A Test Adequacy Infrastructure with Database Interaction Awareness

Gregory M. Kapfhammer
Department of Computer Science
Allegheny College

(in collaboration with Mary Lou Soffa)

Motivation

The Risks Digest, Volume 22, Issue 64, 2003

Jeppesen reports airspace boundary problems

About 350 airspace boundaries contained in Jeppesen NavData are incorrect, the FAA has warned. The error occurred at Jeppesen after a software upgrade when information was pulled from a database containing 20,000 airspace boundaries worldwide for the March NavData update, which takes effect March 20.

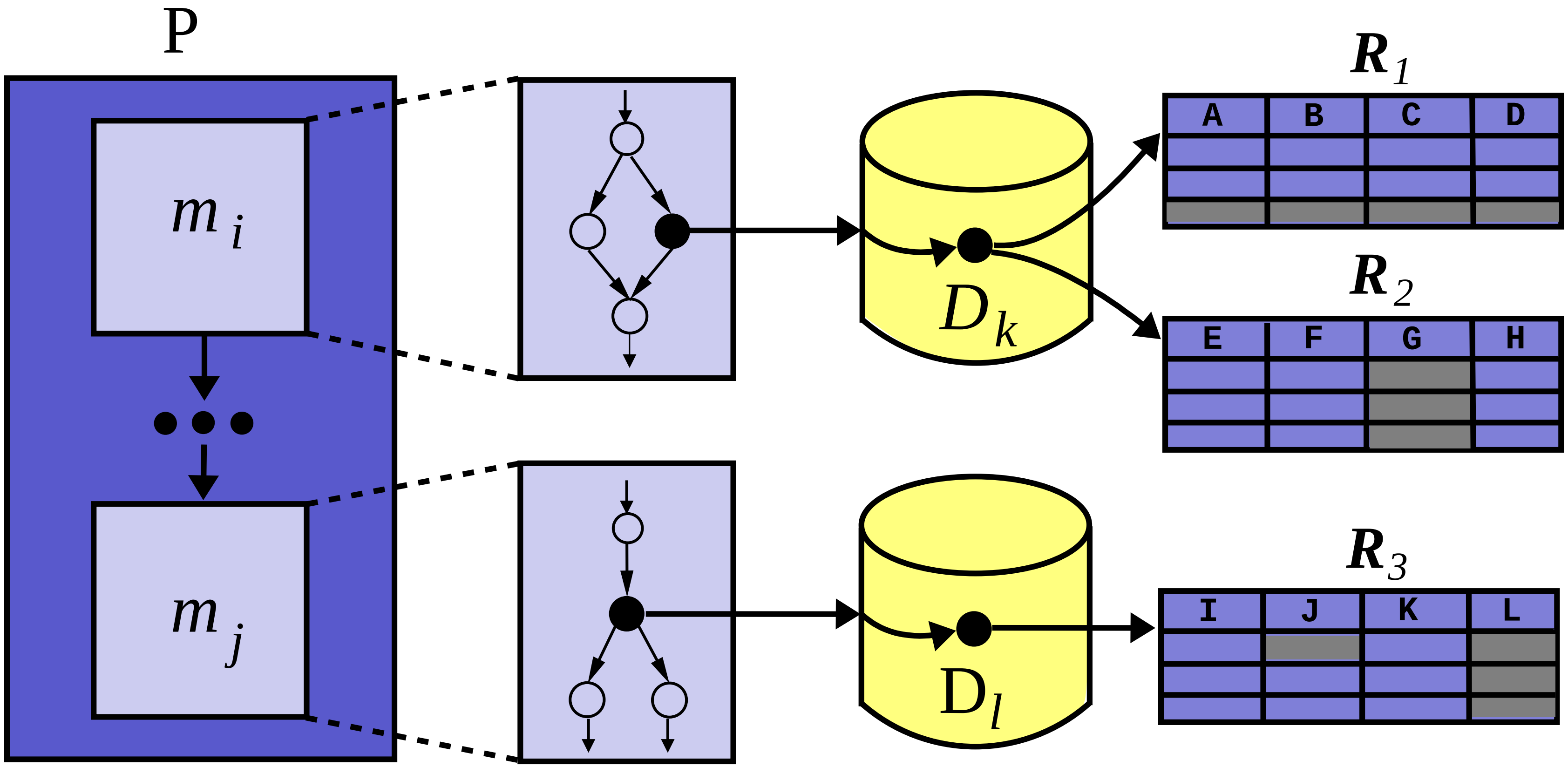
Looking Ahead

- Test adequacy infrastructure that can find faults and establish confidence in the correctness of a database-centric application
  - Model of database interaction faults
  - Unifed application representation
  - Family of test adequacy criteria
- Experiments with real applications that measure the number of test requirements and the time and space overheads incurred by enumeration
  - Foundation for a comprehensive methodology for testing database-centric applications

Testing Challenges

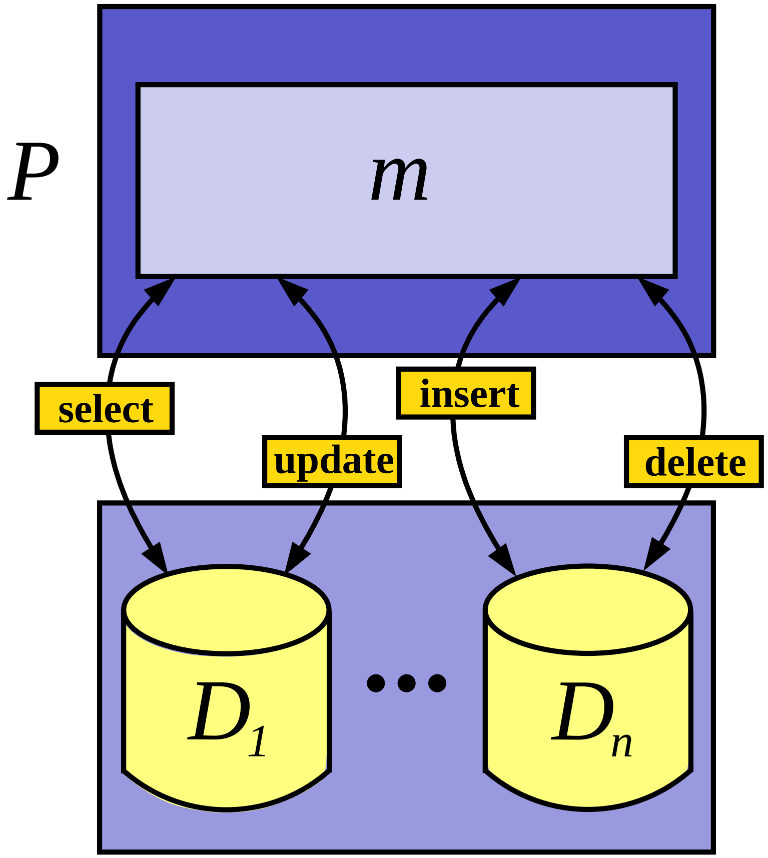
- Must consider the environment in which software applications actually execute
- Should test a program and its interaction with a database
- Database-centric application's state space is well-structured, but essentially infinite (Chays et al.)
- Need to show program does not violate database integrity, where *integrity* = *consistency* + *validity* (Motro)
- Must locate program and database coupling points that vary in granularity

Database-Centric Applications



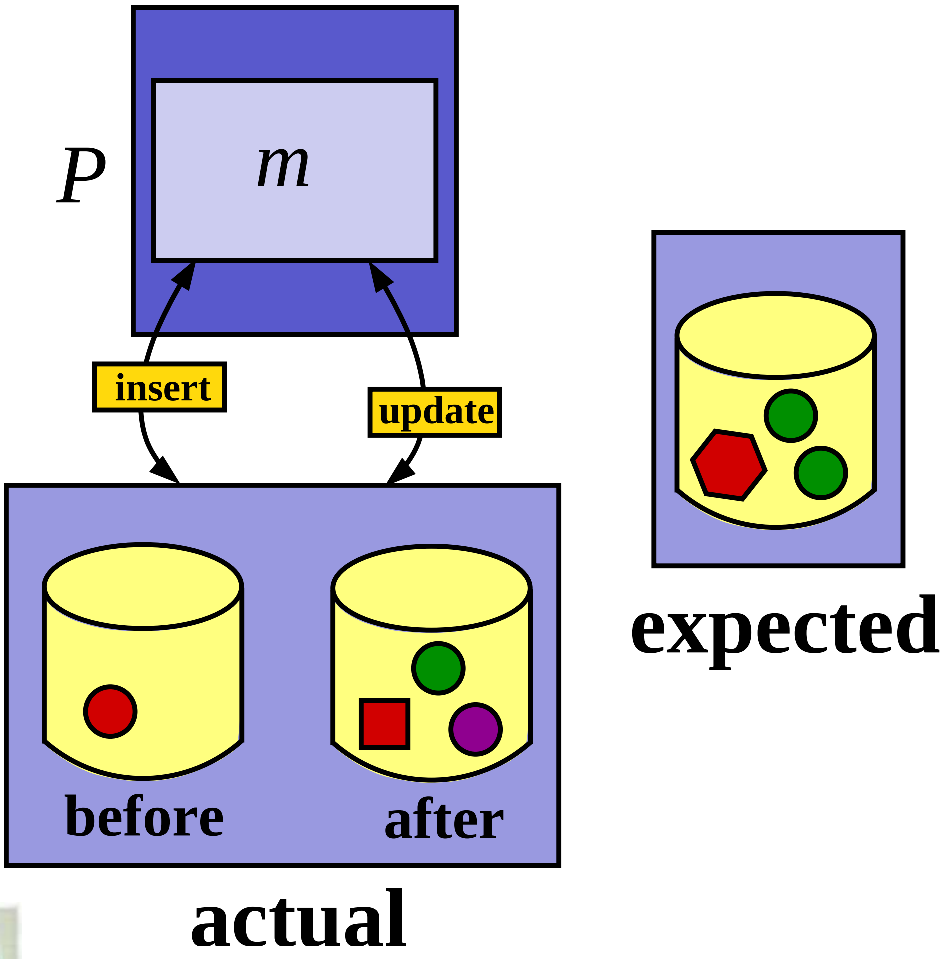
- Program P interacts with two relational databases D_k and D_l

