Simple and Efficient Relational Querying

Dirk Beyer and Andreas Noack

Querying and Manipulating Relations

relations

query & manipulate relations

relations

query & manipulation commands

Dirk Beyer, Open Source Quality Meeting, Feb 2\textsuperscript{nd} 2004
Applications in Reengineering (1)

- Class diagram of a program
- Structure of a design pattern
- Query & manipulate relations
- Possible design pattern instances

Applications in Reengineering (2)

- Class diagram of a program
- Pattern of problematic design
- Query & manipulate relations
- Instances of problematic design
Applications in Reengineering (3)

- Subsystems and their connections
- Query & manipulate relations
- Violations of architectural rules

Applications in Reengineering (4)

- Function call graph
- Query & manipulate relations
- Impact of changes, unused functions
- Transitive closure command
Previous Work

- relational databases and SQL

- Prolog

- calculators for relational algebra
  - Grok [Holt 1998]
  - RPA [Feijs et al. 1998]
  - RelView [Berghammer et al. 1998]

- textual and visual graph querying languages and tools
  - GReQL and GUPRO [Kullbach, Winter 1999]
  - GraphLog and Hy+ [Consens et al. 1992]

Related Work: RDMS and SQL

![Diagram showing tables of a relational database, a relational database management system, and SQL]
Related Work: Prolog

- Prolog facts
- Prolog interpreter
- Prolog facts
- Prolog queries

Related Work: Calculators for Binary Relations

- Relations in a simple textual format
- Relations
- Relations
- Grok [Holt]
- RPA [Feijs et al.]
- RelView [Berghammer et al.]
- Binary relational algebra
**Our Approach: CrocoPat**

- **Input format for relations:** Rigi Standard Format
- **Internal representation of relations:** binary decision diagram
- **Query & manipulation language:** predicate calculus
- **Output format for relations:** Rigi Standard Format

**Contributions**

**Predicate calculus**
- as query and manipulation language
- **expressive**
  - not only binary relations (graphs), but \( n \)-ary relations
  - detection of graph patterns with more than two nodes
- well-known and fairly simple

**Binary decision diagrams (BDDs)**
- as data structure for relations
- efficient
Rigi Standard Format

- input format for relations: Rigi Standard Format
- internal representation of relations: Binary Decision Diagram
- output format for relations: Rigi Standard Format
- query & manipulation language: predicate calculus

Rigi Standard Format (RSF)

Java source code:
```java
class ContainedClass {}
class SuperClass {}
class SubClass extends SuperClass {
    ContainedClass c;
}
```

extracted RSF file:
```
INHERIT SubClass SuperClass
CONTAIN SubClass ContainedClass
```

- simple
- facilitates data exchange with other tools
Query & Manipulation Language

- Input format for relations: Rigi Standard Format
- Internal representation of relations: Binary Decision Diagram
- Output format for relations: Rigi Standard Format
- Query & manipulation language: predicate calculus

Syntax

expression ::= (expression) | TRUE | FALSE | atomic_expression | term = term | ! expression | expression ^ expression | expression + expression | expression -> expression | EX (variable, expression) | FA (variable, expression) | TC (expression, variable, variable)

atomic_expression ::= relation_variable (term_list)
term_list ::= term | term_list, term
term ::= "constant" | variable
**Example: Composite Design Pattern**

CompositePattern(Component, Composite, Leaf) :=
   \[ \text{INHERIT(Composite, Component)} \]
   \[ \wedge \text{CONTAIN(Composite, Component)} \]
   \[ \wedge \text{INHERIT(Leaf, Component)} \]
   \[ \neg \text{CONTAIN(Leaf, Component)}; \]

**Example: Cyclic Dependency**

Depend(C1,C2) := CALL(C1,C2)
   + CONTAIN(C1,C2)
   + INHERIT(C1,C2);
DependTrans(C1,C2) := TC(Depend(C1,C2), C1, C2);
InCycle(C1) := EX(C2, DependTrans(C1,C2) \wedge (C1=C2));
Internal Representation of Relations

- input format for relations: Rigi Standard Format
- internal representation of relations: Binary Decision Diagram
- output format for relations: Rigi Standard Format
- query & manipulation language: predicate calculus

The Need for Efficiency

- Graph patterns with \( n \) nodes are \( n \)-tuples of nodes.
- For a graph with 1000 nodes, there are \( 1000^n \) \( n \)-tuples of nodes.

\[ \rightarrow \] To find graph patterns with more than 2 nodes, huge relations have to be represented and manipulated.

