Current view of AP

- Use benefits of finite automata at class graph level to specify traversals as partial programs
- Use visitors to decorate traversals
- Use adjusters to organize traversals and visitors

New forces

- Mitch’s traversal automata
- Mendelzon’s graph patterns, WebSQL, WebOQL, schema-free data model
- Smaragdakis’ suggestions on strategies

Theory of Traversals

- Influence: SIAM J. Comput. paper by Alberto Mendelzon and Peter Wood: Finding Regular Simple Paths in Graph Databases, 24:6 pages 1235-1258, 1995
  - Conference version: VLDB 1989

History

- Relational model: simple for users and mathematicians. Query languages (relational calculus and relational algebra) not expressive enough (transitive closure of a binary relation not expressible).
- More expressive query languages: Datalog (Ullman) and G+ (Cruz, Mendelzon, Wood)

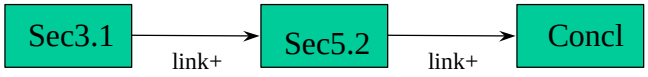
G+

- based on graph traversals
  - Database is a directed labeled graph (corresponding to an object graph in our model)
  - Queries are graph patterns expressed using regular expressions: A graph pattern is a labeled graph:
    - node labels are constants to be matched with db
    - edges are labeled with regular expressions

    (corresponding to our strategies)

Example: Pattern Graph

- Is there a way to go from Section 3.1 to section 5.2 and then to the conclusion without reading any node more than once? (focus on *simple* paths). G = hypertext document. Pattern Graph:



Abstract problem

- REGULAR SIMPLE PATH:
  - Instance: Regular expression R and graph G.
  - Question: Is there a directed simple path p in G satisfying R, where the concatenation of edge labels comprising p is in the language denoted by R.
- Surprise: REGULAR SIMPLE PATH is NP-complete

Abstract problem:

- FIXED REGULAR SIMPLE PATH (R):
  - Instance: Regular expression R and graph G.
  - Question: Is there a directed *simple* path p in G satisfying R, where the concatenation of edge labels comprising p is in the language denoted by R.
- Surprise: FIXED REGULAR SIMPLE PATH(R) is NP-complete for R = (00)*

Related problem

- PATH VIA NODE
  - Instance: Directed graph G=(N,E), and nodes x,y,m in N.
  - Question: Is there a directed simple path from x to y via m?
- PATH VIA NODE is NP-complete

Abstract problem

- REGULAR PATH:
  - Instance: Regular expression R and graph G.
  - Question: Is there a directed path p in G satisfying R, where the concatenation of edge labels comprising p is in the language denoted by R.
- REGULAR PATH is in P.

Proof 1

- Given graph G along with nodes x and y in G, we can view G as an NDFA with initial state x and final state y. Construct the intersection graph I of G and an NDFA M accepting L(R). There is a path from x to y satisfying R if there is a path in I from (x,s0) to (y,sf), for s0 the start state of M and some final state sf in M.

Proof 1

- All this can be done in polynomial time by Hunt, Rosenkrantz and Szymanski, 1976.

Proof 2, using Tarjan 1981

- Tarjan provides a polynomial algorithm for constructing a regular expression R_{xy} which represents the set of all paths between two nodes *x* and *y* of a given graph.
- Is there a path between *x* and *y* satisfying *R*:
  - construct R_{xy}
  - determine whether intersection of *L(R)* and $L(R_{xy})$ is nonempty using NDFAs.

Connections, Implications

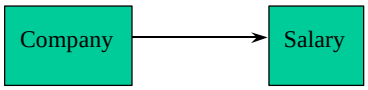
- Toronto approach has only graph patterns (similar to strategies) and database graphs (similar to object graphs). No knowledge about structure of database graphs; i.e., no schema = class graph. Well, they use cycle constraints.
- Toronto paper contains useful facts to better understand traversals and their limitations.

Structure-shyness in Toronto approach

- pattern graph gives topology of navigation
- _* (underscore matches any edge label)

Pattern graph

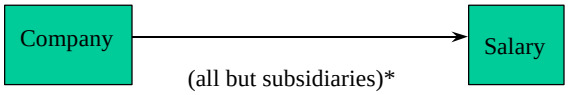
Strategy graph



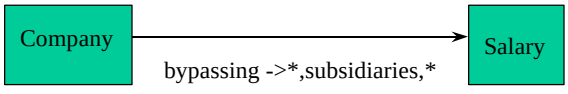
Structure-shyness in Toronto approach

- Bypassing

Pattern graph



Strategy graph



Structure-shyness in Toronto approach

- Toronto approach uses regular expressions
  - positive and negative
  - may be confusing: pattern graph positive
- Strategy graphs use constraint maps
  - negative: what we want to avoid.
  - leave it open how to specify maps
  - could use regular expressions to specify constraint map

Toronto approach

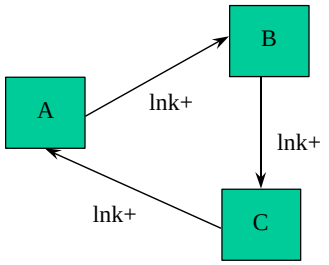
- Shows how to deal with structure-shyness without schema = class graph.
- Proposes uniform automata-based approach to AP also suggested byYannis Smaragdakis
  - all graphs correspond to finite automata: class, strategy, traversal and object graphs
  - Algorithm 1: intersection of two automata.
  - Algorithm 2: intersection of traversal graph and object graph.