Database Interactions



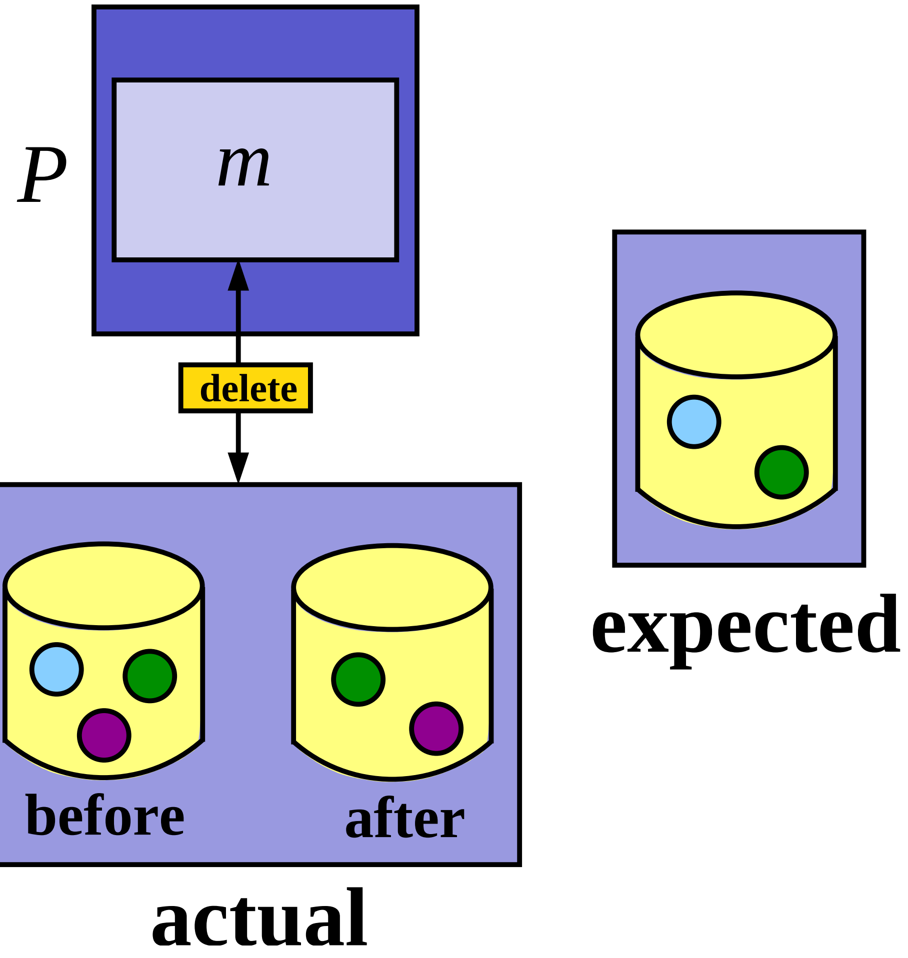
- Program P can view and/or modify the state of the database

Database Interaction Faults: (1-v)



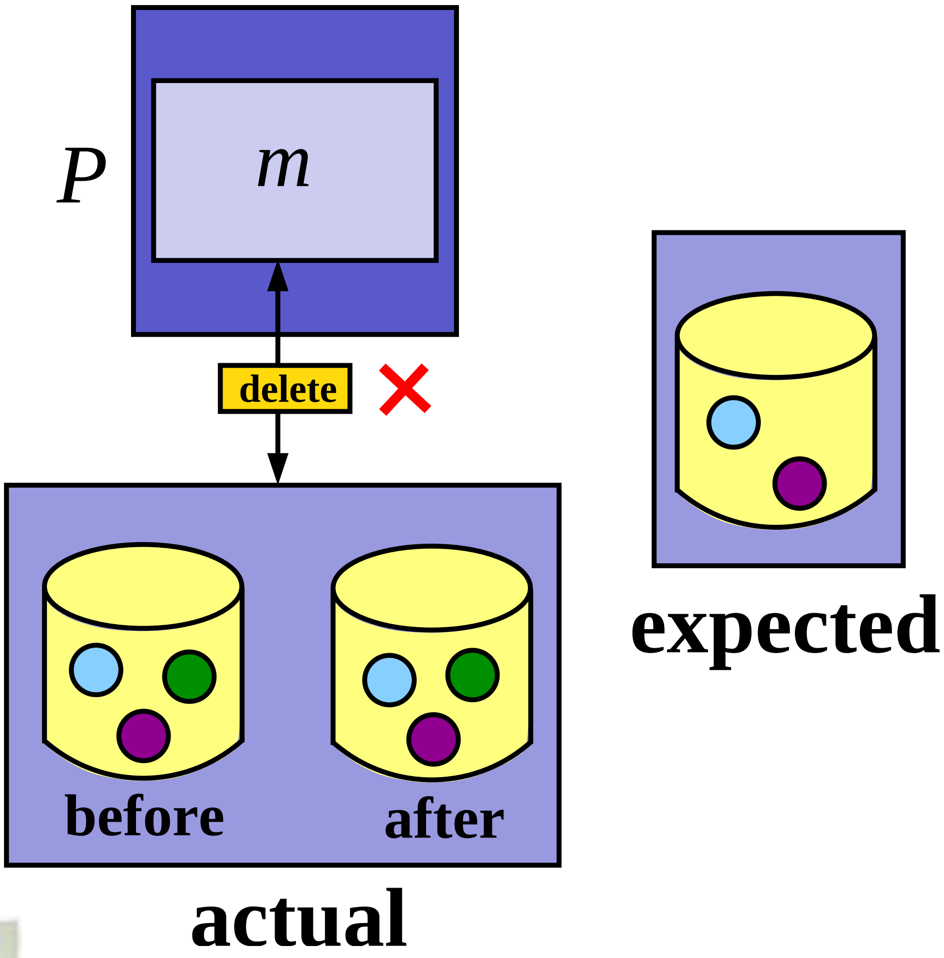
- P uses **update** or **insert** to incorrectly modify items within database
- Commission fault that violates database validity
- Structural adequacy criteria can support fault isolation

Database Interaction Faults: (1-c)



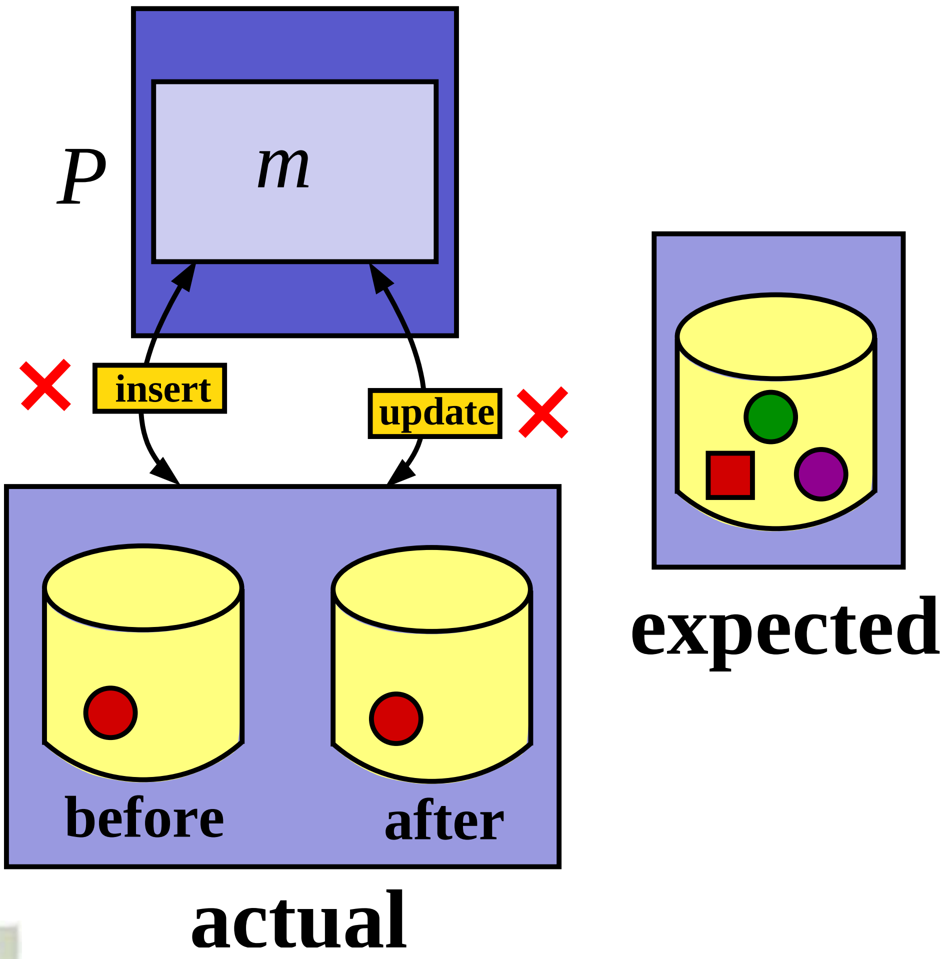
- P uses **delete** to remove incorrect items from database
- Commission fault that violates database completeness
- Structural adequacy criteria can support fault isolation

