Traversal Strategies

Specification and Implementation
Idea of Traversal Strategies

• Defining high-level artifact in terms of a low-level artifact without committing to details of low-level artifact in definition of high-level artifact. Low-level artifact is parameter to definition of high-level artifact.

• Exploit structure of low-level artifact.
Applications of Traversal Strategies

• Application 1
  – High-level: Adaptive program, containing strategy.
  – Low-level: Class graph

• Application 2 (see paper by Dave Mandelin on Prospector and Jungloids PLDI 2005)
  – High-level: High-level API
  – Low-level: Low-level API

Also: Dynamic call graphs!
Similar to a function definition accessing parameter generically

• \textit{High-level(Low-level)}
  – \textit{High-level} does not refer to all information in \textit{Low-level} but \textit{High-level(Low-level)} contains details of \textit{Low-level}.
Overview

• Use structure in graphs to express subgraphs and path sets in those graphs.
• Gain: writing programs in terms of strategies yields shorter and more flexible programs.
• Does not work well on dense graphs and graphs with self loops: use hierarchical approach in this case.
Graphs used

• object graphs
• class graphs
• strategy graphs
• traversal graphs
• propagation graphs = folded traversal graphs
Simplified form of theory

• Focus on class graphs with one kind of nodes and one kind of edges.
Strategy definition: embedded, positive strategies

- Given a graph $G$, a strategy graph $S$ of $G$ is any subgraph of the transitive closure of $G$ with source $s$ and target $t$.
- The transitive closure of $G=(V,E)$ is the graph $G^*=(V,E^*)$, where $E^*=\{(v,w):$ there is a path from vertex $v$ to vertex $w$ in $G\}$.
$S$ is a strategy for $G$
Discussion

• Seems strange: define a strategy for a graph but strategy is independent of graph.

• Many very different graphs can have the same strategy.

• Better: A graph G is an instance of a graph S, if S is a subgraph of the transitive closure of G. (call G: concrete graph, S: abstract graph).
$G_1$ compatible $G_2$

Compatible: connectivity of $G_2$ is in $G_1$
Theory of Strategy Graphs

- Palsberg/Xiao/Lieberherr: TOPLAS ‘95
- Palsberg/Patt-Shamir/Lieberherr: Science of Computer Programming 1997
- Lieberherr/Patt-Shamir: Dagstuhl ‘98 Workshop on Generic Programming (LNCS)
Strategy graph and base graph are directed graphs

Key concepts

- Strategy graph $S$ with source $s$ and target $t$ of a base graph $G$. $\text{Nodes}(S)$ subset $\text{Nodes}(G)$ (Embedded strategy graph).
- A path $p$ is an *expansion* of path $p'$ if $p'$ can be obtained by deleting some elements from $p$.
- $S$ defines *path set* in $G$ as follows: $\text{PathSet}_{st}(G,S)$ is the set of all $s$-$t$ paths in $G$ that are expansions of any $s$-$t$ path in $S$. 
PathSet(G, S)
Learning map

1. Graph paths labeled

2. Class graph

3. Object graph

4. Object traversal defined by concrete path set

5. Strategy graph

6. Name map constraint map

7. Algorithm 1
   in: strategy + class graph
   out: traversal graph

8. FROM-TO computation

9. Traversal graph

10. Propagation graph

11. Zig-zags short-cuts

12. Algorithm 2
   in: traversal + object graph
   out: object traversal

Correspondences
X: class path - concrete path
Y: object path - concrete path
traversal path - class path

Generalization
Other relationships

Numbers: order of coverage

11/1/2005

CSG 711
Learning map

1. Graph paths labeled
2. Class graph
3. Object graph
4. Object traversal defined by concrete path set
5. Strategy graph
6. Name map constraint map
7. Algorithm 1
   in: strategy + class graph
   out: traversal graph
8. FROM-TO computation
9. Traversal graph
10. Propagation graph
11. Zig-zags short-cuts
12. Algorithm 2
   in: traversal + object graph
   out: object traversal

correspondences
X: class path - concrete path
Y: object path - concrete path
traversal path - class path

generalization
other relationships

numbers: order of coverage

11/1/2005 Strategies
Remarks about traversals

• If object graph is cyclic, traversal is not well defined.
• Traversals are opportunistic: As long as there is a possibility for success (i.e., getting to the target), the branch is taken.
• Traversals do not look ahead. Visitors must delay action appropriately.
Strategies: traversal specification

