Navigation in Object Graphs

Mitch Wand
and
Karl Lieberherr

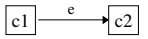
Searching for Reachable Objects

- Task: Given an object o1 of class c1 in an object graph, find all objects of type c2 that are reachable from o1.
- Assumptions: we know the class structure that describes the object graph, but we know nothing else about the object graph except the class of the current object.

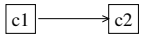
Search using meta information

- we could visit the entire object but that
  - would be wasteful or
  - might lead to wrong results

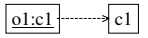
Classes and Objects: Basic Notations



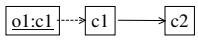
Class c1 has a part e of type c2



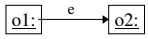
Class c1 inherits from class c2



Object o1 is of class c1

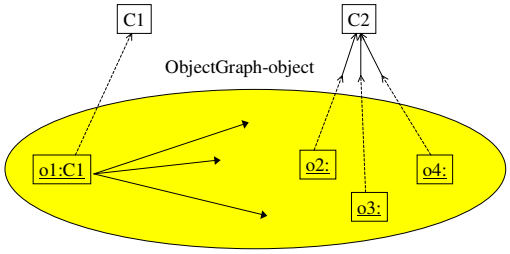


Object o1 is of type c2
(i.e., its class is a subclass of c2)



Object o1 has a part e which is object o2

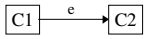
Finding the first step for the search



Which arrows might lead to an object of type C2?

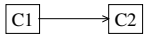
Traversal Strategy: from C1 to C2

Relations between Classes



e(C1,C2)

C(C1,C2) (that is, e(C1,C2) for some e)

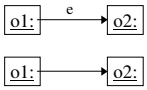


<=(C1,C2), =>(C2,C1)

Class(o1) = C1

Object o1 is of type C2:
Class(o1) <= C2

Relations between Objects



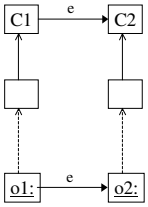
e(o1,o2)

O(o1,o2) (that is, e(o1,o2) for some e)

Operations on Relations

- R.S = {(x,z) | exists y s.t. R(x,y) and S(y,z)}
- R* = reflexive, transitive closure of R

Possible edges in the object graph

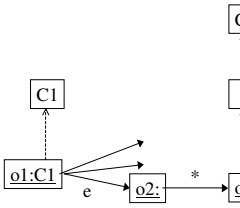


e(o1, o2) implies
class(o1) (<= .e .=>) class(o2)
in the class graph

"up, over, and down"

O(o1, o2) implies
class(o1) (<= .C .=>) class(o2)
in the class graph

Which edges to follow to C2?

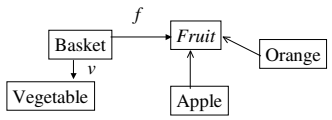


From o1 of class C1, follow edge e iff *there is some object graph* O and some o2, o3 s.t.
(1) e(o1,o2),
(2) O*(o2,o3), and
(3) class(o3) <= C2

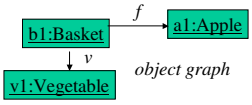
The existential quantifier "there is some object graph" represents our lack of knowledge about the rest of the object graph