What is still new in strategies

- uses schema = class graph = constraints on object graphs. Provides compilation algorithm. Deals with abstract classes and subclass edges. Three level model.
- Model has same expressiveness as graph patterns.
  - Can specify constraint maps using regular expressions: eliminate edges not contained in any path defined by regular expression

What is role of graph in graph patterns?

- (A _+ B _+ C _+)* not correct
- A (_+ B _+ C _+ A)*
- put node names also in paths



A source and target

_:any edge, any node

Regular expressions only

- Can we express any graph pattern or strategy graph as a regular expression?

John Lamping's proposal

operator	regular expression
[A,B]	A.any*.B
through lnk	any*.lnk.any*
bypassing lnk	not(any*.lnk.any*)
through C	any*.C.any*
bypassing C	not(any*.C.any*)
d1 join d2	[d1].[d2] // join point twice!
d1merge d2	[d1]\cup[d2]
not d1	not([d1])
only-through A,b,B	A.b.B

Strategies are shorter

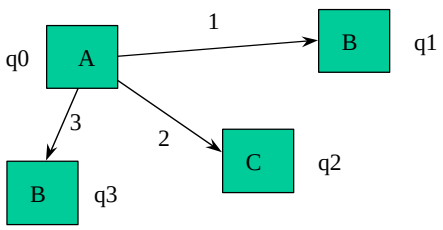
The following to be improved

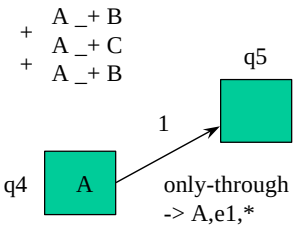
- Traversal automata for strategy graphs

Ordering of edges in strategies

A q0
  traverse to B q1
  traverse to C q2
  traverse to B q3
A q4
  traverse e1 q5

actions before [A,B] and before [A,C], for example





Regular expressions

- define path sets
- but we want to define traversals with specific orderings of path sets

Traversal automata

- Allow us to control ordering and sequencing of traversals (and when to call visit operations)
- Control can be based on
  - edges in class graph
  - edges in strategy graph (new refinement)

Traversal Automata

```
ClassValuedVar1 State1
  traverse relationValuedVar1 State2
  traverse to ClassValuedVar2 State3 following constraint1
  traverse to  ClassValuedVar3 State4 following constraint2
```

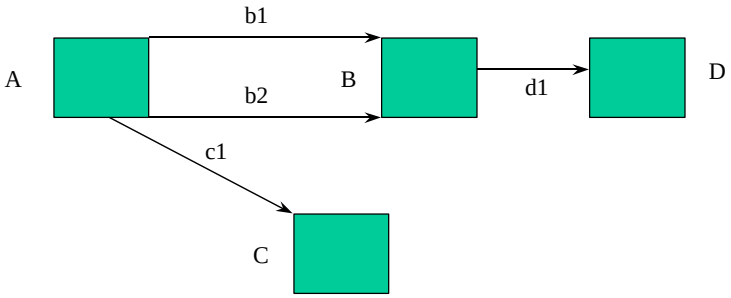
When a class graph is given, the traversal automaton is expanded into:

```
Class1 State1
  traverse relation1 State2
  traverse relation2 State1  // at target switch to State3
  traverse relation3 State1  // at target switch to State4
```


Better view

- So far, traversal automata were defined in terms of class graphs.
- Now we define them in terms of strategy graphs: Need to give names to edges. Edge names are abbreviations of constraints associated with edge.

New kind of strategy graph: looks like a class graph



b1=bypassing ->*,x,*
b2=only-through ->A,b,B
c1=no restriction

A,B,C: class valued variables
x,b: relation-valued variables

Strategic Traversal Automata

A State1 traverse b1 State2
traverse b2 State3
traverse c1 State4
traverse b1 State5
B State2 traverse d1 State5
State5 //nothing

When a class graph is given, the strategic traversal automaton is expanded into a traversal automaton for the class graph.

Benefits: can use standard traversal automata, promotes parts-free programming.

New role of strategies

- Define blueprint for traversal automata.
- Strategic traversal automaton defines a few default traversal automata:
  - DFS, class graph order (what we use now)
  - DFS, strategy graph order
  - BFS, class graph order
  - BFS, strategy graph order

What is new?

- Traversal automata are expressed in terms of class-valued and relation-valued variables.
- Detailed traversals are expressed at higher level: at strategy graph level
- Strategy graphs now have the structure of class graphs with concrete classes only and all parts required.