- Strategies select class-graph paths and then derive concrete paths by applying the natural correspondence.
- Traversals are defined in terms of sets of concrete paths.
- A strategy selects class graph paths by specifying a high-level topology which spans all selected paths.
Strategies

• A strategy $SS$ is a triple $SS = (S,s,t)$, where $S = (C,D)$ is a directed unlabeled graph called the strategy graph, where $C$ is the set of strategy-graph nodes and $D$ is the set of strategy-graph edges, and $s,t \in C$ are the source and target of $SS$, respectively.
Strategies, constraint map

• Need negative constraints
• Given a class graph $G = (V, E, L)$, an element predicate $EP$ for $G$ is a predicate over $V \cup E$. Given a strategy $SS$, a function $B$ mapping each edge of $SS$ to an element predicate is called a constraint map for $SS$ and $G$. 
Strategies, constraint map

• Let $S$ be a strategy graph, let $G$ be a class graph, let $N$ be a name map and let $B$ be a constraint map for $S$ and $G$. Given a strategy-graph path $p = <a_0, a_1, ..., a_n>$, we say that a class graph path $p'$ is a satisfying expansion of $p$ with respect to $B$ under $N$ if there exist paths $p_1, ... , p_n$ such that $p' = p_1 \cdot p_2 ... p_n$ and:
Strategies, constraint map

- For all $0 < i < n + 1$, $Source(p_i) = N(a_{i-1})$ and $Target(p_i) = N(a_i)$.
- For all $0 < i < n + 1$, the interior elements of $p_i$ satisfy the element predicate $B(a_{i-1}, a_i)$. 

Strategies

• Many ways to decompose a path.
• Element constraints never apply to the ends of the subpaths.
• from A bypassing {A,B} to B
Strategies, path sets

- Let $SS = (S,s,t)$ be a strategy, let $G = (V,E,L)$ be a class graph, and let $N$ be a name map for $SS$ and $G$ and let $B$ be a constraint map for $S$ and $G$. The set of concrete paths $PathSet[SS,G,N,B]$ is $\{X(p') | p' \in P_G(N(s),N(t)) \text{ and there exists } p \in P_S(s,t) \text{ such that } p' \text{ is an expansion of } N(p) \text{ w.r.t. } B\}$. 
Strategies

• $\text{PathSet}[SS, G, N] = \text{PathSet}[SS, G, N, B_{TRUE}]$ for the constraint map $B_{TRUE}$ which maps all strategy graph edges to the trivial element predicate that is always TRUE.
Strategies

• Are used in adaptive programs.

• Adaptive programs are expressed in terms of class-valued and relation-valued variables. Class graph not known when program is written.
Learning map

1. **Learning map**

   - **Graph paths labeled**
   - **FROM-TO computation**

   - **Object graph**
   - **Class graph**

   - **Strategy graph**
   - **Traversal graph**

   - **Name map constraint map**

   - **Propagation graph**

   - **Algorithm 1**
     - in: strategy + class graph
     - out: traversal graph

   - **Algorithm 2**
     - in: traversal + object graph
     - out: object traversal

   - **Correspondences**
     - X: class path - concrete path
     - Y: object path - concrete path
     - Traversal path - class path

   - **Numbers**
     - ordered by coverage

   - **Generalization**
     - other relationships

11/1/2005
What we tried.