from Basket to Orange

Example

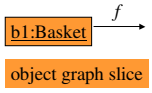


class graph



object graph

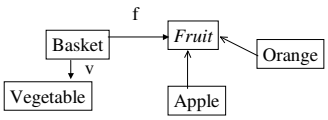
premature termination



object graph slice

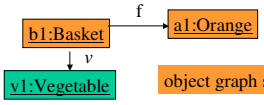
from Basket to Orange

Example



class graph

mapping:
o1 b1
o2 a1
o3 a1
e f



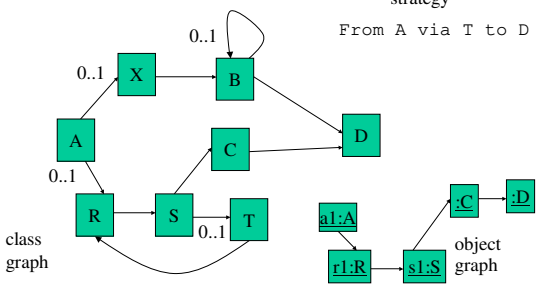
object graph slice

object graph

Example B1

strategy

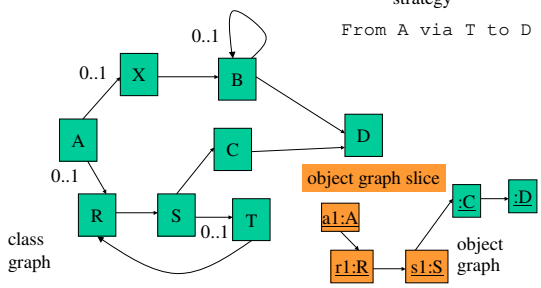
```
From A via T to D
```



Example B1

strategy

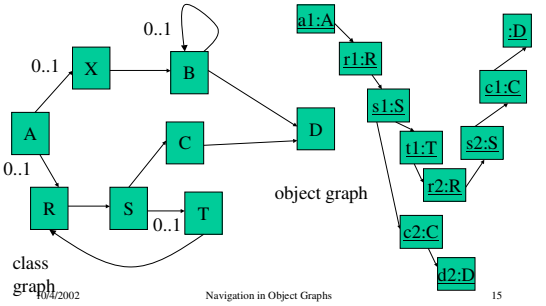
```
From A via T to D
```



Example B2

strategy

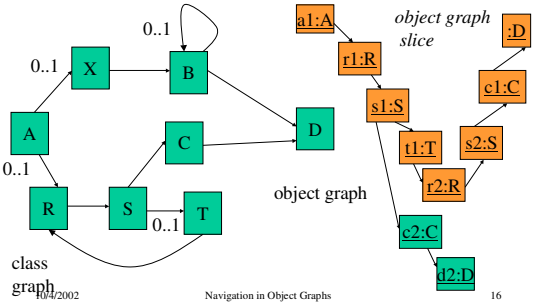
```
From A via T to D
```



Example B2

strategy

```
From A via T to D
```



Lack of Knowledge

- Objects of a given class may be very different.
- We want to go down edges without looking ahead!
- We don't want to go down edges that are guaranteed to be unsuccessful (never reaching a target object).

Object graph conforms to class graph

- The object graph O must follow the rules of the class graph: the object graph cannot contain more information than the class graph allows.

For all edges e(o1,o2) in the object graph:
e(o1, o2) implies
class(o1) (<= .e .=>) class(o2)
in the class graph

From dynamic to static characterization

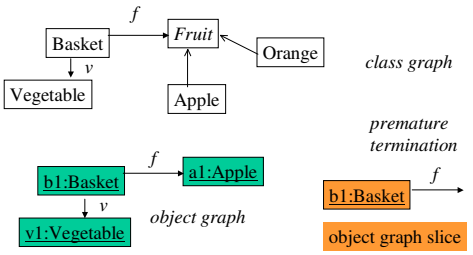
From o1 of class c1, follow edge e iff there is some object graph O and some o2, o3 s.t.
1. e(o1,o2),
2. O*(o2,o3), and
3. class(o3) <= c2

From o1 of class c1, follow edge e iff there are classes c', c'' s.t.
1. c1 <=.e.=> c'
2. c' (<=.C.=>)* c'' and
3. c'' <= c2

Let c' be class(o2), c'' be class(o3)

from Basket to Orange

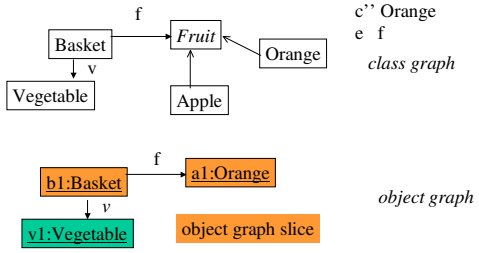
static characterization Example



from Basket to Orange

static characterization Example

mapping:
c1 Basket
c' Orange
c'' Orange
e f



from c1 to c2

Relational Formulation

From object o of class c1, to get to c2, follow edges in the set

POSS(c1,c2,o)={e | c1 <=.e.=>.(<=.C.=>)*.<= c2 }

Can easily compute these sets for every c1, c2 via transitive-closure algorithms.
POSS = abbreviation for: following these edges it is still possible to reach a c2-object for some c1-object rooted at o.

