**OBJECTIVE**

- 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 (CDG), 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 ”$e = x^2 + 4w \mod 2 + z$” syntactically depends on $w$, semantically there is no dependency as the evaluation of ”$4w \mod 2$” 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.

**SYNTAX-BASED DOPDG**

- The DOPDG [2] is an extension of PDG with two additional dependences.
  1. Program-Database Dependences (PD-dependences).
  2. Database-Database Dependences (DD-dependences).

**INTRODUCTION**

- The SQL statement $Q = (A, \phi)$ where $A$ represents an action-part and $\phi$ represents a conditional-part.
- Let $\rho_{db} \in \Sigma_{db}$ be a database state and $S : Q \times \Sigma_{db} \rightarrow \Sigma_{db}$ be a semantic function.
- Functions $\mathcal{A}_{use}$, $\mathcal{A}_{def}$ compute used- and defined-part of target table $t$ by $Q$ as below where $\mathcal{A}_{use}(Q,t,i) = \rho_{db}(i) \cup \rho_{def}(i)$
  $$\mathcal{A}_{use}(Q,t,i) = \rho_{db}(i) \cup \rho_{def}(i)$$
- Functions $\mathcal{A}_{def}$, $\mathcal{A}_{use}$ compute used- and defined-part of target table $t$ by $Q$ as below where $\mathcal{A}_{use}(Q,t,i) = \rho_{db}(i) \cup \rho_{def}(i)$
  $$\mathcal{A}_{use}(Q,t,i) = \rho_{db}(i) \cup \rho_{def}(i)$$
- Formally, the SQL statement $Q_2 = (A_2, \phi_2)$ with target$(Q_2) = \prime$ is DD-Dependent on another SQL statement $Q_1$, for $T(L(Q_2) \supseteq L(Q_2))$ if $Q_2 \in \{Q_{ins}, Q_{del}, Q_{dec} \}$ and the overlapping-part $T(\mathcal{A}_{use}(Q_2,t') \cap \mathcal{A}_{def}(Q_1,t)) \neq \emptyset$.

**DEPENDENCY COMPUTATIONS**

- The SQL statements $Q_1$ and $Q_2$ are semantically independent when $T = \mathcal{A}_{use}(Q_1,t) \cap \mathcal{A}_{use}(Q_2,t') = \emptyset$.
  $$P_{Q_1} \cap P_{Q_2} = \emptyset \land P_{Q_1} \cap P_{Q_2} = \emptyset$$
  where $P_{Q_1}$ and $P_{Q_2}$ denote the defined-part of $Q_1$ and $P_{Q_2}$ denotes the used-part of $Q_2$.

**SEMANTIC-BASED DEPENDENCY**

- The SQL statement $Q = (A, \phi)$ where $A$ represents an action-part and $\phi$ represents a conditional-part.
- Let $\rho_{db} \in \Sigma_{db}$ be a database state and $S : Q \times \Sigma_{db} \rightarrow \Sigma_{db}$ be a semantic function.
- Functions $\mathcal{A}_{use}$, $\mathcal{A}_{def}$ compute used- and defined-part of target table $t$ by $Q$ as below where $\mathcal{A}_{use}(Q,t,i) = \rho_{db}(i) \cup \rho_{def}(i)$
  $$\mathcal{A}_{use}(Q,t,i) = \rho_{db}(i) \cup \rho_{def}(i)$$
- Functions $\mathcal{A}_{def}$, $\mathcal{A}_{use}$ compute used- and defined-part of target table $t$ by $Q$ as below where $\mathcal{A}_{use}(Q,t,i) = \rho_{db}(i) \cup \rho_{def}(i)$
  $$\mathcal{A}_{use}(Q,t,i) = \rho_{db}(i) \cup \rho_{def}(i)$$
- Formally, the SQL statement $Q_2 = (A_2, \phi_2)$ with target$(Q_2) = \prime$ is DD-Dependent on another SQL statement $Q_1$, for $T(L(Q_2) \supseteq L(Q_2))$ if $Q_2 \in \{Q_{ins}, Q_{del}, Q_{dec} \}$ and the overlapping-part $T(\mathcal{A}_{use}(Q_2,t') \cap \mathcal{A}_{def}(Q_1,t)) \neq \emptyset$.

**APPLICATIONS**

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

**THE SNAPSHOT OF THE TOOL SemDDA**

**REFERENCES**


**TOOL ARCHITECTURE**

**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 $P \neq P$ be a polyhedra and $P_T$, $P_P$ be the polyhedral form of the true- and false-part of the database. The abstract transition semantics is defined as $S : C \times P \rightarrow P(P)$ where $C$ is the set of database statements and $P$ is the set of all polyhedra. The computation of $\mathcal{A}_{use}$ and $\mathcal{A}_{def}$ are defined w.r.t. abstract semantics as below:
- Insert: $S[\text{INSERT}(Q_2, \alpha), \rho] = S[\text{INSERT}(Q_2, \alpha), \text{true}] = \{t \cup \alpha\}$ where $t$, $\alpha$ is the polyhedron represented by the inserted tuple values.
- $\mathcal{A}_{use}(Q_{ins}, t) = (\emptyset)$ and $\mathcal{A}_{def}(Q_{ins}, t) = (\emptyset, \emptyset)$.
- Delete: $S[\text{DELETE}(Q_2, \alpha), \rho] = S[\text{DELETE}(Q_2, \alpha), \text{true}] = \{t \setminus \alpha\}$ where $\mathcal{A}_{use}(Q_{del}, t) = (\emptyset)$ and $\mathcal{A}_{def}(Q_{del}, t) = (\emptyset, \emptyset)$.
- Update: $S[\text{UPDATE}(Q_2, \alpha), \rho] = S[\text{UPDATE}(Q_2, \alpha), \text{true}] = \{t \cup \alpha\}$ where $\mathcal{A}_{use}(Q_{upd}, t) = (\emptyset)$ and $\mathcal{A}_{def}(Q_{upd}, t) = (\emptyset, \emptyset)$.
- Test: $S[\text{TEST}(Q_2, \alpha), \rho] = S[\text{TEST}(Q_2, \alpha), \text{true}] = \{t \cup \alpha\}$ where $\mathcal{A}_{use}(Q_{test}, t) = (\emptyset)$ and $\mathcal{A}_{def}(Q_{test}, t) = (\emptyset, \emptyset)$.