The data structure **Binary Decision Diagram (BDD)** [Bryant 1986] can represent huge relations efficiently.
Binary Decision Diagram

- results from collapsing isomorphic subtrees of a decision tree
  - redundancy is exploited to compress the representation

Experimental Evaluation: Systems

<table>
<thead>
<tr>
<th>System</th>
<th>Classes</th>
<th>LOC</th>
<th>RSF Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>JHotDraw 5.2</td>
<td>168</td>
<td>17 819</td>
<td>878</td>
</tr>
<tr>
<td>JDK 1.4.0 AWT</td>
<td>384</td>
<td>141 267</td>
<td>1 504</td>
</tr>
<tr>
<td>JWAM 1.6</td>
<td>2 397</td>
<td>284 818</td>
<td>12 298</td>
</tr>
<tr>
<td>JDK 1.4.0</td>
<td>5 312</td>
<td>1 179 576</td>
<td>28 699</td>
</tr>
<tr>
<td>Eclipse 2.02</td>
<td>8 925</td>
<td>1 181 270</td>
<td>63 121</td>
</tr>
</tbody>
</table>

- CrocoPat 2.1, MySQL 3.23.48 , Grok R15.0
- Linux PC with 1 GHz AMD Athlon CPU
- memory usage restricted to
  - 50 MB for CrocoPat,
  - 400 MB for MySQL and Grok
Experimental Evaluation: Queries

- Component
  - Leaf
  - Composite

Composite Design Pattern

- Super
  - Sub
- Super
  - Sub
  - DegSub

Degenerate Inheritance

Subclass Knowledge

- Super
- Sub
- C

Cyclic Dependency

Experimental Evaluation: Results

<table>
<thead>
<tr>
<th></th>
<th>JHotDraw</th>
<th>JDK AWT</th>
<th>JWAM</th>
<th>JDK</th>
<th>Eclipse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composite</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CrocoPat)</td>
<td></td>
<td></td>
<td></td>
<td>3.9</td>
<td>9.8</td>
</tr>
<tr>
<td>Deg. Inheritance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CrocoPat)</td>
<td></td>
<td></td>
<td></td>
<td>4.7</td>
<td>9.8</td>
</tr>
</tbody>
</table>

< 3 s
## Experimental Evaluation: Results

<table>
<thead>
<tr>
<th></th>
<th>JHotDraw</th>
<th>JDK AWT</th>
<th>JWAM</th>
<th>JDK</th>
<th>Eclipse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subcl. Knowledge</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Grok)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subcl. Knowledge</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CrocoPat)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cyclic Dependency</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Grok)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cyclic Dependency</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CrocoPat)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt; 3 s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>75.6 s</td>
<td>&gt;400 MB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8.3 s</td>
<td>30.3 s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>115 s</td>
<td>&gt;400 MB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8.4 s</td>
<td>31.9 s</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Experimental Evaluation: MySQL vs. CrocoPat

- transitive closure of the union of `CALL`, `CONTAIN`, and `INHERIT`

<table>
<thead>
<tr>
<th></th>
<th>JHotDraw</th>
<th>JDK AWT</th>
<th>JWAM</th>
<th>JDK</th>
<th>Eclipse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transitive Closure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(MySQL)</td>
<td>0.35 s</td>
<td>29.5 s</td>
<td>16.8 s</td>
<td>&gt;400 MB</td>
<td>&gt;400 MB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>or &gt;1 h</td>
<td>or &gt;1 h</td>
</tr>
<tr>
<td>Transitive Closure</td>
<td>0.18 s</td>
<td>0.27 s</td>
<td>2.24 s</td>
<td>8.37 s</td>
<td>30.9 s</td>
</tr>
<tr>
<td>(CrocoPat)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Applications

• completed: integration with Sotograph, a commercial software analysis workbench

• ongoing: integration with visualization

• for your application, CrocoPat 2.1:
  – is Open Source
  – has Quality
  – is available at:
    http://www.google.com/search?q=CrocoPat

Conclusions

The predicate calculus based language
• is fairly easy to use and
• sufficiently expressive for many structural software analyses, including the detection of graph patterns.

The BDD based implementation
• scales well to the analysis of large systems.

The tool CrocoPat
• is easy to integrate with other tools.
• is applied in industry.