Generalizations

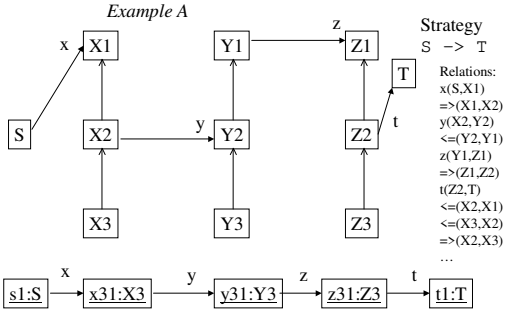
- More complex strategies
- "from c1 through c2 to c3"
  - Use "waypoint navigation"; get to a c2 object, then search for a c3 object.
- More complex strategy graphs also doable in this framework

Cyclic and acyclic object graphs

- If the object graph is acyclic, algorithm search will terminate, since every step either decreases the longest chain of links in the object graph or decreases the length of R.
- If the object graph is cyclic, then we need to mark each searched object with the state R in which it was reached (to not visit it again in the same state).

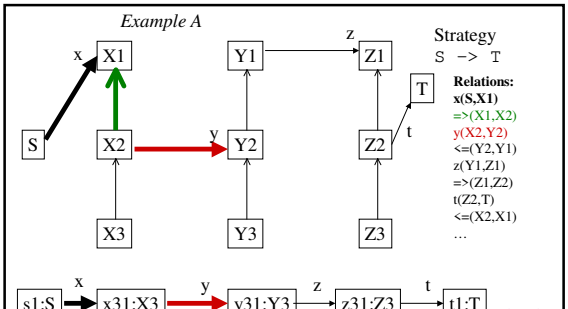
Extra Slides

Example A



go down x iff S and T are in relation: <=.x. =>.(<=.C.=>)*.<=

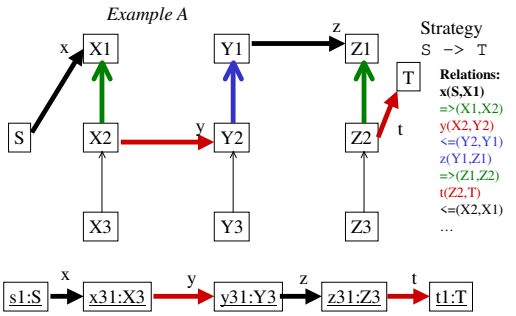
Example A



<=,=> not used

go down x iff **S <=.x. =>.(<=.C.=>)*.<= T**

Example A

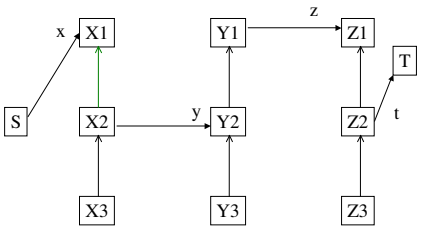


<=,=> not used

go down x iff **S <=.x. =>.(<=.C.=><=.C.=>.<=.C=>).<= T**

Just in terms of relations

Relations are sets of pairs, ordering is irrelevant.

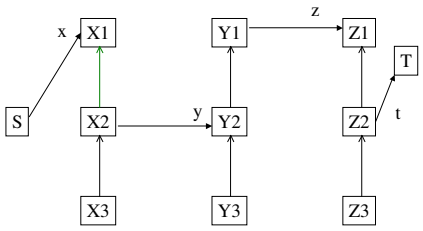


Set: {S,T,X1,X2,X3,Y1,Y2,Y3,Z1,Z2,Z3}

Write graph in terms of relations

Set: {S,T,X1,X2,X3,Y1,Y2,Y3,Z1,Z2,Z3}

Relations are sets of pairs, ordering is irrelevant.



Relations:
x,y,z,t,<=,=>
x(S,X1)
y(X2,Y2)
z(Y1,Z1)
t(Z2,T)
<=(Y2,Y1)
<=(X2,X1)
<=(X3,X2)
…
=>(X2,X3)
=>(X1,X2)
=>(Z1,Z2)
…

Set: {S,T,X1,X2,X3,Y1,Y2,Y3,Z1,Z2,Z3}

Compose relations

Relations are sets of pairs, ordering is irrelevant.