Database Interaction Faults: (2-v)



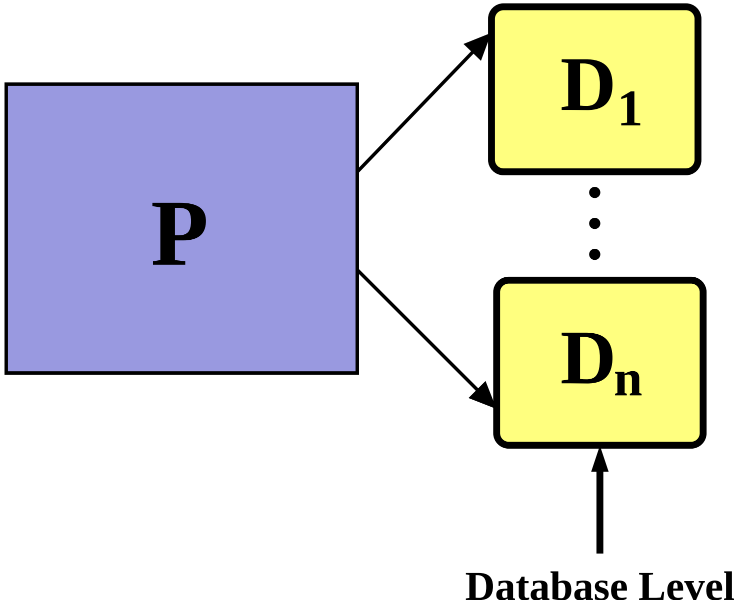
- P does not submit **delete** to remove items from database
- Commission or omission fault that violates database validity
- Structural adequacy criteria cannot easily support omission fault isolation

Database Interaction Faults: (2-c)



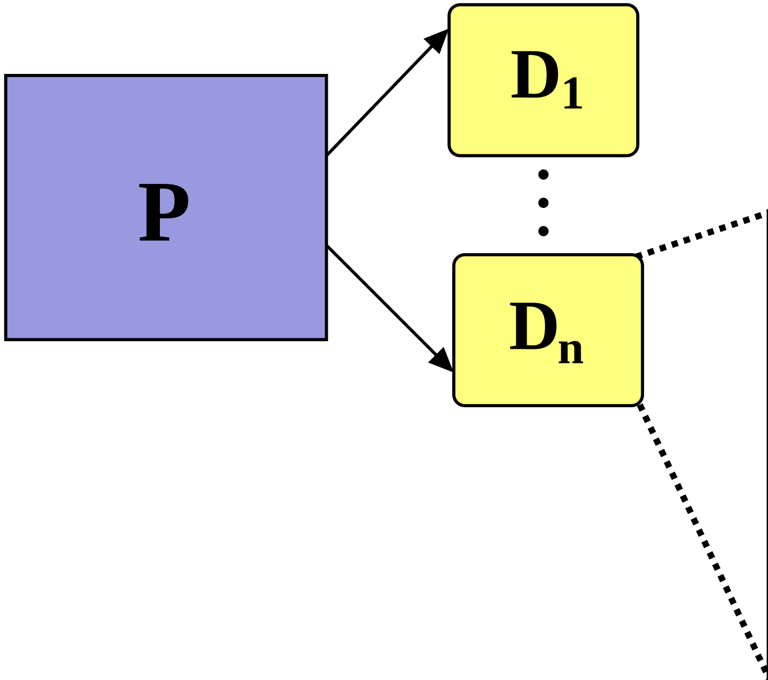
- P does not submit **update** or **insert** to database
- Commission or omission fault that violates database completeness
- Structural adequacy criteria cannot easily support omission fault isolation

Database Interaction Levels



- A program can interact with a relational database at different levels of granularity

Database Interaction Levels

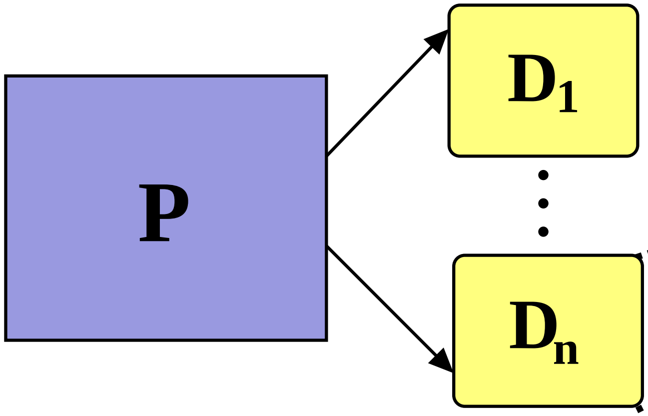


UserInfo

card_number	pin_number	user_name	acct_lock
1	32142	Brian Zorman	0
2	41601	Robert Roos	0
3	45322	Marcus Bittman	0
4	56471	Geoffrey Arnold	0

Relation Level

Database Interaction Levels

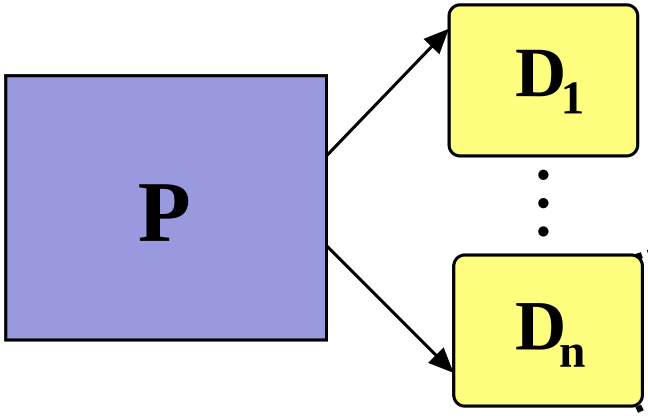


UserInfo

card_number	pin_number	user_name	acct_lock
1	32142	Brian Zorman	0
2	41601	Robert Roos	0
3	45322	Marcus Bittman	0
4	56471	Geoffrey Arnold	0

Record Level

Database Interaction Levels

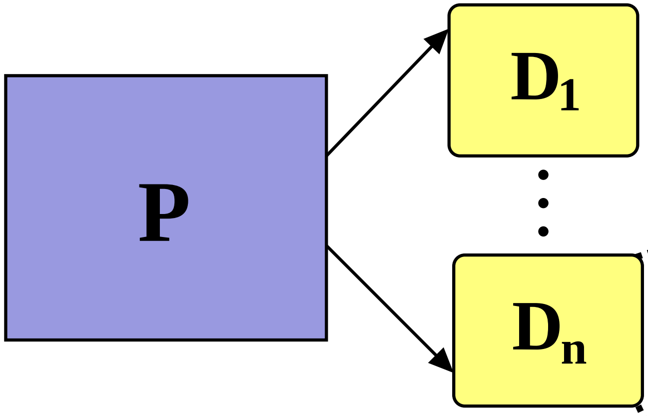


UserInfo

card_number	pin_number	user_name	acct_lock
1	32142	Brian Zorman	0
2	41601	Robert Roos	0
3	45322	Marcus Bittman	0
4	56471	Geoffrey Arnold	0

Attribute Level

Database Interaction Levels



UserInfo

card_number	pin_number	user_name	acct_lock
1	32142	Brian Zorman	0
2	41601	Robert Roos	0
3	45322	**Marcus Bittman**	0
4	56471	Geoffrey Arnold	0

