ELEC 875
Design Recovery
and
Automated Evolution

Week 3
Grok and Standard Transforms

Next Week

Michael L. Van De Vanter, "The Documentary Structure of Source Code" Journal of Information and Software Technology, Elsevier, Volume 44, No 13, pp. 767-782

Vaclav Rajlich, N. Wilde, "The role of Concepts in Program Comprehension" Proc. 2002 International Workshop on Program Comprehension (IWPC'02), June 2002, Paris, 271-278.

I.T. Bowman, R. Holt, N.V. Brewster, "Linux as a Case Study: It's Extracted Software Architecture", Proc. 21st International Conference on Software Engineering (ICSE'99), May 1999, Los Angeles, pp. 555-563

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.

Grok

- 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 JGrok available

Grok - Input of Relations

- RSF - Rigi Standard Format
  - ◊ triple format

```
funcdef main main.c
defloc 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

```
funcdef main main.c
defloc main “main.c:10”

$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

    functions = { “main”, “foo”, “bar”, “bat” }

    vars = { “m”, “x”, “y” }

    refs = {“x”, “z”}
  - ◊ union/intersection/complement

    ents = functions + vars

    vrefs = vars ^ refs

    vnrefs = vars - refs
  - ◊ cardinality

    numvars = #vars
  - ◊ sets can be read and written to files, one entity per line

Grok - Operators

- Relations
  - ◊ Cross Product

    foo = functions X refs
  - ◊ Relations are sets of tuples, so all set operators work on relations in the obvious way
  - ◊ domain/range(codomain)

    f = dom foo

    r = rng refs
  - ◊ relation composition

    h = f o g == { (x,y) | y = g(f(x))}

Grok - Operators

- Relations
  - ◊ Id constructor (S is a set)

    r = id S === {(x,x)} for all x in S
  - ◊ inverse (n is a relation)

    m = inv n ----- 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 | x in S and (x,y) in R)}
    R.s = s . inv R

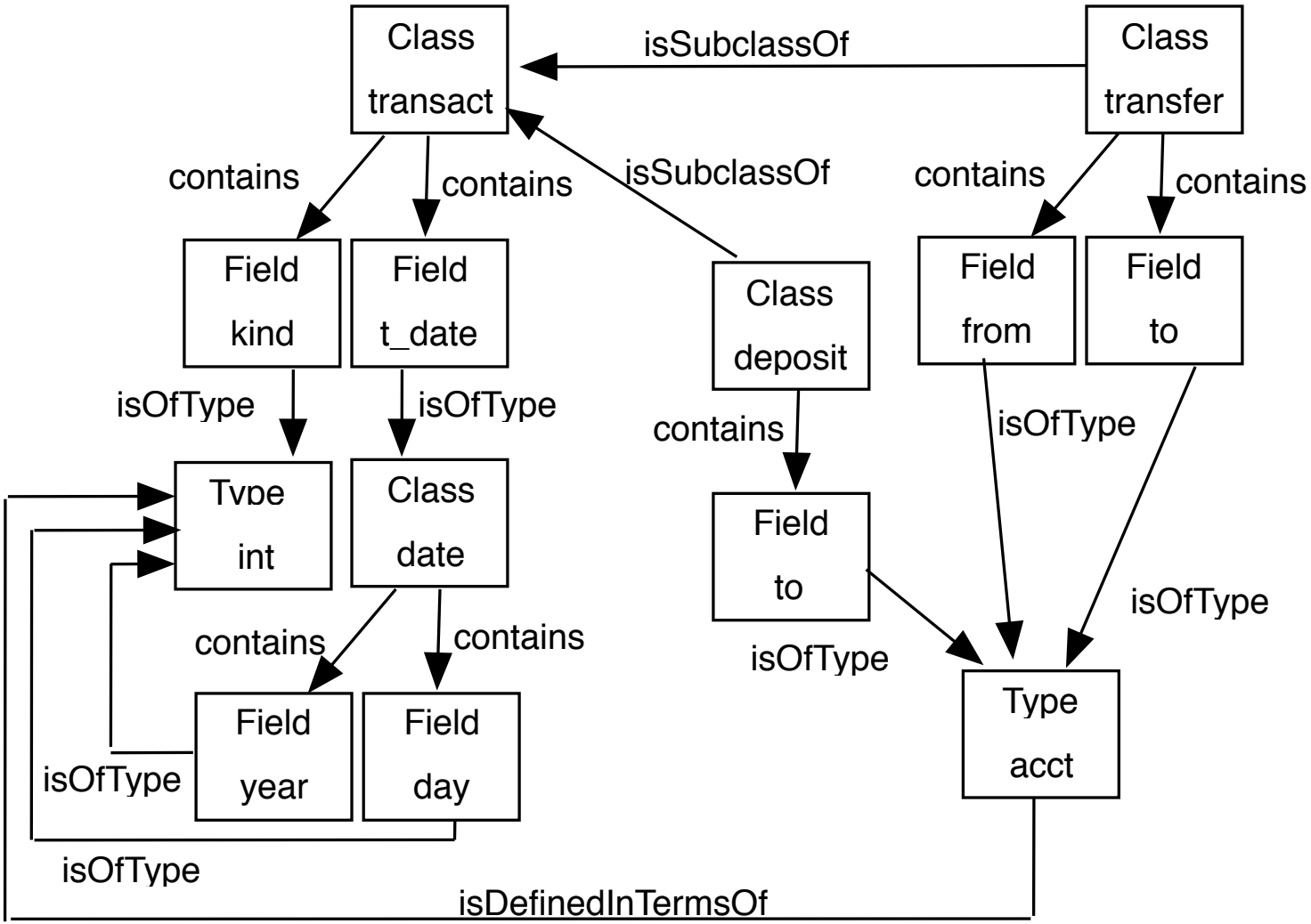
{“f”,”g”} . invokes == all functions invoked by f and g
{“f”,”g”} . invokes+ = all functions invoked directly or indirectly by f and g
{“f”,”g”} . invokes* = all functions invoked directly or indirectly by f and g including f and g.