Expansion

- Given a class graph, translate a strategic traversal automaton to a traversal automaton
  - write traversal graph as traversal automaton and expand it following information in strategic traversal automaton.
    - reorder traversals
    - add more traversals

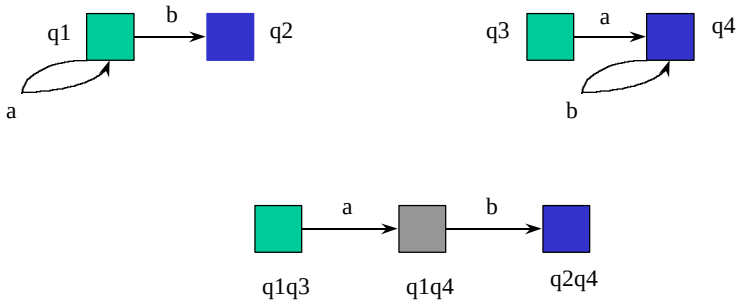
Evolution

- Graphs (object graphs), need to traverse, know about their structure (class graph), formulate traversals at class graph level using PL. Has flavor of traversal automata. Implementation:
  - state-less: leads to exponential size code
  - with state: becomes efficient

Evolution (continued)

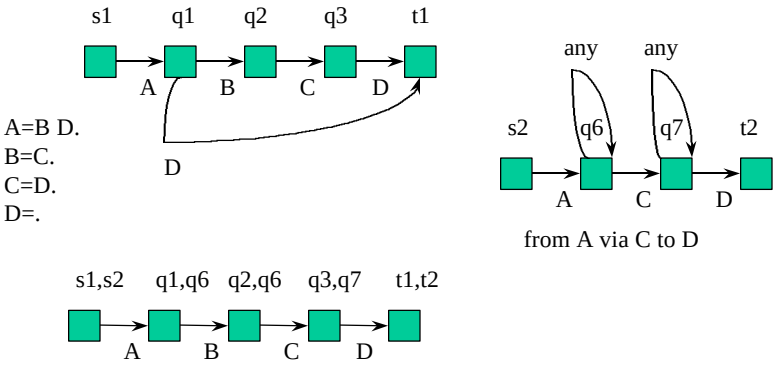
- Instead of using PL, use strategies: abstraction of class graph using regular expressions over class graph. We lose some of the flexibility of the traversal automata solution: strategies define only certain default traversals.
- Solution: use strategic traversal automata to gain flexibility.

Intersection of NDFA is similar to traversal graph construction



Intersection of two DFAs

Traversal Graph Construction



Integrated view of algorithms 1 and 2: for path existence

- Both are similar to the intersection of two NDFAs
  - Algorithm 1: NDFA for strategy graph and NDFA for class graph: results in NDFA for traversal graph.
  - Algorithm 2: NDFA for traversal graph and NDFA for object graph: results in NDFA which tells us whether there is a non-empty traversal

Recall: Intersection of NDFAs

- An NDFA is a 5-tuple: *M=(S,A,d,p0,F)*, *S* finite set of states, *A* is input alphabet, *d* is a state transition function which maps *Ax(S union epsilon)* to the set of subsets of *S*, *p0* is the initial state, and *F* is the set of final states.

Recall: Intersection of NDFAs

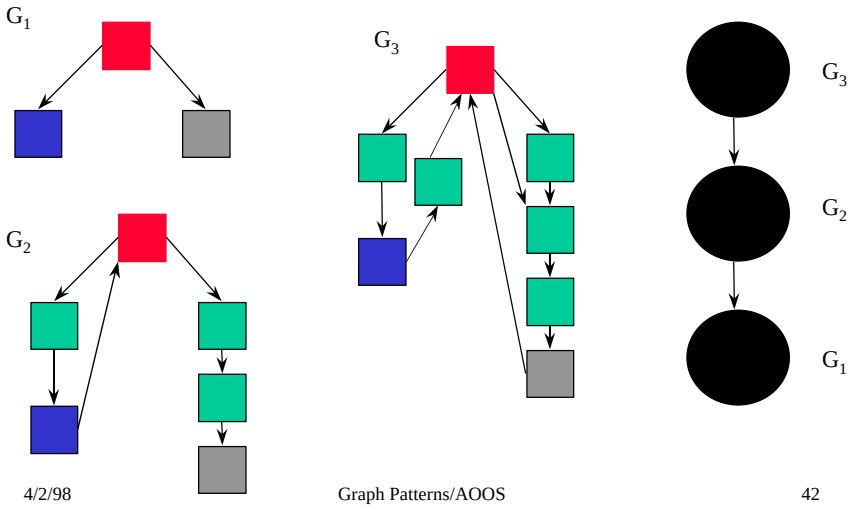
- *M1=(S1,A,d1,p0,F1)* and *M2=(S2,A,d2,q0,F2)*.
- The NDFA for M1 intersect M2 is *I=(S1xS2,A,d,(p0,q0),F1xF2)*, where for *a* in *A*, *(p2,q2)* in *d((p1,q1),a)* if and only if *p2* in *d1(p1,a)* and *q2* in *d2(q1,a)*.