Attribute Value Level

Database Interaction Points: DML

select $A_1, A_2, \ldots, A_q$
from $r_1, r_2, \ldots, r_m$
where Q

delete from r
where Q

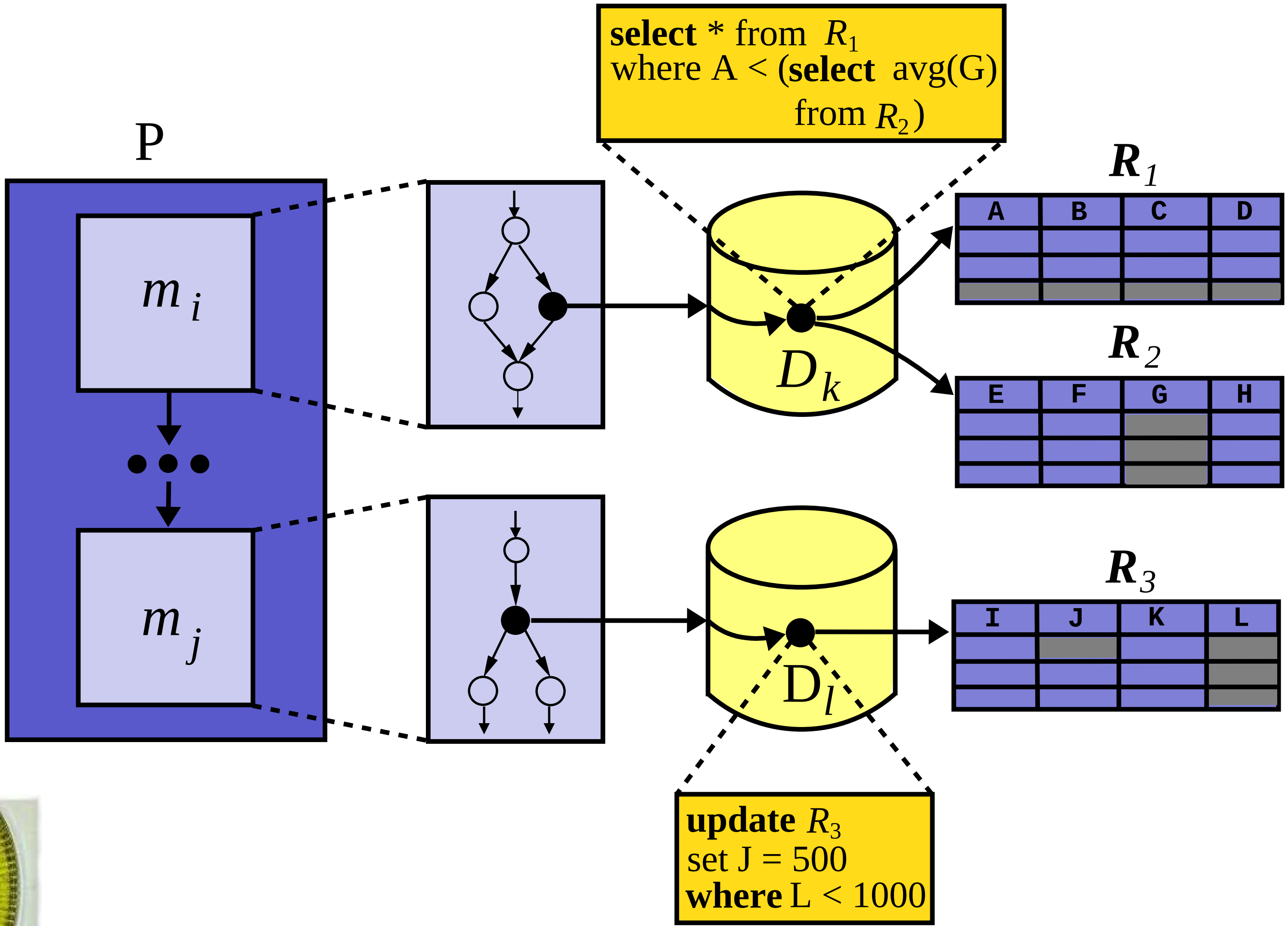
insert into $r(A_1, A_2, \ldots, A_q)$
values$(v_1, v_2, \ldots, v_q)$

update r
set $A_l = F(A_l)$
where Q

Analyzing Database Interaction Points

- Database interaction point $I_r \in I$ corresponds to the execution of a SQL DML statement
- Consider the relevant portions of SQL that are parsed by HSQLDB ($\mathtt{http://hsqldb.sf.net}$)
- Interaction points are normally encoded within Java programs as dynamically constructed `String`s
- **select** uses D_k, **delete** defines D_k, **insert** defines D_k, **update** defines and/or uses D_k

Refined Database-Centric Application



Test Adequacy Concepts

- P violates a database D_k's validity when it:
  - **(1-v)** inserts entities into D_k that do not reflect real world
- P violates a database D_k's completeness when it:
  - **(1-c)** deletes entities from D_k that still reflect real world
- In order to verify **(1-v)** and **(1-c)**, T must cause P to define and then use entities within $D_1, \ldots, D_n$!

Data Flow Information

- Interaction point:
  “UPDATE UserInfo SET acct_lock = 1 WHERE card_number =” + card_number + “;”;
  - Database Level: *define(BankDB)*
  - Attribute Level: *define(acct_lock)* and *use(card_number)*
- Data fbw information varies with respect to the granularity of the database interaction

Database Entities

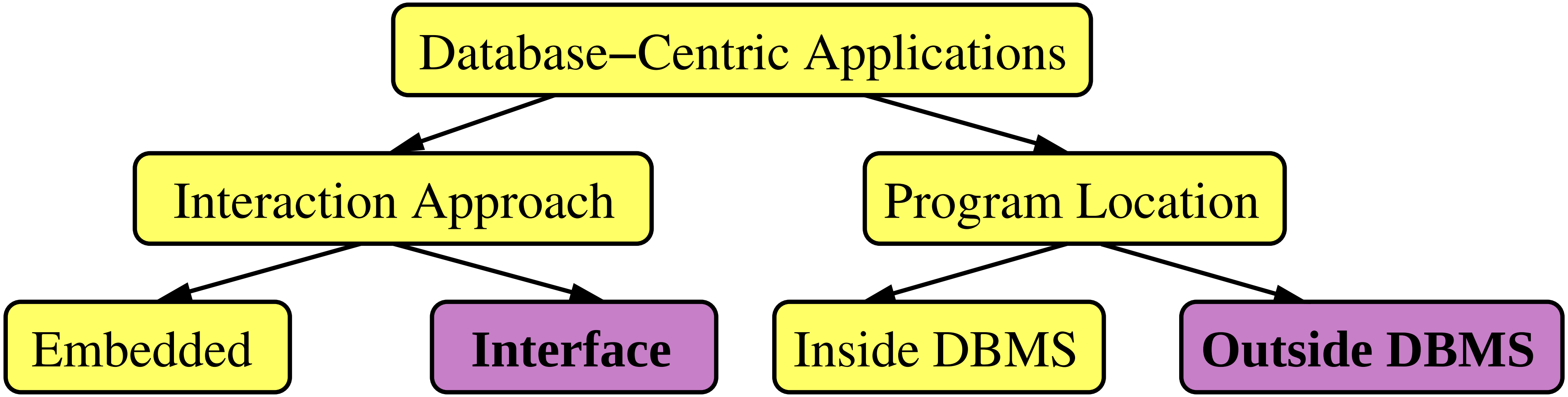
UserInfo

card_number	pin_number	user_name	acct_lock
1	32142	Brian Zorman	0
2	41601	Robert Roos	0
3	45322	Marcus Bittman	0
4	56471	Geoffrey Arnold	0

$$A_v(I_r) = \{ \boxed{1}, \boxed{32142}, \ldots, \boxed{\text{Geoffrey Arnold}}, \boxed{0} \}$$

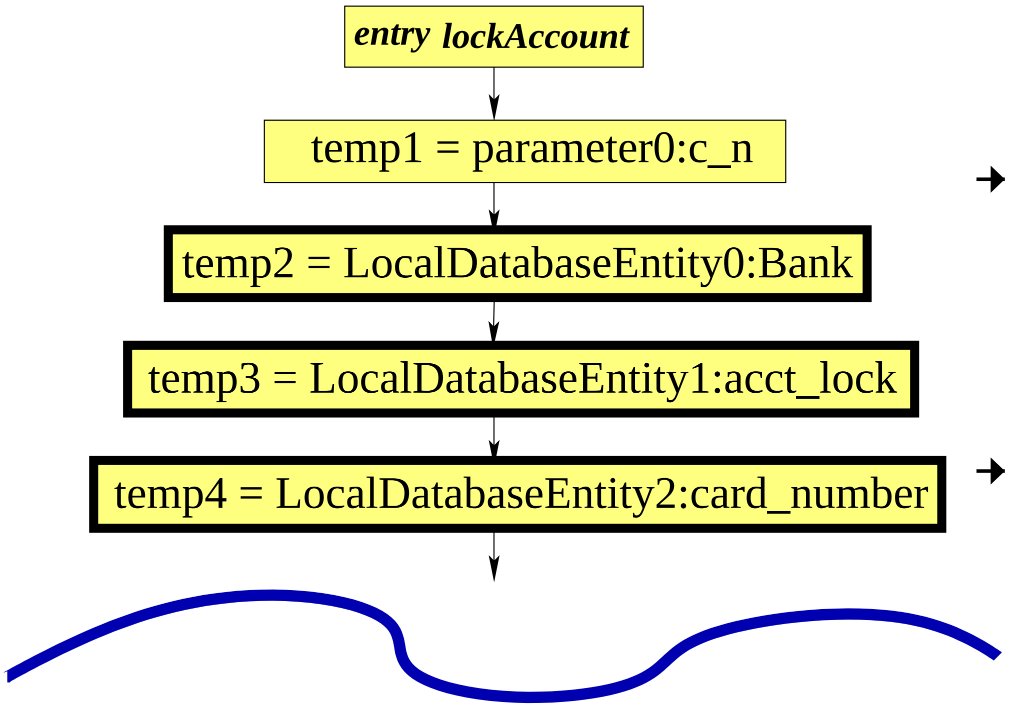
- Enumerate database entities at the attribute value level