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 := inv defines* o contains o defines*

- Use relation

  - routine uses a var, or a routine invokes a function

  U := sets + uses + invokes

- Parent (P := inv C)
- Sibling (S := P o C - ID)
- Descendent D := C+
- Ancestor A:= P+

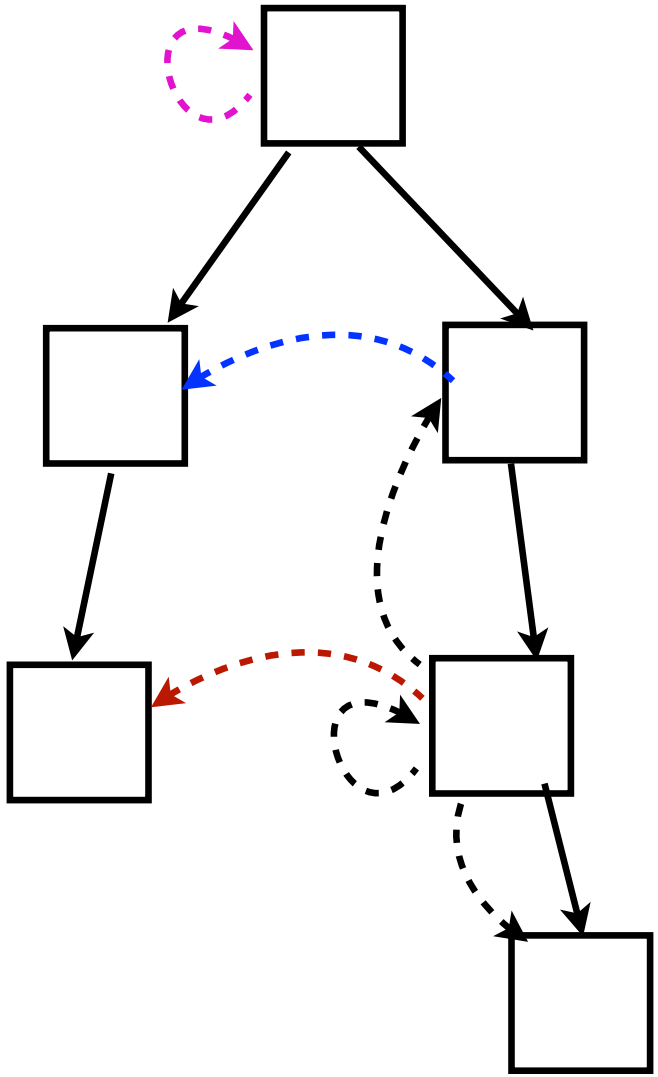
Lifting

- 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

  HLU := (D o U o A) - ID - D - A

defines * allows us to use source elements

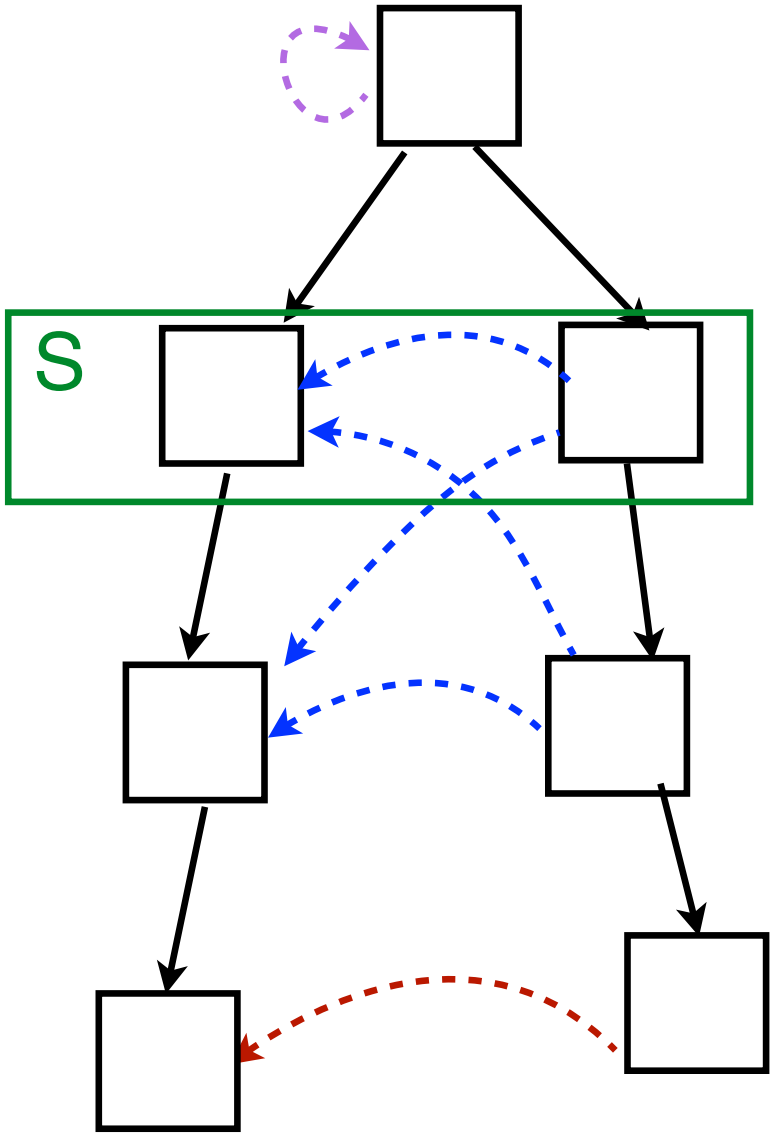


Lifting

- Sometimes need to filter to a layer
- If more than two levels, links all

  HLU := (D o U o A) - ID - D - A

- Restrict to a Layer
  - HLU2 := HLU ^ (S x S)



Hide Interior

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

S:= { "the element" }
SD := S . D *the set of all elements contained*
TargU := SD . U - SD *all nodes used by SD*
SrcU := U . SD - SD *all nodes that use SD*
NewU := (S x TarU) *all nodes used by SD are used by S*
+(SrcU x S) *all nodes that use SD use S*
delset SD