Graph Layers

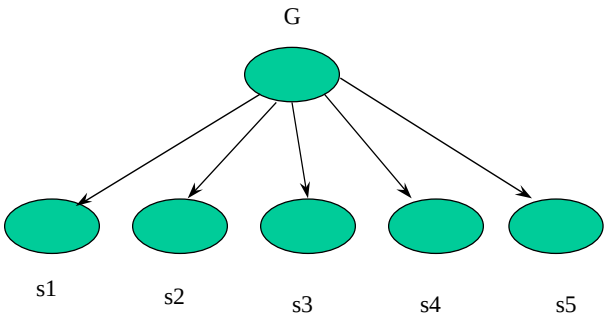
graph layers G1, G2, … ,Gn
Gi is an abstraction of Gi+1
Can embed Gi in Gi+1
Paths in Gi exist in Gi+1 in expanded form
Gi determines traversals in Gi+1
Hierarchy of graph refinements

Use traversal automaton if necessary

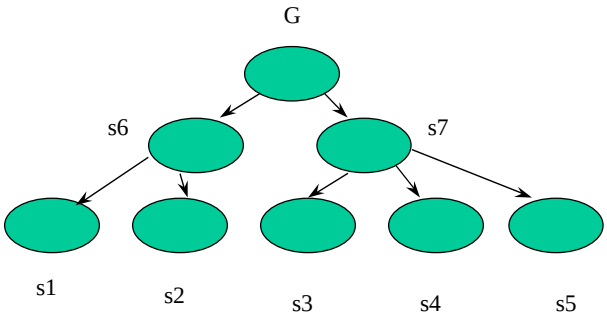
Hierarchy



Current way of AP

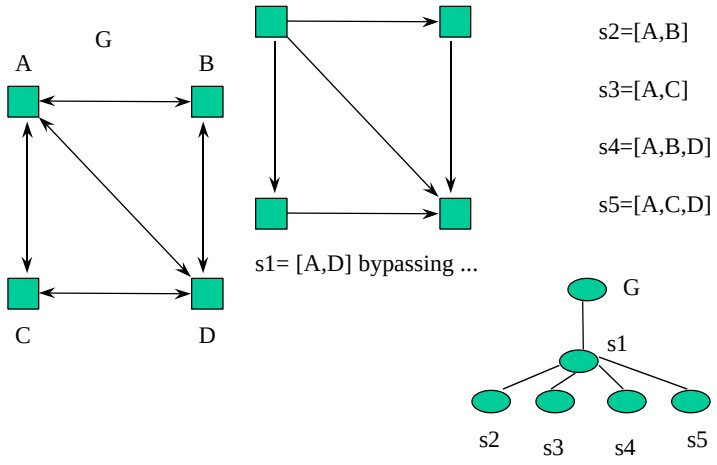


Better way of AP



s6, s7 shield s1 through s5 from changes to G

Hierarchical development of strategies



Layered strategies

- strategies of the form (-> A {B,C,D} bypassing ...) reduce the graph size and result is still a graph.
- Should strategies at inner nodes be of this form?

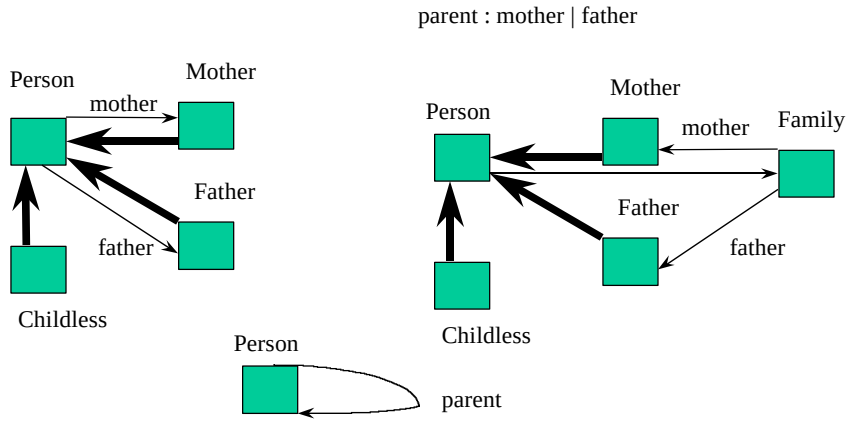
“New” view of strategy graphs

- So far we mapped strategy graphs into class graphs.
- Why not map them into object graphs?
- The purpose of strategy graphs is to express algorithms in a structure-shy way.
- In some cases better achieved by mapping strategies into object graphs.