- Path set is represented by subgraph of class graph, called propagation graph. Propagation graph is translated into a set of methods. Works in many cases. Two important cases which do not work:
  - short-cuts
  - zig-zags
Short-cut

class graph

A

B

X

C

strategy graph with name map

A

B

C

strategy:
{A -> B
B -> C}

propagation graph

A

B

X

C

B

X

C

0..1
Short-cut

Incorrect traversal code:
class A {void t(){x.t();}}
class X {void t(){if (b!==null)b.t();c.t();}}
class B {void t(){x.t();}}
class C {void t(){}}

Correct traversal code:
class A {void t(){x.t();}}
class X {void t(){if (b!==null)b.t2();
            void t2(){if (b!==null)b.t2();c.t2();} 
        }
    class B {void t2(){x.t2();}}
class C {void t2(){}}

strategy: 
{A -> B
 B -> C}
Short-cut

strategy:
{A -> B
B -> C}

abstract representation of traversal code

class graph

traversal method t

traversal method t2

thick edges with incident nodes: traversal graph
<A C D E G> is excluded

At a D-object need to remember how we got there. Need argument for traversal methods. Represent traversal by tokens in traversal graph.
Compilation of strategies

• Two parts
  – construct graph which expresses the traversal $PathSet[SS,G,N,B]$ in a more convenient way: traversal graph $TG(SS,G,N,B)$. Represents allowed traversals as a “big” graph.
  – Generate code for traversal methods by using $TG(SS,G,N,B)$.
Compilation of strategies

• Idea of traversal graph:
  – Paths defined by from A to B can be represented by a subgraph of the class graph. Compute all edges reachable from A and from which B can be reached. Edges in intersection form graph which represents traversal.
  – Generalize to any strategies: Need to use big graph but above from A to B approach will work.
Compilation of strategies

• Idea of traversal graph:
  – traversal graph is “big brother” of propagation graph
  – is used to control traversal
  – FROM-TO computation: Find subgraph consisting of all paths from A to B in a directed graph: Fundamental algorithm for traversals
  – Traversal graph computation is FROM-TO computation.
Strategy behind Strategy

• Instead of developing a specialized algorithm to solve a specific problem, modify the data until a standard algorithm can do the work. May have implications on efficiency.

• In our case: use FROM-TO computation.
FROM-TO computation

- Problem: Find subgraph consisting of all paths from A to B in a directed graph.
  - Forward depth-first traversal from A
    - colored in red
  - Backward depth-first traversal from B
    - colored in blue
  - Select nodes and edges which are colored in both red and blue.
Traversal graph computation

Algorithm 1

• Let the strategy graph $S = (C,D)$ and let the strategy graph edges be $D = \{e_1, e_2, \ldots, e_k\}$.

• 1. Create a graph $G' = (V', E')$ by taking $k$ copies of $G$, one for each strategy graph edge. Denote the $i$th copy as $G^i = (V^i, E^i)$.

• The nodes in $V^i$ and edges in $E^i$ are denoted with superscript $i$, as in $v^i$, $e^i$, etc.
Why $k$ copies?

- Mimics using $k$ distinct traversal method names.
- Run-time traversals need enough state information.
Traversal graph computation

• Each class-graph node \( v \) corresponds to \( k \) nodes in \( V' \), denoted \( v^1, \ldots, v^k \).

• Extend \textit{Class} mapping to apply to nodes of \( G' \) by setting \( \text{Class}(v^i) = v \), where \( v^i \in V \) and \( v \in V \).
Preview of step 2

- Link the copied class graphs through temporary use of intercopy edges.
- Each strategy graph node is responsible for additional edges in the traversal graph.
- If strategy graph node has one incoming and one outgoing edge, one edge is added.
Preview of step 2

- Addition of edges from one copy to the next:

\[ A \rightarrow f \rightarrow C \]

intercopy edge

\[ f \] may be \diamond
Traversals graph computation

2.a For each strategy-graph node \( a \in C \): Let \( I = \{ei_1, \ldots ,ei_n\} \) be the strategy-graph edges incoming into \( a \), and let \( O=\{eo_1, \ldots ,eo_m\} \) be the set of strategy graph edges outgoing from \( a \). Let \( N(a)=v \in V \). Add \( n \) times \( m \) edges \( v^j \) to \( v^l \) for \( j=1, \ldots ,n \) and \( l = 1, \ldots ,m \). Call these edges intercopy edges.
Traversability graph computation

• 2.b For each node $v^i \in G'$ with an outgoing intercopy edge: Add edges $(u^i,f,v^j)$ for all $u^i$ such that $(u^i,f,v^i) \in E^i$, and for all $v^j$ which are reachable from $v^i$ through intercopy edges only.