Relations: x,y,z,t,<=,=>
x(S,X1)
y(X2,Y2)
z(Y1,Z1)
t(Z2,T)
<=(Y2,Y1)
<=(X2,X1)
<=(X3,X2)
…
=>(X2,X3)
=>(X1,X2)
=>(Z1,Z2)
…

inheritance in reverse

inheritance

Are S and T in the relation: <=.x. =>.(<=.C.=><=.C.=>.<=.C=>).<=

Set: {S,T,X1,X2,X3,Y1,Y2,Y3,Z1,Z2,Z3}

Just in terms of relations

Relations are sets of pairs, ordering is irrelevant.

Relations: x,y,z,t,<=,=>
x(S,X1)
y(X2,Y2)
z(Y1,Z1)
t(Z2,T)
<=(Y2,Y1)
<=(X2,X1)
<=(X3,X2)
…
=>(X2,X3)
=>(X1,X2)
=>(Z1,Z2)
…

The order in which we consume the pairs

Relations:
x(S,X1)
=>(X1,X2)
y(X2,Y2)
<=(Y2,Y1)
z(Y1,Z1)
=>(Z1,Z2)
t(Z2,T)

go down x iff **S** <=.x. =>.(<=.C.=><=.C.=>.<=.C=>).<= **T**

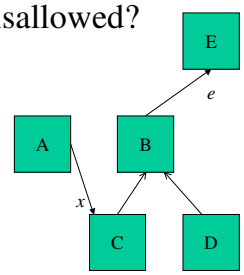
How big is the relation?

How many pairs does the relation contain in this example?

<=.x. =>.(<=.C.=><=.C.=>.<=.C=>).<=

What is disallowed?

Where can you go from A?



go down x iff S and T are in relation: <=.x. =>.(<=.C.=>)*.<=

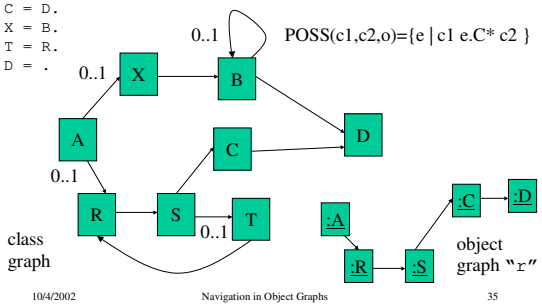
class dictionary

```
A = ["x" X] ["r" R].
B = ["b" B] D.
R = S.
S = ["t" T] C
C = D.
X = B.
T = R.
D = .
```

strategy

```
A -> T
T -> D
```

Example B



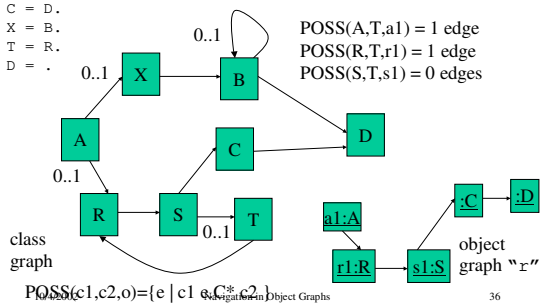
class dictionary

```
A = ["x" X] ["r" R].
B = ["b" B] D.
R = S.
S = ["t" T] C
C = D.
X = B.
T = R.
D = .
```