Application Types



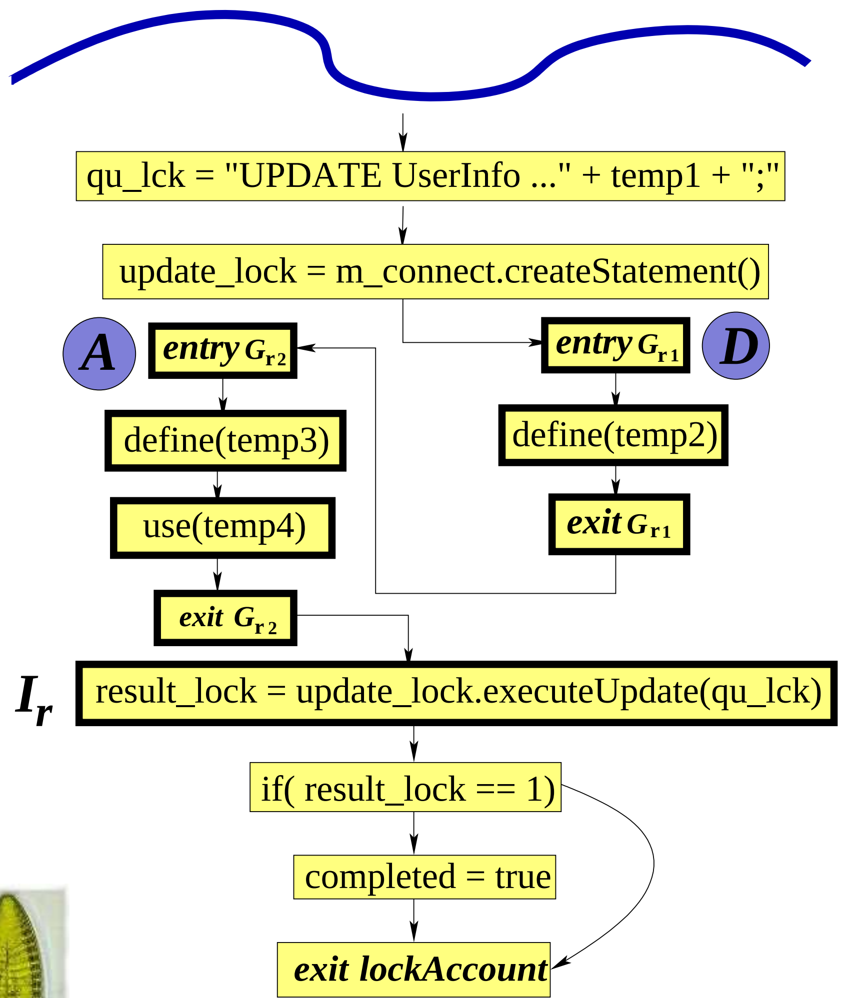
- Testing methodology relevant to all types of applications
- Current tool support focuses on Interface-Outside applications
- **Example:** Java application that submits SQL `Strings` to HSQLDB relational database using JDBC drivers

The DICFG: A Unified Representation



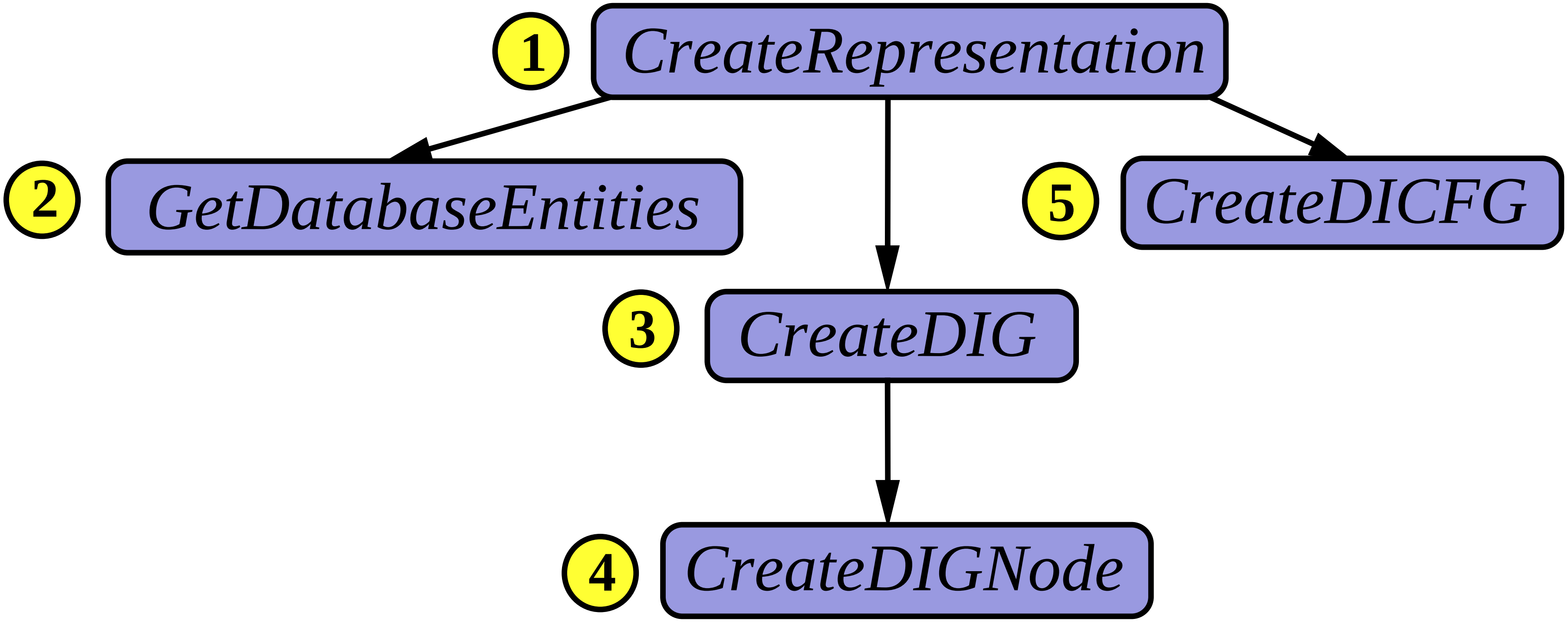
- "Database-enhanced" CFG for `lockAccount`
- Automatically constructed with tool support
- Define temporaries to represent the program's interaction at the levels of database and attribute

The DICFG: A Unified Representation



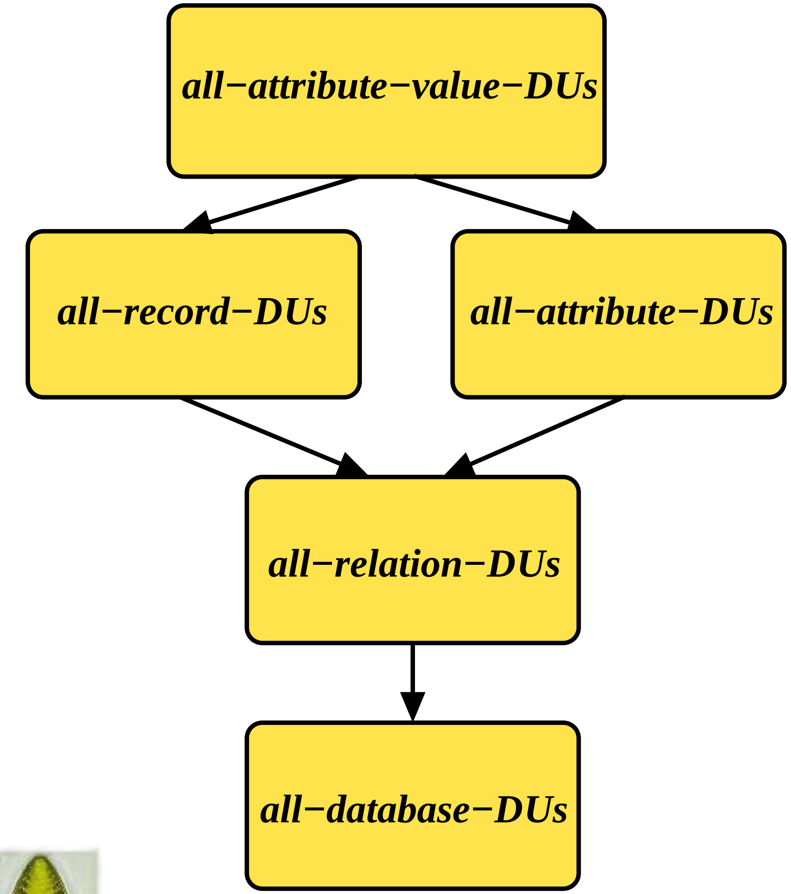
- Database interaction graphs (DIGs) are placed before interaction point I_r
- Multiple DIGs can be integrated into a single CFG
- `String` at I_r is determined in a control-fbw sensitive fashion using enhanced BRICS JSA

DICFG Construction Algorithms



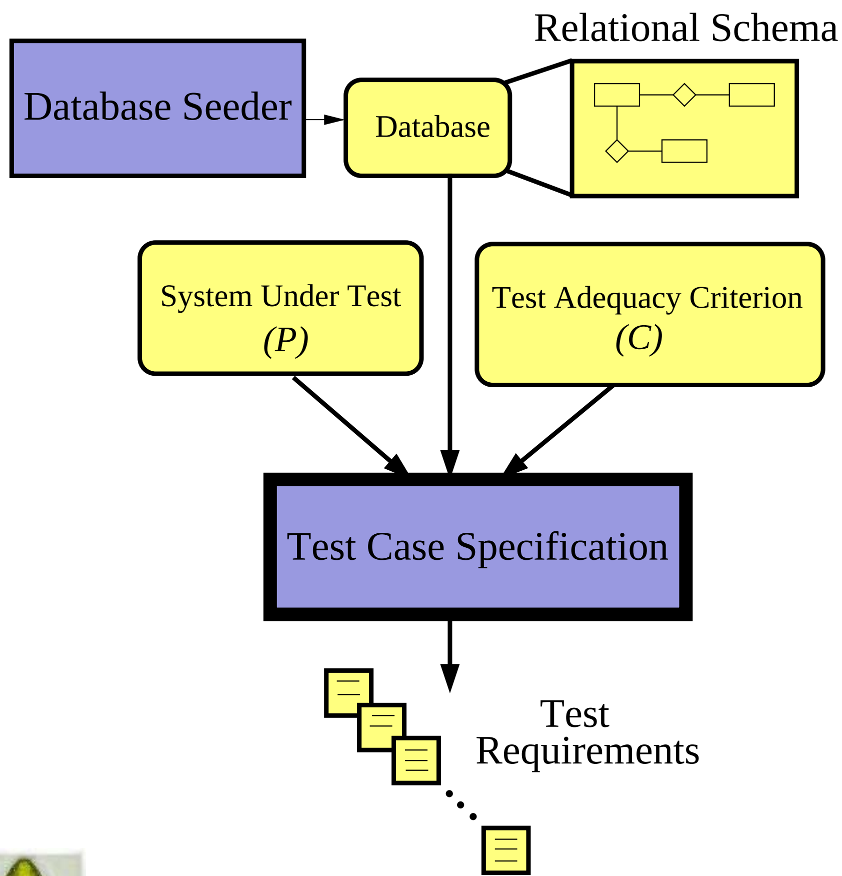
- Iteratively construct a database aware CFG to support data flow analysis and enumerate test requirements