• 2.c Remove all intercopy edges added in step 2.a.
Note: there is a bug lurking here!

• It took a while to find it. Doug Orleans found it in April 99.
  – We used traversal strategies for over two years
  – Paper was reviewed by reviewers of a top journal (Journal of the ACM)
• Solution: switch steps two and three. Why?
Preview of step 3

• Delete edges and nodes which we do not want to traverse.
Traversals graph computation

3. For each strategy-graph edge \( e_i \) = from \( a \) to \( b \): Let \( N(a) = u \) and \( N(b) = v \). Remove from the subgraph \( G^i \) all elements which do not satisfy the predicate \( B(e_i) \), with the exception of \( u^i \) and \( v^i \).

- \( V^i = \{v^i, u^i\} \cup \{w^i \mid B(e_i)(w) = \text{TRUE}\} \), and
- \( E^i = \{(w^i, l, y^i) \mid B(e_i)(w, l, y) = B(e_i)(w) = B(e_i)(y) = \text{TRUE}\} \).
Preview of step 4

• Get ready for the FROM-TO computation in the traversal graph: need a single source and target.
Traversal graph computation

- 4.a Add a node $s^*$ and an edge $(s^*, N(s)^i)$ for each edge $e_i$ outgoing from $s$ in the strategy graph, where $s$ is the source of the strategy.

- 4.b Add a node $t^*$ and an edge $(N(t)^i, t^*)$ for each edge $e_i$ incoming into $t$ in the strategy graph, where $t$ is the target of the strategy.
Traversal graph computation

• 4.c Mark all nodes and edges in $G'$ which are both reachable from $s^*$ and from which $t^*$ is reachable, and remove unmarked nodes and edges from $G'$. Call the resulting graph $G''=(V'',E'')$.

• The above is an application of the FROM-TO computation.
Traversal graph computation

5. Return the following objects:

- The graph obtained from $G''$ after removing $s^*$ and $t^*$ and all their incident edges. This is the traversal graph $TG(SS, G, N, B)$.
- The set of all nodes $v$ such that $(s^*, v)$ is an edge in $G''$. This is the start set, denotes $T_s$.
- The set of all nodes $v$ such that $(v, t^*)$ is an edge in $G''$. This is the finish set, denoted $T_f$. 
Traversal graph properties

• If $p$ is a path in the traversal graph, then under the extended $Class$ mapping, $p$ is a path in the class graph. (Roughly: traversal graph paths are class graph paths.)
Short-cut

strategy:
{ A -> B
  B -> C }

class graph

traversal method t
0..1

traversal method t2
0..1

abstract representation of traversal code

thick edges with incident nodes: traversal graph
Can now think in terms of a graph and need no longer path sets. But graph may be bigger.

**Traversal graph properties**

- Let $SS$ be a strategy, $G$ a class graph, $N$ a name map, and let $B$ be a constraint map. Let $TG = TG(SS, G, N, B)$ be the traversal graph and let $T_s$ be the start set and $T_f$ the finish set generated by algorithm 1. Then $X(Class(P_{TG}(T_s, T_f))) = PathSet[SS, G, N, B]$. (Roughly: Paths from start to finish in traversal graph are the paths selected by strategy.)
abstract representation of traversal code

strategy:
{A -> B
B -> C}

class graph

traversal method t

start set

traversal method t2

finish set

thick edges with incident nodes: traversal graph
Learning map

1. Graph paths labeled
   - FROM-TO computation
   - Correspondences
     - X: class path - concrete path
     - Y: object path - concrete path
     - Traversal path - class path