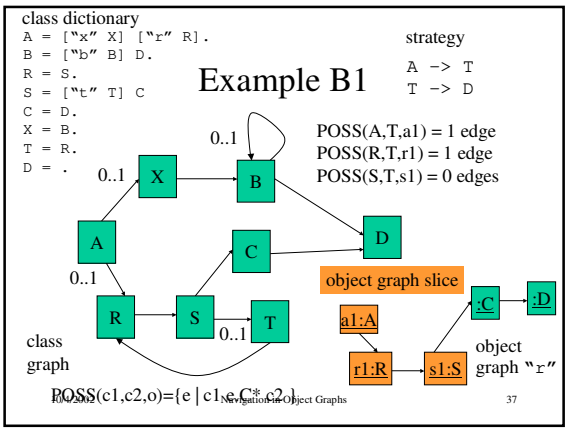
strategy

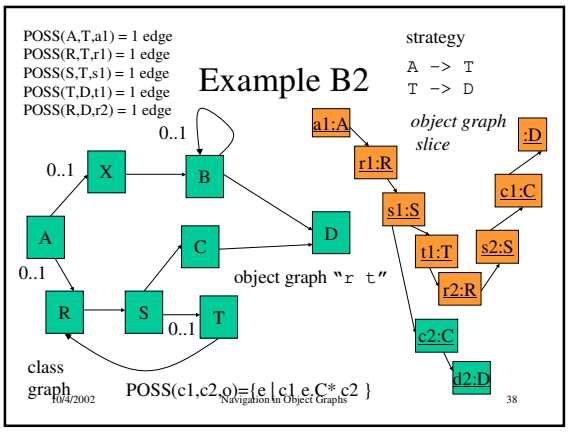
```
A -> T
T -> D
```

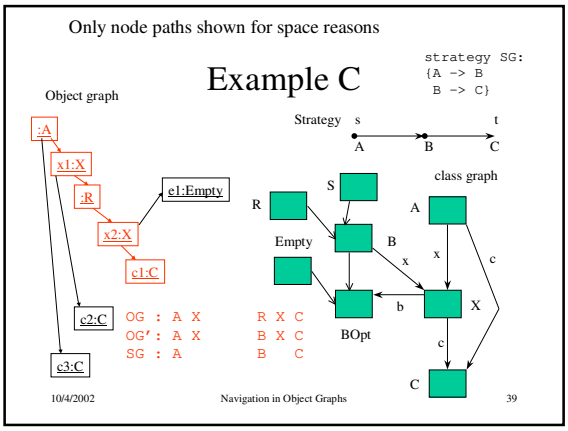
Example B1

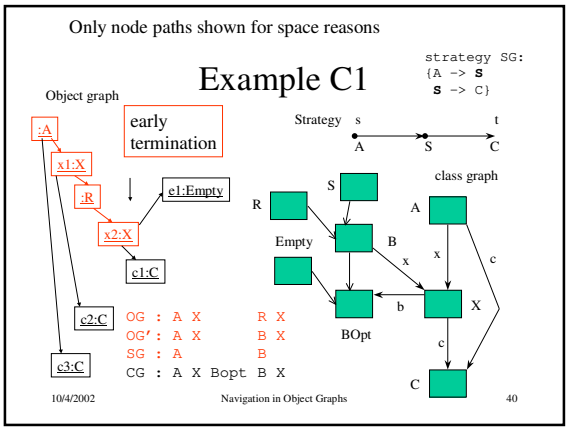


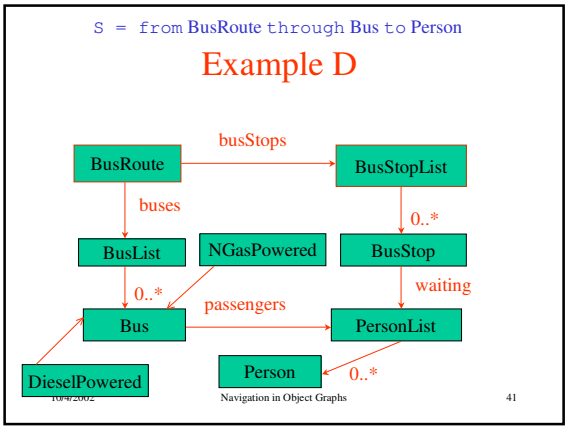
POSS(c1,c2,o)={e | c1 e.C* c2 }

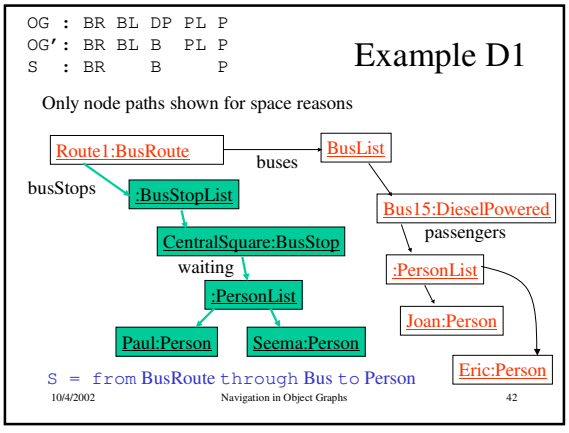

class dictionary
A = ["x" X] ["r" R].
B = ["b" B] D.
R = S.
S = ["t" T] C
C = D.
X = B.
T = R.
D = .
strategy
A -> T
T -> D
Example B1
POSS(A,T,a1) = 1 edge
POSS(R,T,r1) = 1 edge
POSS(S,T,s1) = 0 edges
object graph slice
class graph
object graph "r"
POSS(c1,c2,o)={e | c1 e C* c2 }
10/4/2002
Navigation in Object Graphs
37


