ELEC 875
Design Recovery
and
Automated Evolution

Week 3
Grok and Standard Transforms
Next Week


Relational Databases

• On Disk Data Structures
  ◊ optimized for huge databases
    - many millions of records
  ◊ optimized for IT based queries
  ◊ `select avg(sales)
    from employee
    where commission > 0.5`
  ◊ `select manager
    from employee
    where name = “James Higgins”`
  ◊ allows update to small number of records

• Spectacular for these types of queries
Program Analysis Queries

- **example**
  - Common Ancestor Subsystem of Two modules
    - equivalent IT query: common boss of two employees
    - requires recursive SQL (in latest version)
  - requires multiple queries to the same table
- updates to single records are rare
- often add entire derived relations to the database
- some individual queries
- Queries often need to use every record in the relation
- Relational DBs not optimized for these types of queries
  - not surprising, very minuscule portion of database use.
Groks

- Initial Version in 1995, Ric Holt
- Optimized for large Databases
  ◦ hundreds of thousands of facts
- Heinlein - Stranger in a Strange Land
- Relational Algebra Calculator
  ◦ Discrete Math
  ◦ Sets and Relations
- Ram Based
  ◦ Queries tend to use entire relations at a time
  ◦ Recursive Queries
- 32 Bit only, java version called JGroks available
Grok - Input of Relations

- RSF - Rigi Standard Format
  ◊ triple format
  ```
  func def main main.c
  def loc main "main.c:10"
  include main.c stdio.h
  calls main foo
  sets foo x
  parameter foo y
  ```

- Automatic discovery of domain and range sets
  ◊ just use names in relations
- Attributes are just another relation
Grok - Input of Relations

- TA - Tuple Attribute format
  - ER based notation
  - Definition of instances
  - Attributes instead of relations

```c
funcdef main main.c
defloc main "main.c:10"
```

```c
$INSTANCE main {defloc = "main.c:10"}
```

- Relations can also be extended

- translated to RSF internally
Grok - Input of Relations

- TA-Schema Definition
  - Allows the user to specify the schema of the data
  - Not explicitly checked
  - Schema is also compiled into relations
  - Can write a grok program that checks the data against the schema
    - already done
Grok - Operators

- Sets
  - construction
    \[
    \text{functions} = \{ "main", "foo", "bar", "bat" \}
    \text{vars} = \{ "m", "x", "y" \}
    \text{refs} = \{ "x", "z" \}
    \]
  - union/intersection/complement
    \[
    \text{ents} = \text{functions} + \text{vars}
    \text{vrefs} = \text{vars} \cap \text{refs}
    \text{vnrefs} = \text{vars} - \text{refs}
    \]
  - cardinality
    \[
    \text{numvars} = \#\text{vars}
    \]
- sets can be read and written to files, one entity per line
Grok - Operators

- Relations
  - Cross Product
    \[ \text{foo} = \text{functions} \times \text{refs} \]
  - Relations are sets of tuples, so all set operators work on relations in the obvious way
  - domain/range(codomain)
    \[ f = \text{dom foo} \]
    \[ r = \text{rng refs} \]
  - relation composition
    \[ h = f \circ g \equiv \{ (x,y) \mid y = g(f(x)) \} \]
GroK - Operators

- Relations
  - Id constructor (S is a set)
    \[ r = id S \implies \{(x,x)\} \text{ for all } x \text{ in } S \]
  - Inverse (n is a relation)
    \[ m = inv n \quad \text{i.e. } n^{-1} \]
  - Transitive closure
    \[ R^+ \]
  - Transitive, reflexive closure
    \[ R^* \]
Grok - Operators

• Sets and Relations

◊ projection (s is set, R is relation)

\[ s.R = \{ y \mid x \in S \text{ and } (x,y) \in R \} \]

\[ R.s = s . \text{inv} \ R \]

\{"f","g"\} . invokes == all functions invoked by f and g

\{"f","g"\} . invokes+ = all functions invoked directly or indirectly by f and g

\{"f","g"\} . invokes* = all functions invoked directly or indirectly by f and g including f and g.
Grok - Scripting

• Grok also has a scripting language:
  ◊ conditionals (if)
  ◊ looping
  ◊ arguments
  ◊ file io

• Other numerous options including options to ask for names of sets, relations and variables, string operations, id operations, file I/O
Relational Algebra Practice..

the types of all fields of subclasses of the class ‘transact’
Wins and Losses..

- General maintenance queries

- Some easy (win), some not so easy (loss)
Standard Relations

- Contains - in DMM
  \( C := \text{inv defines}^* \circ \text{contains} \circ \text{defines}^* \)

- Use relation
  - routine uses a var, or a routine invokes a function
  \( U := \text{sets} + \text{uses} + \text{invokes} \)

- Parent (\( P := \text{inv } C \))
- Sibling (\( S := P \circ C - \text{ID} \))
- Descendent (\( D := C^+ \))
- Ancestor (\( A := P^+ \))
Lifting

• a routine/method invokes a routine/method in DMM
• a routine/method sets/uses a variable (g/f/l)

• Want to compute relation between classes/files

\[ \text{HLU} := (D \circ U \circ A) - \text{ID} - D - A \]
defines * allows us to use source elements
Lifting

- Sometimes need to filter to a layer

- If more than two levels, links all
  \[ \text{HLU} := (D \circ U \circ A) - ID - D - A \]

- Restrict to a Layer
  \[ \text{HLU}_2 := \text{HLU} \land (S \times S) \]
Hide Interior

- Hide nodes "inside" a given element
  - i.e. contained...
  - includes a lift as part of the transformation (NewU)

\[
\begin{align*}
S &:= \{ "\text{the element}" \} \\
SD &:= S \cdot D \text{ the set of all elements contained} \\
TargU &:= SD \cdot U - SD \text{ all nodes used by } SD \\
SrcU &:= U \cdot SD - SD \text{ all nodes that use } SD \\
NewU &:= (S \times TarU) \text{ all nodes used by } SD \text{ are used by } S \\
&\quad + (SrcU \times S) \text{ all nodes that use } SD \text{ use } S \\
delset SD
\end{align*}
\]
Others

• Hide Exterior - narrow the graph to a particular subsystem
• Diagnostic - for a given lifted edge, find the lower level edge that caused it
• Sifting - finding nodes with a given characteristic
  - example is nodes that only used are leaf nodes, while nodes that use others are higher
• Kidnapping - refactoring
  - method or field that is used more by other classes?
  - routine in wrong file?
  - does not actually change the code (what-if)
Losses

• patterns must be specified in relational algebra
  - no real memory between queries, or of paths.

- grok has scripting, and imperative statements, so can build relations iteratively keeping temporary results
  - no longer pure relational algebra