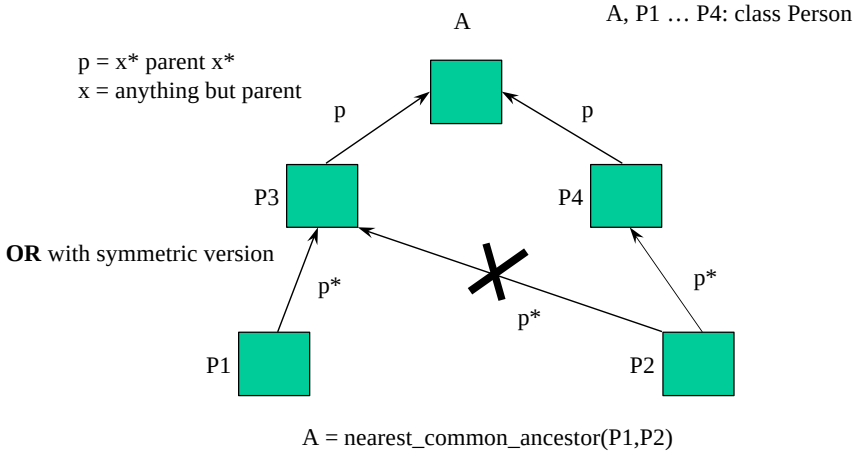
Some surprises along the way

- Extend strategies with regular expressions on edges
- Express that certain paths are not allowed to exist

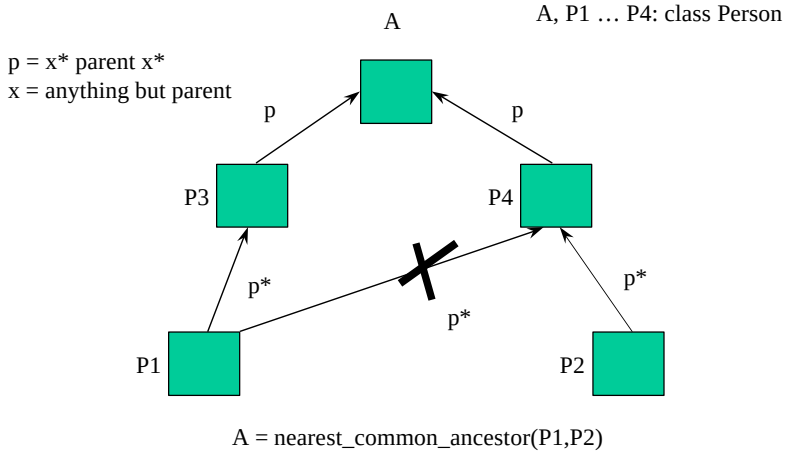
Nearest Common Ancestor



Strategy graph



Strategy graph: symmetric



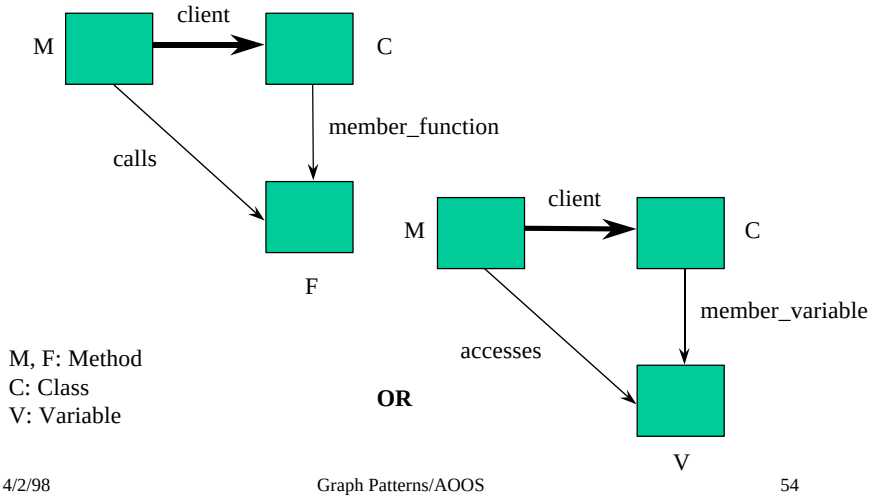
Strategy graphs have changed

- Constraints are regular expressions
- Nodes are mapped to objects
- Also express relationships which are not allowed to exist

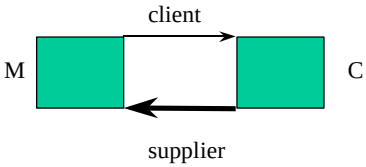
Implementation

- Pattern matching for graphs
- When pattern matches, execute code. Pattern matching visitor
- Need to do search for desired pattern