Test Adequacy Criteria



- Database interaction association (DIA) involves the *def* and *use* of a database entity
- DIAs can be located in the DICFG with data flow analysis
- *all-database-DUs* requires tests to exercise all DIAs for all of the accessed databases

Generating Test Requirements

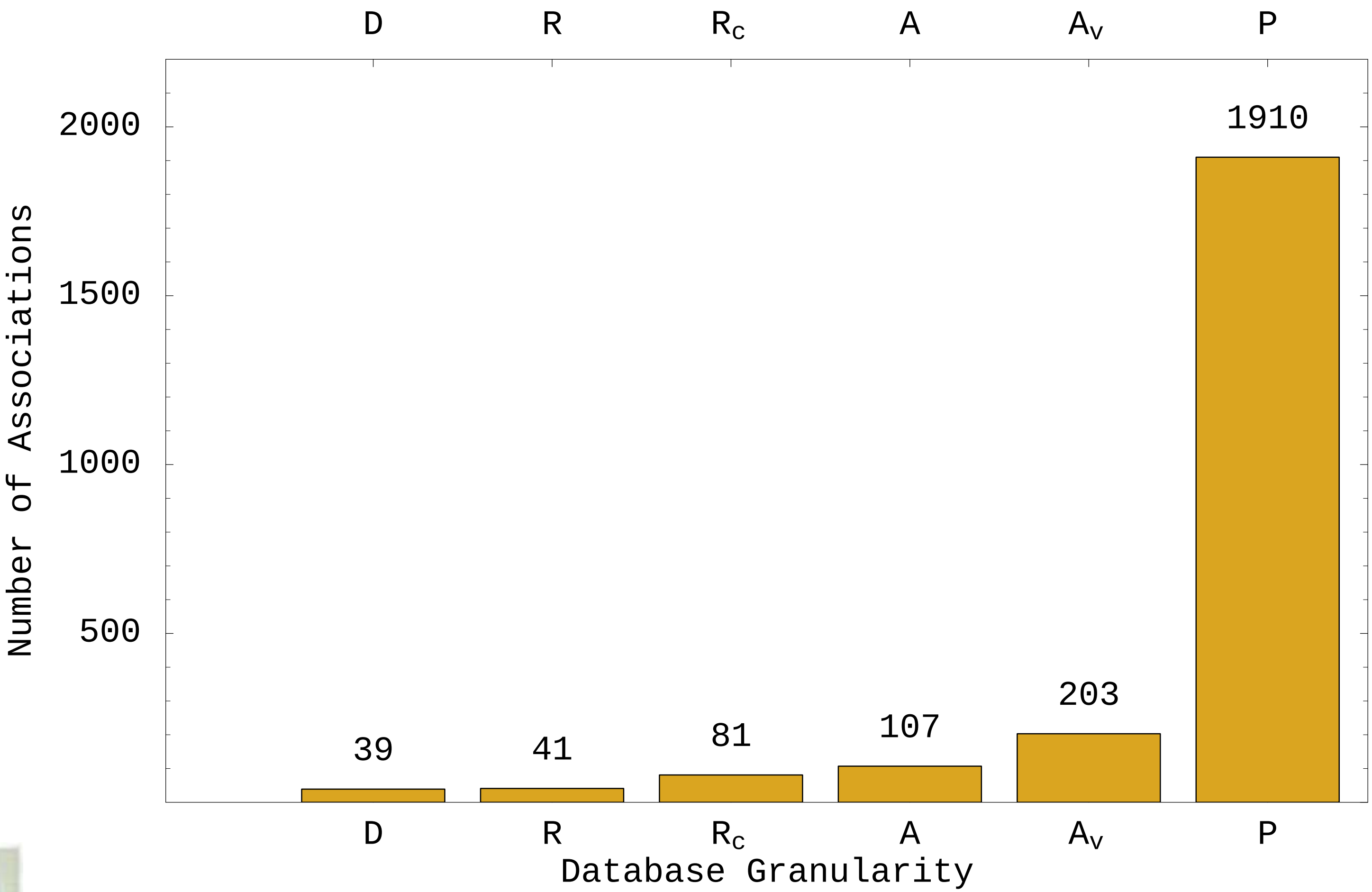


- Measured time and space overhead when computing family of test adequacy criteria
- Modifi ed `ATM` and `mp3cd` to contain appropriate database interaction points
- Soot 1.2.5 to calculate intraprocedural associations
- GNU/Linux workstation with kernel 2.4.18-smp and dual 1 GHz Pentium III Xeon processors

Experiment Goals and Design

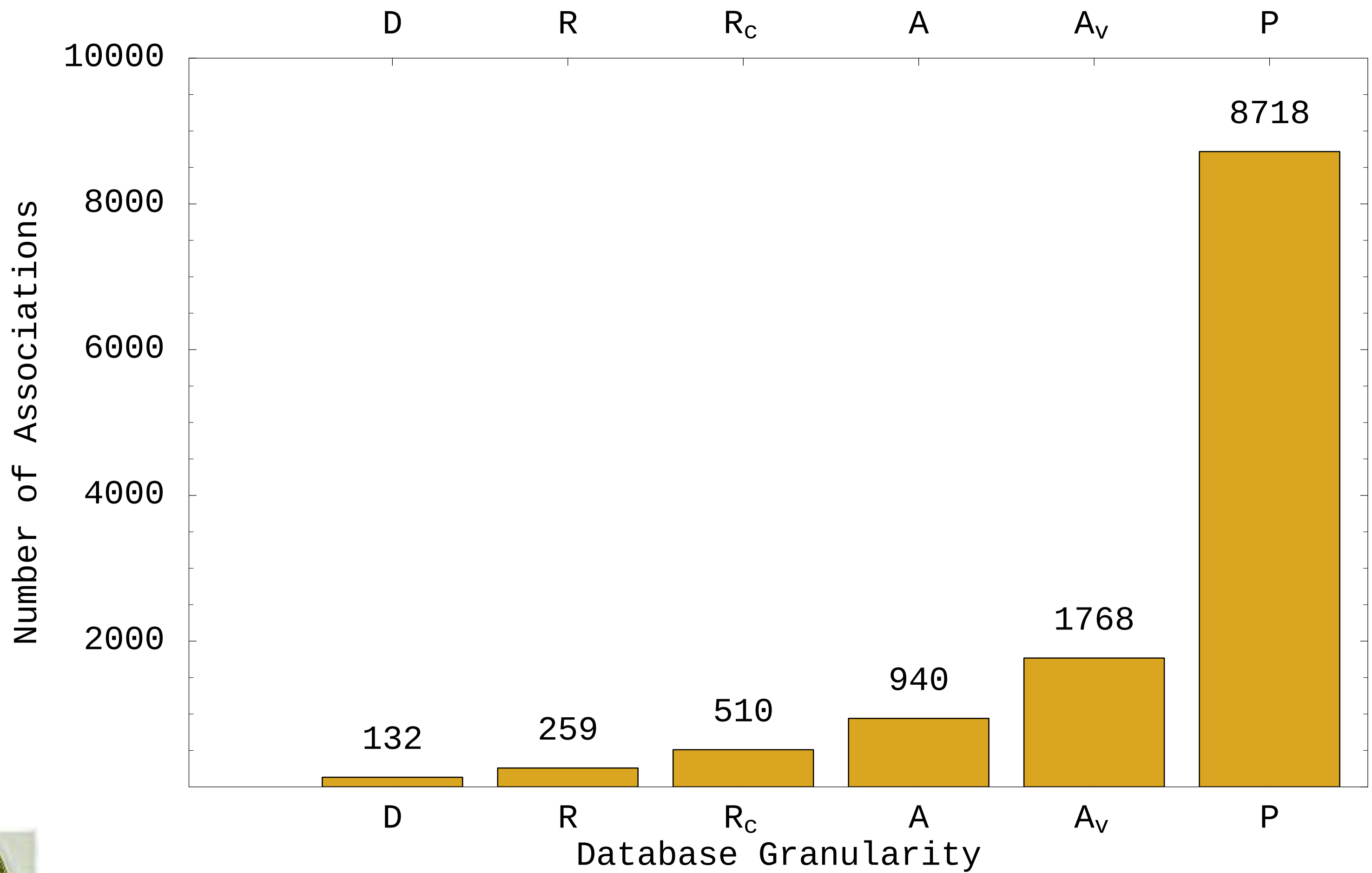
- **Reseach Question One:** Does the incorporation of database interactions yield more test requirements?
- **Reseach Question Two:** Can test requirement enumeration be performed efficiently if database interactions are included?
- **Experiment Metrics:** Number of test requirements ($\mathcal{TR}$), time overhead ($\mathcal{T}$), and space overhead ($\mathcal{S}$)
- Applications: `ATM` (1732 NCSS and 136 methods) and `mp3cd` (2913 NCSS and 452 methods)