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

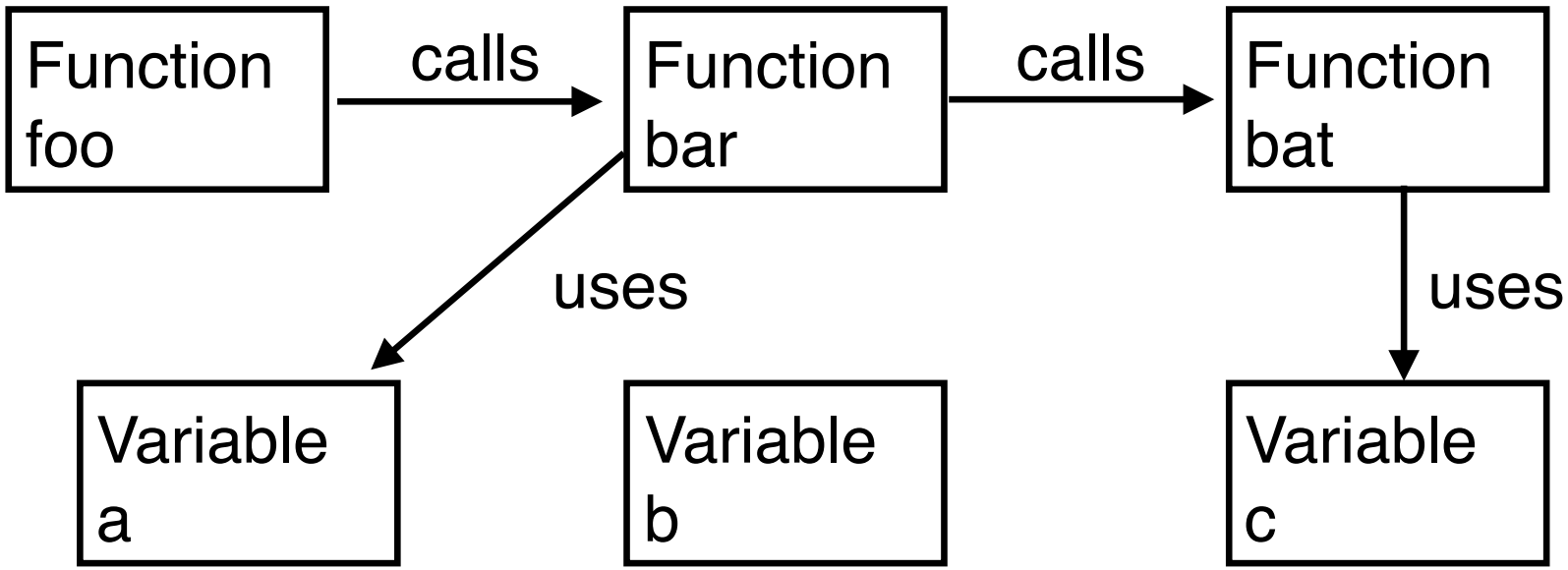
Current Status

- Grok is older, written in Turing Plus
  - 32bit version for Linux
- jgrok is available, written in java and jar files
  - Source code available.

Alternatives

- graph database servers (e.g. graphdb)
  - RDF (XML based triples)
  - SPARQL query language
- Ontology + Inferencing
  - e.g. can specify usesObject and usesComponent are subclasses of uses relation.
  - e.g. can add rules that infer higher level concepts from lower level concepts

Graph Database



Graph Database

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix : <http://pyxis.ece.queensu.ca/dmm#> .

:DMMEntity a rdfs:Class .
:Function rdfs:subClassOf :DMMEntity .
:Variable rdfs:subClassOf :DMMEntity .

:calls a rdf:Property .
:calls  rdfs:domain :Function .
:calls  rdfs:range :Function .

:uses a rdf:Property .
:uses rdfs:domain :Function .
:uses rdfs:range :Variable .
```

Graph Database

:foo a :Function .
:bar a :Function .
:bat a Function .
:a a :Variable .
:b a :Variable .
:c a :Variable .

:foo :calls :bar .
:bar :calls :bat .

:bar :uses :a .
:bat :uses :c .

SPARQL Queries

```
PREFIX : <http://pyxis.ece.queensu.ca/dmm#>
select ?v where {
    :foo :calls+/:uses ?v .
}
```

- variables used by functions called by foo (a,c)

```
PREFIX : <http://pyxis.ece.queensu.ca/dmm#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select * where {
   ?v rdf:type :Variable .
   FILTER (
    NOT EXISTS { ?f :uses ?v .}
   )
}
```

- unused variables (b)