2. Object graph
3. Class graph
4. Object traversal defined by concrete path set
5. Strategy graph
6. Name map constraint map
7. Algorithm 1
   - In: strategy + class graph
   - Out: traversal graph

8. Propagation graph
9. Traversal graph
10. Zig-zags short-cuts
11. Numbers: order of coverage
12. Algorithm 2
   - In: traversal + object graph
   - Out: object traversal
Traversal methods algorithm
Algorithm 2

• Idea is to traverse an object graph while using the traversal graph as a road map.
• Maintain set of “tokens” placed on the traversal graph.
• May have several tokens: path leading to an object may be a prefix of several distinct paths in $PathSet[SS, G, N, B]$. 
Traversal method algorithm

- Traversal method $\text{Traverse}(T)$, where $T$ a set of tokens, i.e., a set of nodes in the traversal graph.

- When $\text{Traverse}(T)$ invokes visit at an object, that object is added to traversal history.
Traversing method algorithm

- *Traversal*(T) is generic: same method for all classes.
- *Traversal*(T) is initially called with the start set \( T_s \) computed by algorithm 1.
Traversal methods algorithm

• Traverse($T$), guided by traversal graph $TG$.
  
  1. define a set of traversal graph nodes $T'$ by $T'=${$v$ | $\text{Class}(v)=\text{Class}(\texttt{this})$ and there exists $u \in T$ such that $u=v$ or $(u, \varnothing, v)$ is an edge in $TG$}.
  
  2. If $T'$ is empty, return.
  
  3. Call $\texttt{this}.\text{visit()}$.  

Traversals methods algorithm

– 4. Let $Q$ be the set of labels which appear both on edges outgoing from a node in $T' \in TG$ and on edges outgoing from $this$ in the object graph. For each field name $l \in Q$, let

$$T_l = \{v \mid (u,l,v) \in TG \text{ for some } u \in T'\}.$$

– 5. Call $this.l.Traverse(T_l)$ for all $l \in Q$, ordered by “<“, the field ordering.
Short-cut

Object graph

A( <x> X( <b> B( <x> X( <c> C()) ) ) ) )

Traversal graph

strategy:
{A -> B
 B -> C}
Short-cut

Object graph

A( <x> X(<b> B(<x> X(<c> C()))))

Traversal graph

strategy:
{A -> B
B -> C}

start set

finish set

Used for token set and currently active object
Short-cut

Object graph

A(
  <x> X(
    <b> B(
      <x> X(
        <c> C())
      ))
    ))

Traversal graph

strategy:
{A -> B
 B -> C}

Used for token set and currently active object
Short-cut

strategy:
\{ A \rightarrow B \\
B \rightarrow C \}\n
Object graph

A(  
<x> X(  
<b> B(  
<x> X(  
<c> C())  
<c> C())  

<.<> X(  
<b> B(  
<x> X(  
<c> C())  
<c> C())  

Used for token set and currently active object
Short-cut

Strategy:
{A -> B
  B -> C}

Object graph

A(
  <x> X(
    <b> B(
      <x> X(
        <c> C())
        <c> C())
      <x> X(
        <c> C())
    <b> B(
      <x> X(
        <c> C())
      <x> X(
        <c> C())
    <x> X(
      <c> C())
  <b> B(
    <x> X(
      <c> C())
  <c> C())

Traversal graph

Used for token set and currently active object

start set

A

B

X

X

b

b

0..1

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>

<..>
Short-cut

Object graph

A(
  <x> X(
    <b> B(
      <x> X(
        <c> C())
    ))
  ))

Traversal graph

A

start set

B

0..1

C

finish set

strategy:
{A -> B
 B -> C}

Used for token set and currently active object
Short-cut

A(
<x> X( ☷)
<b> B(
<x> X(  
<c> C(()))
<c> C()))