Number of Test Requirements: ATM



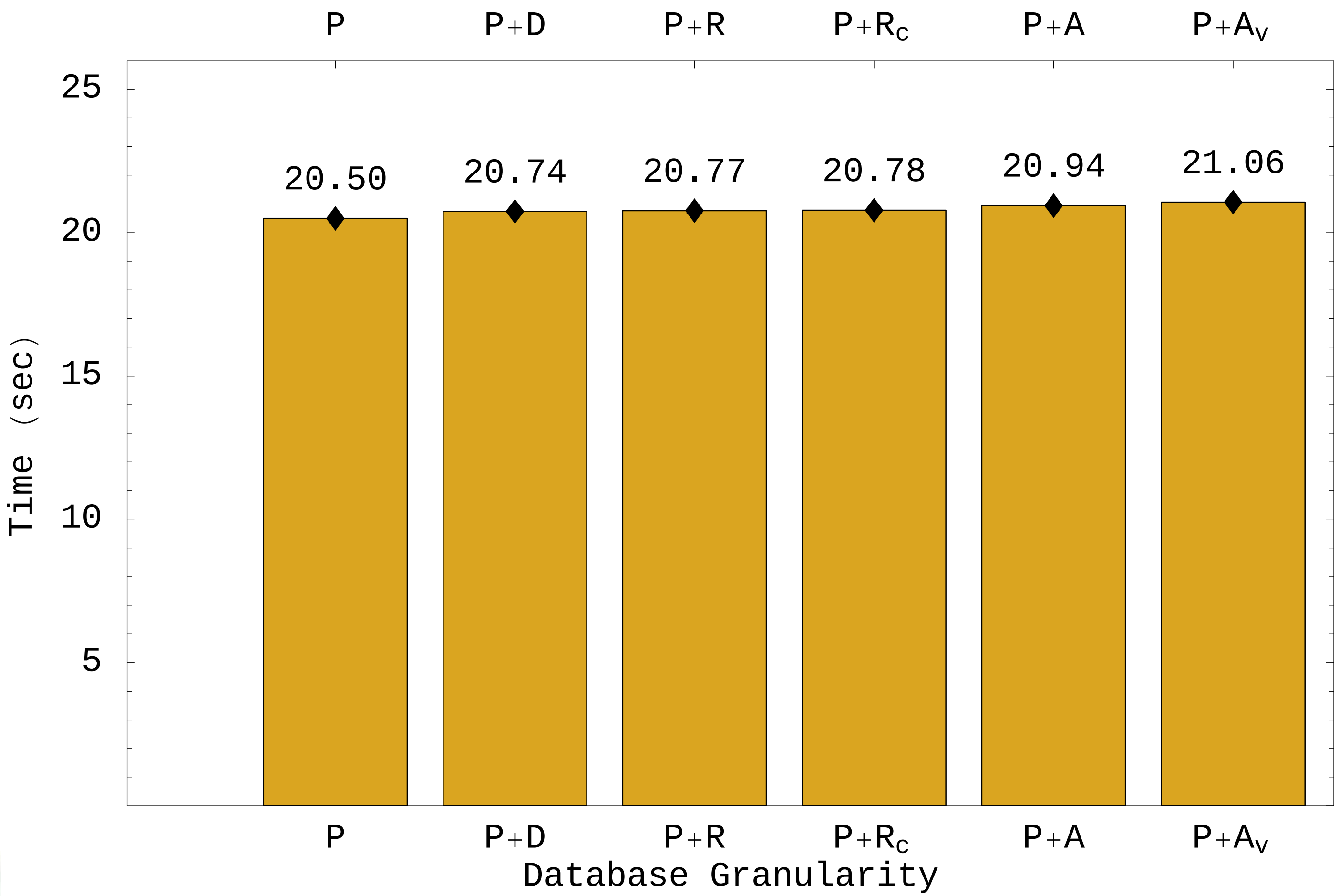
- 80.7% increase in number of test requirements from D to A_v

Number of Test Requirements: `mp3cd`



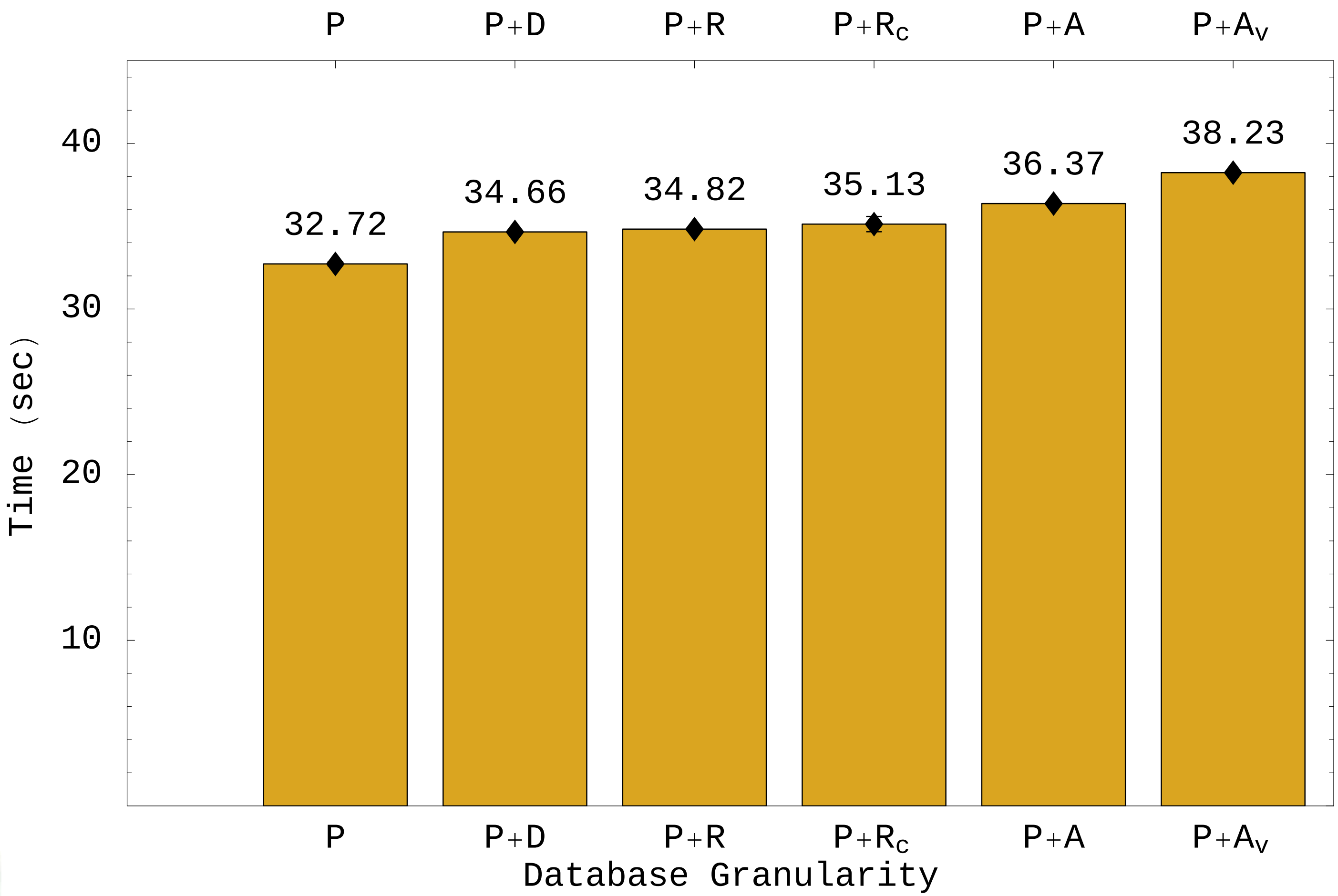
- 92.5% increase in number of test requirements from D to A_v

