcap \beta) \wedge P_F = (\mathtt{P} \sqcap \neg\beta)$

Update: $\bar{\mathbf{S}}[\![\mathtt{UPDATE}(\vec{v}_d, \vec{e}),\ \phi\rangle]\!]\mathtt{P} = \bar{\mathbf{S}}[\![\mathtt{UPDATE}(\vec{v}_d, \vec{e}),\ true\rangle]\!]\mathtt{P}_T \cup \bar{\mathbf{S}}[\![\mathtt{UPDATE}(\vec{v}_d, \vec{e}),\ false\rangle]\!]\mathtt{P}_F = \{\mathtt{P}'_T\ ,\ \mathtt{P}_F\}$
$\mathfrak{A}_{use}(Q_{upd},\ t) = \langle \mathtt{P}_T \rangle$ and $\mathfrak{A}_{def}(Q_{upd},\ t) = \langle \mathtt{P}_T,\ \mathtt{P}'_T \rangle$

Delete: $\bar{\mathbf{S}}[\![\mathtt{DELETE}(\vec{v}_d), \phi\rangle]\!]\mathtt{P} = \bar{\mathbf{S}}[\![\mathtt{DELETE}(\vec{v}_d), true\rangle]\!]\mathtt{P}_T = \{\mathtt{P}'\}$
$\mathfrak{A}_{use}(Q_{del},\ t) = \langle \mathtt{P}_T \rangle$ and $\mathfrak{A}_{def}(Q_{del},\ t) = \langle \mathtt{P}_T, \emptyset \rangle$

Insert: $\bar{\mathbf{S}}[\![\mathtt{INSERT}(\vec{v}_d, \vec{e}), \phi\rangle]\!]\mathtt{P} = \bar{\mathbf{S}}[\![\mathtt{INSERT}(\vec{v}_d, \vec{e}), true\rangle]\!]\mathtt{P} = \{\mathtt{P} \sqcup \mathtt{P}_{new}\}$ where $\mathtt{P}_{new}$ is the polyhedron represented by the inserted tuple values.
$\mathfrak{A}_{use}(Q_{ins},\ t) = \langle \emptyset \rangle$ and $\mathfrak{A}_{def}(Q_{ins},\ t) = \langle \emptyset, \mathtt{P}_{new} \rangle$

Select: $\bar{\mathbf{S}}[\![\langle \mathtt{SELECT}\big(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2,\ g(\vec{e})\big),\ \phi_1\rangle]\!]\mathtt{P}$
$= \bar{\mathbf{S}}[\![\langle \mathtt{SELECT}\big(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2,\ g(\vec{e})\big), true\rangle]\!]\mathtt{P}_T \bigsqcup \bar{\mathbf{S}}[\![\langle \mathtt{SELECT}\big(v_a,\ f(\vec{e'}),\ r(\vec{h}(\vec{x})),\ \phi_2,\ g(\vec{e})\big),\ false\rangle]\!]\mathtt{P}_F$
$= \{\mathtt{P}\}$ The polyhedron is unchanged .
$\mathfrak{A}_{use}(Q_{sel},\ t) = \langle \mathtt{P}_T \rangle$ and $\mathfrak{A}_{def}(Q_{sel},\ t) = \langle \emptyset, \emptyset \rangle$

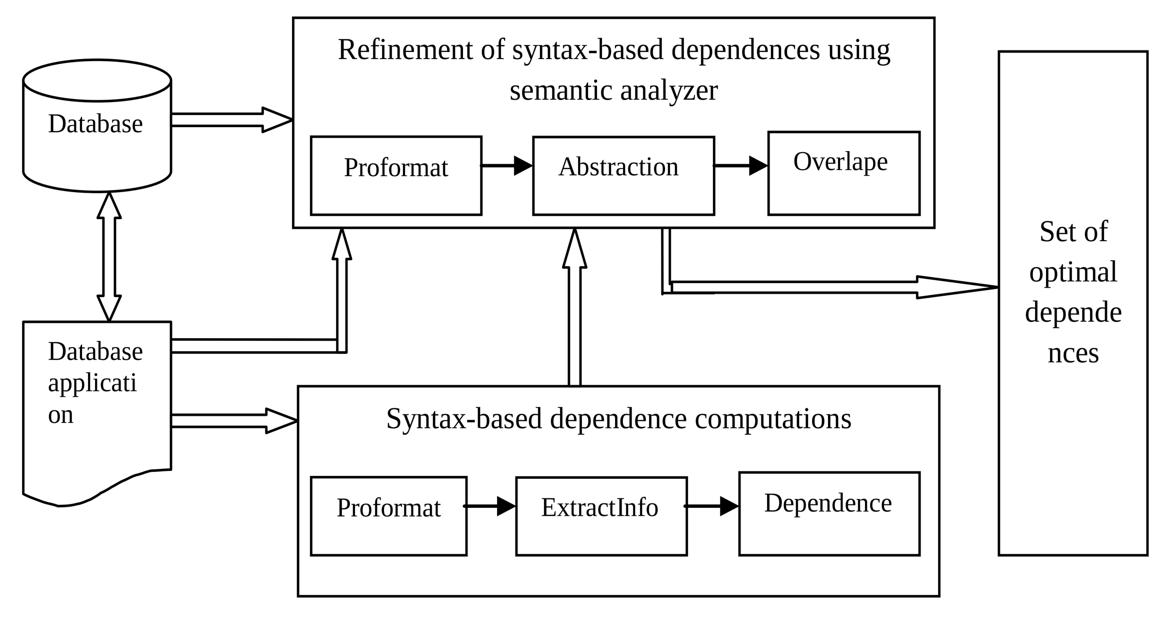
Dependency Computations

• The SQL statements $Q1$ and $Q2$ are semantically independent when $\Upsilon = \mathfrak{A}_{def}(Q_1, t) \cap \mathfrak{A}_{use}(Q_2, t') = \emptyset$ iff

$$\mathtt{P}_T^{Q_1} \sqcap \mathtt{P}_T^{Q_2} = \emptyset \wedge \mathtt{P}_T'^{Q_1} \sqcap \mathtt{P}_T^{Q_2} = \emptyset$$

where $\mathtt{P}_T^{Q1}$ and $\mathtt{P}_T'^{Q1}$ denote the *defined*-part of $Q1$ and $\mathtt{P}_T^{Q2}$ denotes the *used*-part of $Q2$.

Tool Architecture



Applications

• Program Slicing.
• Information Flow Security Analysis.
• Data provenance.
• Concurrent System modeling.

Future Plan

• Currently we are extending our tool by defining abstract semantic of database applications in non-relational abstract domain interval and relational abstract domain octagon.

• We will also perform experiment on benchmark codes using our tool.

References

[1] A. Podgurski and L. A. Clarke, "A formal model of program dependences and its implications for software testing, debugging, and maintenance," IEEE Trans. on SE, 1990
[2] D. Willmor, S. M. Embury, and J. Shao, "Program slicing in the presence of a database state," in Proc. of IEEE ICSM, 2004.
[3] I. Mastroeni and D. Zanardini, "Data dependencies and program slicing: from syntax to abstract semantics," in Proc. of Partial evaluation & semantics-based prog. manipulation, 2008.
[4] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Proc. of the POPL'77, ACM Press.

The snapshot of the tool SemDDA