POSS(A,T,a1) = 1 edge
POSS(R,T,r1) = 1 edge
POSS(S,T,s1) = 1 edge
POSS(T,D,t1) = 1 edge
POSS(R,D,r2) = 1 edge
strategy
A -> T
T -> D
Example B2
object graph slice
object graph "r t"
class graph
POSS(c1,c2,o)={e | c1 e C* c2 }
10/4/2002
Navigation in Object Graphs
38


Only node paths shown for space reasons
Example C
strategy SG:
{A -> B
B -> C}
Object graph
Strategy
class graph
OG : A X
OG': A X
SG : A
R X C
B X C
B C
BOpt
10/4/2002
Navigation in Object Graphs
39


Only node paths shown for space reasons
Example C1
strategy SG:
{A -> S
S -> C}
Object graph
early termination
Strategy
class graph
OG : A X
OG': A X
SG : A
CG : A X Bopt B X
R X
B X
B
BOpt
10/4/2002
Navigation in Object Graphs
40

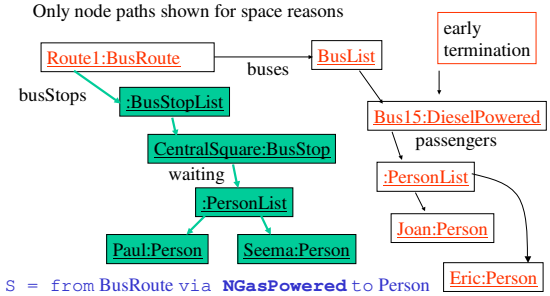

S = from BusRoute through Bus to Person
Example D
busStops
buses
passengers
waiting
0..*
Navigation in Object Graphs
41


OG : BR BL DP PL P
OG': BR BL B PL P
S : BR B P
Example D1
Only node paths shown for space reasons
buses
busStops
passengers
waiting
S = from BusRoute through Bus to Person
10/4/2002
Navigation in Object Graphs
42

OG : BR BL
OG': BR BL
S : BR

Example D2

Only node paths shown for space reasons



S = from BusRoute via **NGasPowered** to Person

from c1 bypassing {x1,x2, … ,xn} to c2

Relational Formulation

From object o of class c1, to get to c2, follow edges in the set

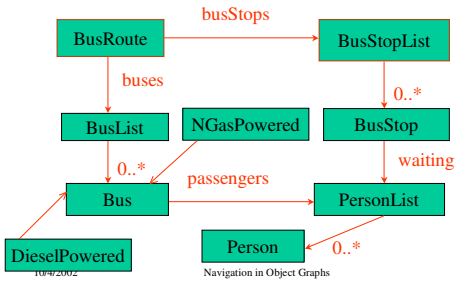
POSS(c1,c2,o)={e | c1 <=.e.=> (<=.C.=>)* <= c2 }

POSS = abbreviation for: following these edges it is still possible to reach a c2-object for some c1-object rooted at o.

Delete x1,x2, … ,xn and all edges incident with these nodes from the class graph (assumed to be different from c1, c2).

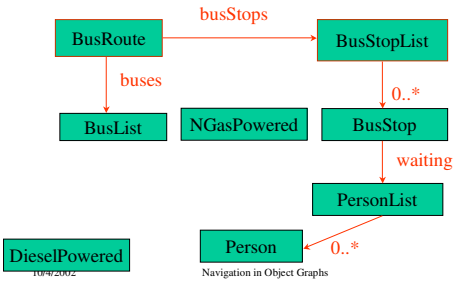
S = from BusRoute bybassing Bus to Person

Example D



S = from BusRoute bybassing Bus to Person

Example D



Conclusions

- Programming language elements are mathematical objects having precise mathematical definitions.
- Exercise in applying an abstract algorithm to concrete inputs. Mapping abstract situations to concrete situations.
- Separation of concerns is also useful for defining programming language elements
  - separate subgraph selected from
  - how the subgraph is traversed (depth-first etc.)
- In earlier works: meaning of a traversal strategy for an object graph
  - was a traversal history
  - now it is a subgraph of the object graph. A traversal history can be defined ...