Time Overhead: ATM



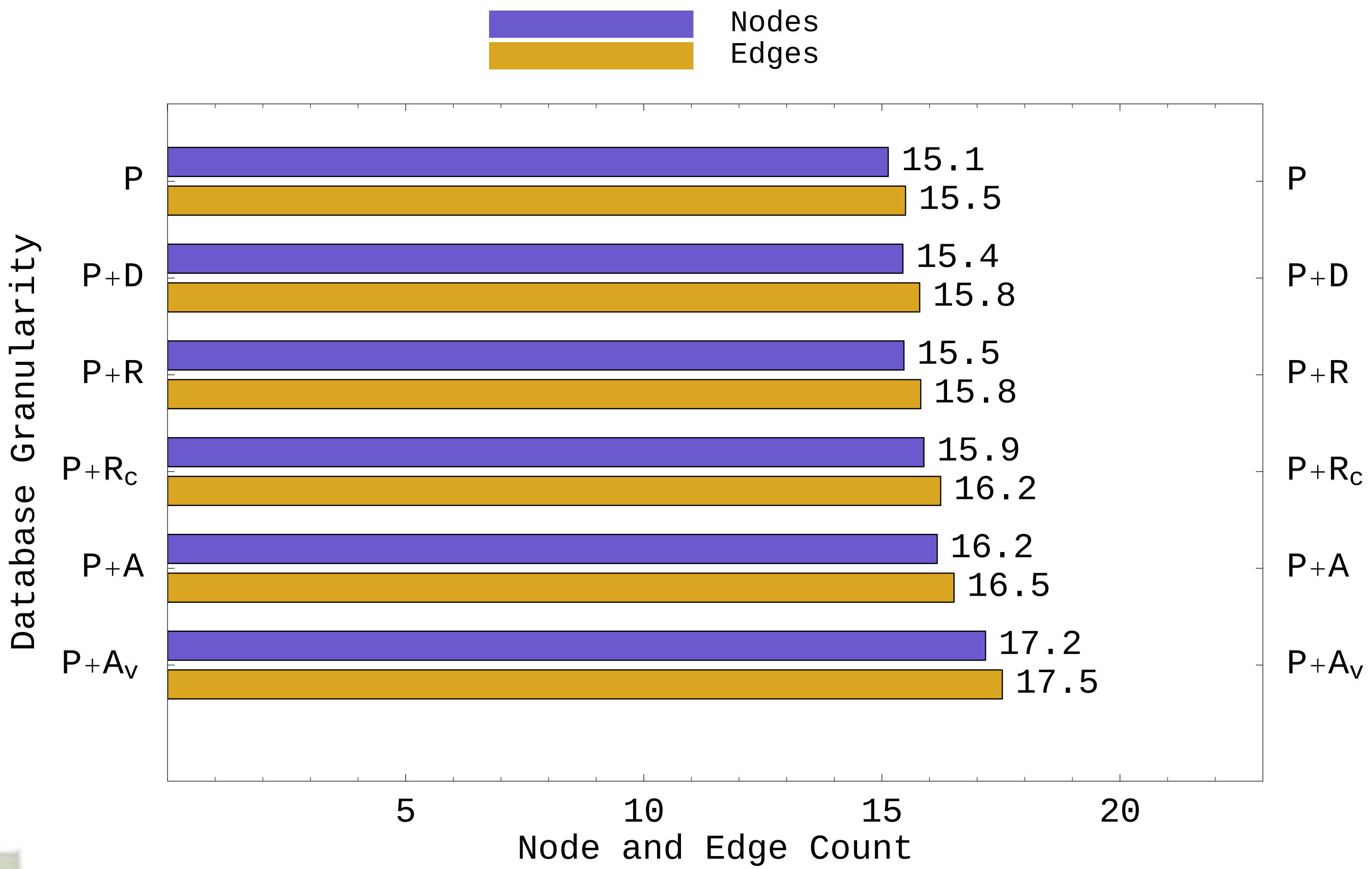
- 2.7% increase in time overhead from P to $P + A_v$

Time Overhead: `mp3cd`



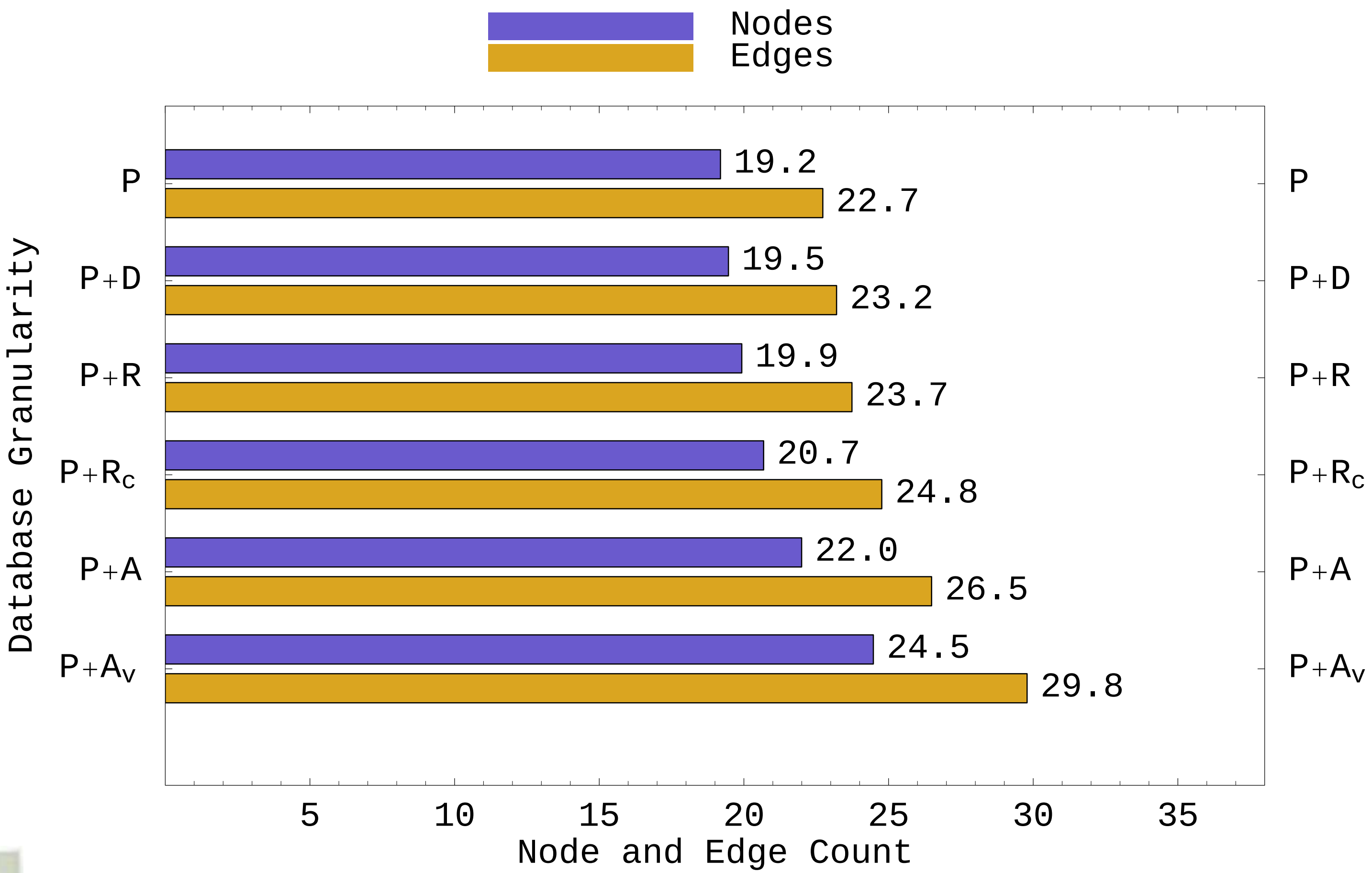
- 14.4% increase in time overhead from P to $P + A_v$

Space Overhead: ATM



- Average number of {DI}CFG nodes and edges is stable

Space Overhead: `mp3cd`



- `mp3cd` has more database interactions and larger database

Average Increase in CFG Nodes

	$\mathcal{SN}^{\%}_{\mathcal{I}}(R, D)$	$\mathcal{SN}^{\%}_{\mathcal{I}}(R_c, R)$	$\mathcal{SN}^{\%}_{\mathcal{I}}(A, R)$
`ATM`	.6	2.5	4.3
`mp3cd`	2.0	3.8	9.5

	$\mathcal{SN}^{\%}_{\mathcal{I}}(A_v, R_c)$	$\mathcal{SN}^{\%}_{\mathcal{I}}(A_v, A)$	$\mathcal{SN}^{\%}_{\mathcal{I}}(A_v, D)$	$\mathcal{SN}^{\%}_{\mathcal{I}}(A_v, P)$
`ATM`	7.5	5.8	10.4	12.2
`mp3cd`	15.5	10.2	20.4	21.6

Average Increase in CFG Edges

	$\mathcal{SE}_{\mathcal{I}}^{\%}(R, D)$	$\mathcal{SE}_{\mathcal{I}}^{\%}(R_c, R)$	$\mathcal{SE}_{\mathcal{I}}^{\%}(A, R)$
`ATM`	0.0	2.4	4.2
`mp3cd`	2.1	4.4	10.5

	$\mathcal{SE}_{\mathcal{I}}^{\%}(A_v, R_c)$	$\mathcal{SE}_{\mathcal{I}}^{\%}(A_v, A)$	$\mathcal{SE}_{\mathcal{I}}^{\%}(A_v, D)$	$\mathcal{SE}_{\mathcal{I}}^{\%}(A_v, P)$
`ATM`	7.4	5.7	9.7	11.4
`mp3cd`	16.7	11.0	22.1	23.8

Related Work

- Jin and Offutt and Whittaker and Voas have suggested that the environment of a software system is important
- Chan and Cheung transform SQL statements into C code segments
- Chays et al. and Chays and Deng have created the category-partition inspired AGENDA tool suite
- Neufeld et al. and Zhang et al. have proposed techniques for database state generation
- Dauo et al. focused on the regression testing of database-driven applications

Ongoing Research

- Test suite execution that minimizes number of costly database restarts and initializations
- Test coverage monitoring through a database interaction calling context tree (DICCT)
- Regression test suite reduction and prioritization that incorporates database aware adequacy and test case cost
- Detailed empirical studies with ten case study applications of varying code and database size
- Comprehensive tool support to assist the testing of database-centric applications

Conclusions

- Must test the program’s interaction with the database
- Test adequacy infrastructure provides : (i) database interaction fault model, (ii) unifi ed application representation, (iii) family of test adequacy criteria
- Unique family of test adequacy criteria to detect all type (1) and some type (2) violations of database validity and completeness
- Intraprocedural database interactions can be computed from a DICFG with minimal time and space overhead
- Foundation for a complete testing methodology

Further Resources

Gregory M. Kapfhammer and Mary Lou Soffa. A Family of Test Adequacy Criteria for Database-Driven Applications. In *ESEC/FSE 2003*.

Gregory M. Kapfhammer. Software Testing. CRC Press Computer Science Handbook. June, 2004.

http://cs.allegheny.edu/~gkapfham/research/diatoms/