Law of Demeter/client

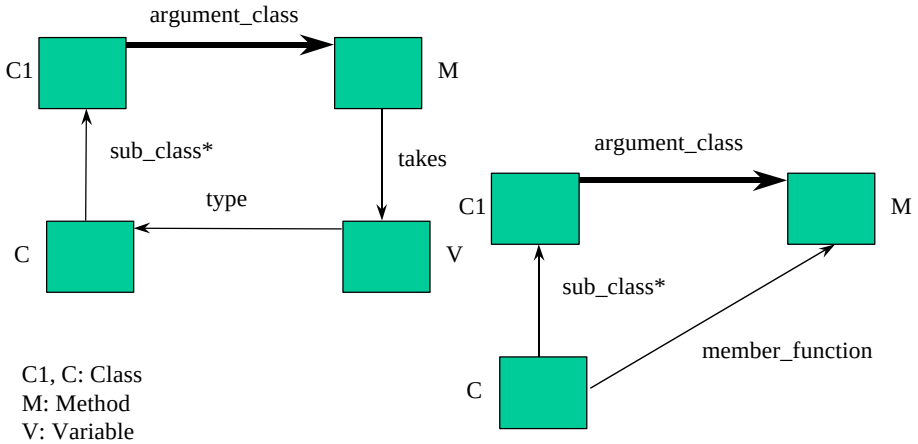


Law of Demeter/supplier



M: Method
C: Class

Law of Demeter/argument class



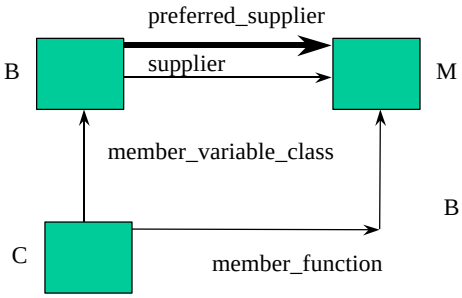
C1, C: Class
M: Method
V: Variable

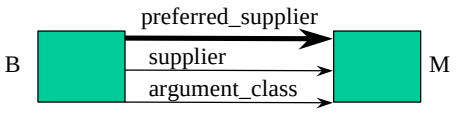
OR

Law of Demeter (simplified)

All suppliers must be preferred

B, C: Class
M: Method



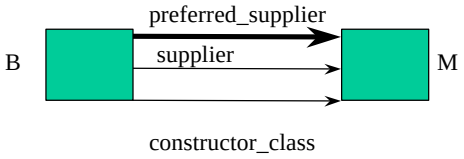


OR

Law of Demeter

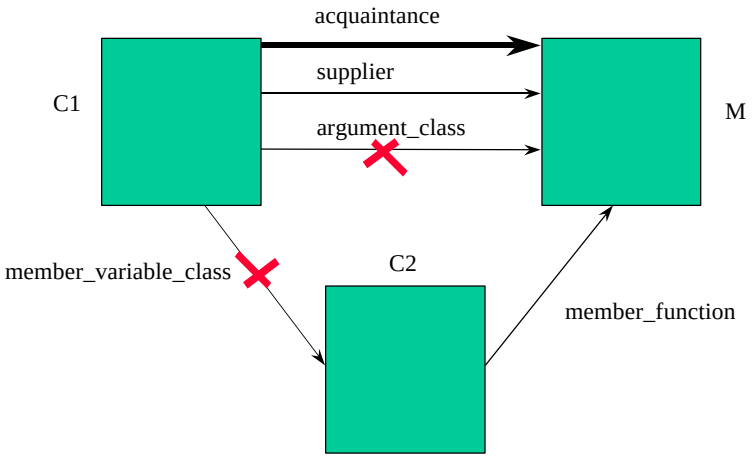
constructor_class:
M calls a constructor of class B

B: Class
M: Method



OR

Other example of negation



GraphLog

- Visual query language for the Hy+ visualization system
  - M. Consens, Master Thesis, 1989, Ph.D. 1995
  - M.Consens, A. Mendelzon: SIGMOD ‘90
- Query processing: translate queries and data into logic programs for execution by
  - LDL
  - CORAL

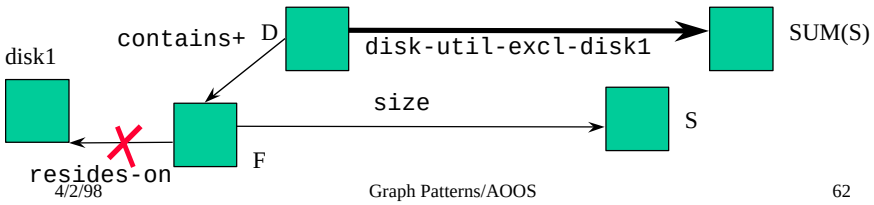
GraphLog

- Many databases can be naturally viewed as graphs.
- Even a relational db: can be represented by a directed multigraph having an edge labeled r(c1,…,ck) from a node labeled (a1,…,ai) to a node labeled (b1,…,bj) corresponding to each tuple (a1,…,ai,b1,…,bj,c1,…ck) of relation r.

small case: constant (disk1)
upper case: variable (D)

Recursive query with negation

```
tc-contains(D,F) <- contains(D,F).
tc-contains(D,F) <- tc-contains(D,E),
  contains(E,F).
disk-util-excl-disk1(D,SUM(S)) <-
  tc-contains(D,F), size(F,S), not
  resides-on(F,disk1).
```



Logic Programs

- p,q predicate symbols
- atom: p(X1,…,Xn) or X=Y
- literal: positive or negative atom
- Horn clauses `P <- A,B,C`

Complexity

- queries expressible in GraphLog are exactly those which are evaluable in non-deterministic logarithmic space in the size of the database.

Differences: Graph Patterns/ Strategies

- straight-line strategies are easily expressible as graph patterns
- but cyclic strategies are not expressible as graph patterns. If a graph pattern is cyclic, then also the matching objects must be cyclic.