<新加内容> Used for token set and currently active object

After going back to X
Traversing algorithm property

• Let $O$ be an object tree and let $o$ be an object in $O$. Suppose that the Traverse methods are guided by a traversal graph $TG$ with finish set $T_f$. Let $H(o,T)$ be the sequence of objects which invoke visit while $o.Traverse(T)$ is active, where $T$ is a set of nodes in $TG$. Then traversing $O$ from $o$ guided by $X(P_{TG}(T,T_f))$ produces $H(o,T)$. 
Zig-zags

strategy graph with name map

traversal graph = strategy graph (essentially)

class graph

<A C D E G> is excluded
shorter: \{A\rightarrow D \ D\rightarrow F \ F\rightarrow G \ A\rightarrow B \ B\rightarrow E \ E\rightarrow G\}
Zig-zags

strategy graph with name map

traversal graph = strategy graph (essentially)

class graph

object tree

11/1/2005

Strategies
Zig-zags

strategy graph with name map

 traversal graph = strategy graph (essentially)

A( B( D( E( G()) F( G()))))

object tree

class graph

11/1/2005

Strategies
Zig-zags

strategy graph
with name map

A -> B -> C -> D -> E -> F -> G

traversal graph = strategy graph (essentially)

strategy graph with name map

class graph

object tree

$\langle A, C, D, E, G \rangle$ is excluded

A( B( D( E( G()) F( G()))))

A B C D E F G

class graph

A B C D E F G

11/1/2005 Strategies 74
Zig-zags

strategy graph with name map

traversal graph = strategy graph (essentially)

<A C D E G> is excluded
Zig-zags

strategy graph with name map

traversal graph = strategy graph (essentially)

<A C D E G> is excluded

object tree

A(
  B(
    D(
      E(
        G())
        F(
        G())))
    C(
      D(
        E(
        G())
        F(
        G())))
  C(
    D(
      E(
      G()))
      F(
      G()))))

class graph
Zig-zags

strategy graph with name map

object tree

class graph

<traversal graph = strategy graph (essentially)>

<A C D E G> is excluded
Zig-zags

strategy graph with name map

traversal graph = strategy graph (essentially)

<A C D E G> is excluded

object tree

class graph
Zig-zags

strategy graph with name map

traversal graph = strategy graph (essentially)

< A C D E G > is excluded

object tree

class graph

11/1/2005

11/1/2005 Strategies
Main Theorem

• Let $SS$ be a strategy, let $G$ be a class graph, let $N$ be a name map, and let $B$ be a constraint map. Let $TG$ be the traversal graph generated by Algorithm 1, and let $T_s$ and $T_f$ be the start and finish sets, respectively.
Main Theorem (cont.)

- Let $O$ be an object tree and let $o$ be an object in $O$. Let $H$ be the sequence of nodes visited when $o.Traverse$ is called with argument $T_s$, guided by $TG$. Then traversing $O$ from $o$ guided by $PathSet[SS,G,N,B]$ produces $H$. 
Complexity of algorithm

• Algorithm 1: All steps run in time linear in the size of their input and output. Size of traversal graph: \( O(|S|^2 |G| d_0) \) where \( d_0 \) is the maximal number of edges outgoing from a node in the class graph.

• Algorithm 2: How many tokens? Size of argument \( T \) is bounded by the number of edges in strategy graph.
Simplifications of algorithm

• If no short-cuts and zig-zags, can use propagation graph. No need for traversal graph. Faster traversal at run-time.

• Presence of short-cuts and zig-zags can be checked efficiently (compositional consistency).

• See chapter 15 of AP book.
Extensions

- Multiple sources
- Multiple targets
- Intersection of traversals
Summary

• Abstract model behind strategy graphs.
• How to implement strategy graphs.
• How to apply: Precise meaning of strategies; how to write traversals manually (watch for short-cuts and zig-zags).
Where to get more information

• Paper with Boaz-Patt Shamir (strategies.ps in my FTP directory)
• Implementation of Demeter/Java and AP Library shows you how algorithms are implemented in Demeter/Java (and Java). See Demeter/Java resources page.
• Chapter 15 of AP book.
Feedback

• Send email to lieber@ccs.neu.edu.