Differences: Graph Patterns/ Strategies

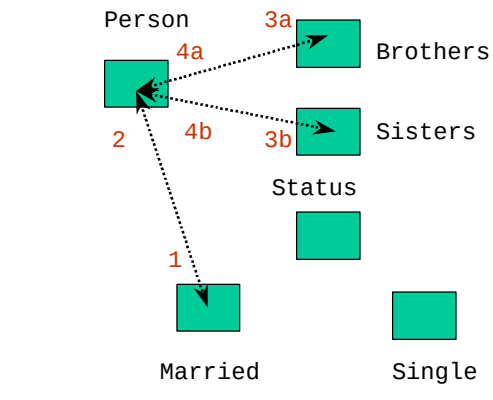
- On the other hand: cyclic graph patterns are not expressible as strategy graphs
- But cyclic graph patterns have problems with path summarization

Duplication

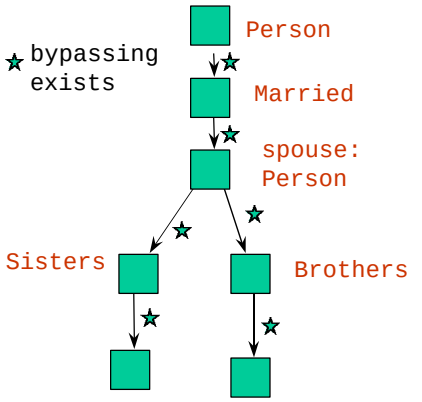
- Graph patterns duplicate information

Traversal dependent roles

Class graph with super-imposed strategy graph



Strategy graph

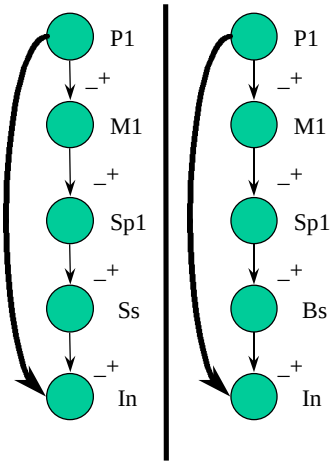


sisters_in_law: Person

brothers_in_law: Person

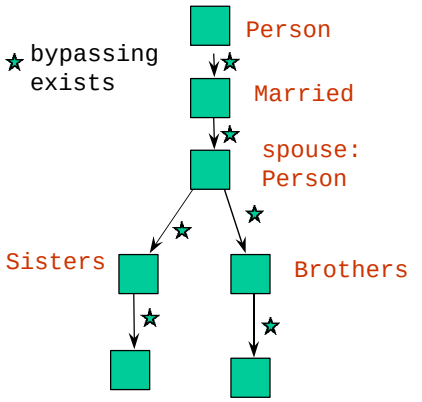
Traversal dependent roles

Graph pattern



Strategy graph

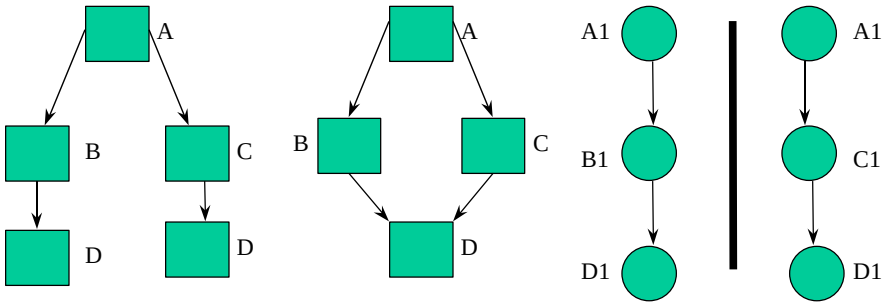
In_Law



sisters_in_law: Person

brothers_in_law: Person

Strategies and graph patterns



strategy graph = strategy graph graph pattern

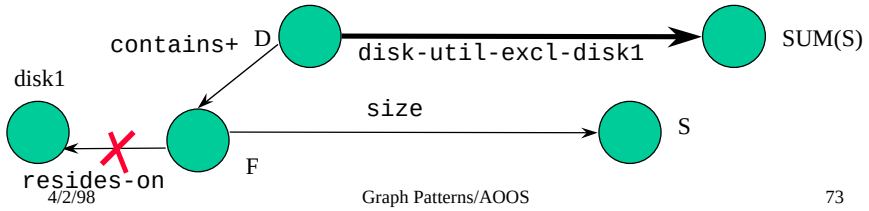
Express graph patterns as adaptive programs

- Not a general translation?

small case: constant (disk1)
upper case: variable (D)

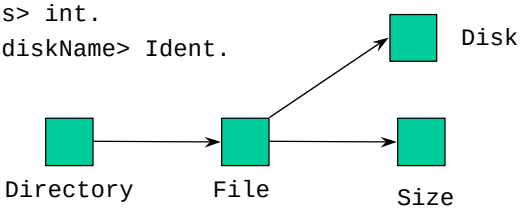
Recursive query with negation

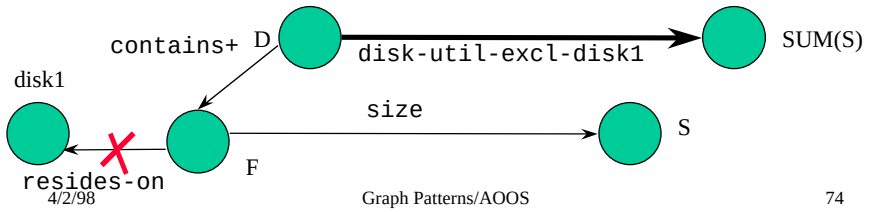
```
tc-contains(D,F) <- contains(D,F).
tc-contains(D,F) <- tc-contains(D,E),
  contains(E,F).
disk-util-excl-disk1(D,SUM(S)) <-
  tc-contains(D,F), size(F,S), not
  resides-on(F,disk1).
```



With AP

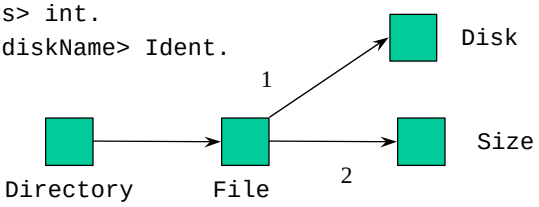
```
Directory = <contains> List(Directory) <files> List(File).
File = <residesOn> Disk <size> Size.
Size = <s> int.
Disk = <diskName> Ident.
```





With AP

```
Directory = <contains> List(Directory) <files> List(File).
File = <residesOn> Disk <size> Size.
Size = <s> int.
Disk = <diskName> Ident.
```

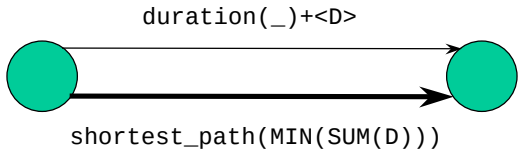


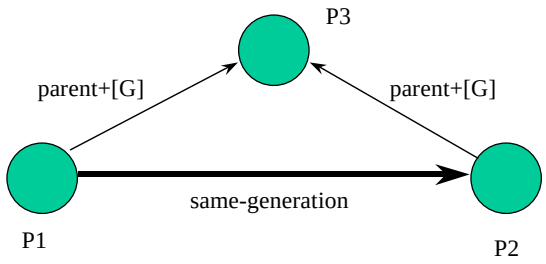

```
Directory {
  int disk_util_excl_disk1() to {Disk,Size} {
    int total; Ident dn;
    init (@ total = 0; @)
    before Disk (@ dn = diskName)
    before Size (@ if !(dn.equal("disk1")) total += s;
    return (@ total @)
  }
}
```


My current view

- Graph patterns work well for certain special cases. They have similar structure-shy properties as adaptive programs.
- But AP is more general and works in all cases.
- Graph patterns do not have enough benefits to warrant a special syntax?

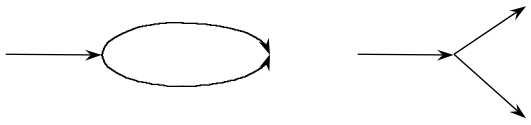
Variants of Graph Patterns





Questions

- What is the complexity of strategy equivalence? substrategy checking?
  - some algebraic identities: D1*(D2+D3)=D1*D2+D1*D3



Questions

- What is the complexity of traversal equivalence? subtraversal checking?
  - Relevant result: Determining whether a regular expression over {0} does not denote 0* is NP-complete. Hence, inequivalence of regular expressions is NP-hard. (see Mendelzon)

Need for intersection

- A=B.B=C.C=.&A=X.X=Y.Y=C.
- A=X.X=B.B=Y.Y=C.C=. or A=X.X=Y.Y=B.B=C.C=. etc. would be much longer

Composition of adaptive programs

- Two strategy graphs
  - A=B.B=C.C=.
  - A=X.X=Y.Y=C.
- Want to do both traverals in one:
  - A=B.B=C.C=. & A=X.X=Y.Y=C.
  - class graph: A=B.B=X.X=Y.Y=C.C=. yes
  - class graph: A=B X.B=C.C=.X=Y.Y=C. no

Succinct specification of path sets in graphs

- strategy graph*
- traversal automaton
- strategic traversal automaton*
- graph pattern*
- regular expression with any*
- regular expression

*: not self-contained; needs class graph

Regular expressions of two kinds

- Self-contained
- with respect to a graph A _* B means: take any edge in the graph from A to B. This is more constrained than an ordinary regular expression.
  - Can be extended into a self-contained regular expression where details of graph are encoded

Succinct specification of path sets for families of graphs

- strategy graph
- strategic traversal automaton
- graph pattern
  - regular expressions are using any*: any edge
  - “any” is modulo a graph

Slogan of Adaptive Programming

- Apply automata theory at software architecture-level to control navigation through software architectures.
- Why is automata theory good for structure-shyness? Regular expressions allow "wildcards:
  - engine (subpart)* name
  - engine _* name