MILCOM 2015
LEVERAGING TECHNOLOGY—THE JOINT IMPERATIVE
OCTOBER 26–28, 2015·TAMPA, FL

Tutorial: Practical Program Analysis for Discovering Android Malware

Module 1: Graph Paradigm for Program Analysis

Suresh Kothari – kothari@iastate.edu
Benjamin Holland – bholland@iastate.edu

Acknowledgment: co-workers and students
DARPA contracts FA8750- 12-2-0126 & FA8750-15-2-0080

ensoft
conquer complexity

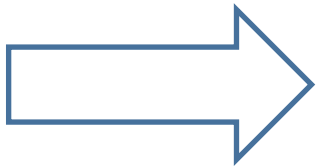
IOWA STATE UNIVERSITY

Agenda

- Economics, security, and safety of software systems
- DARPA software cybersecurity programs
- Fundamental challenges and technology shortcomings
- New vision:
  - Models to reason about hard problems of software
  - Automation to amplify human intelligence
  - The XCSG graph schema and the Atlas Platform
- Lab 1

Extreme change

Figure 1. IBM System/360 Model 40 Data Processing System



1960: a computer cost 25 million dollars, and a programmer's annual salary 6 thousand dollars. 64 KB operating system footprint.

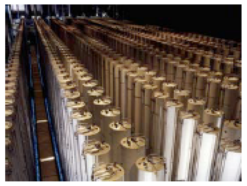
Today: a computer costs one thousand dollars, and a programmer's annual salary 60 thousand dollars. 8.5 GB Windows 8 footprint.

Relative to computer, programmer became 250,000 times more expensive & the software grew 125,000 times larger.

Many targets for software sabotage

SCADA Systems

Source: Laing O'Rourke



Source: Dept. of Energy

Medical Devices

Source: www.seekinglpha.com

Source: www.medtechbusiness.com

Computer Peripherals

Source: HP

Source: www.buy.com

Source: www.bagitech.com

Communication Devices

Source: NASA

Source: www.engadget.com

Source: GD C4S

Vehicles

Source: www.militaryaerospace.com

Source: www.naval-technology.com

Source: www.motortrend.com

Software-driven and Internet-enabled communication, control, and financial systems

Catastrophic consequences

Software error caused the 500 million dollar rocket to explode

Fighter Jets F4 to F35: 80% functionality transitioned from hardware to software

F35 Jet: 24 million lines of code

Agenda

- Economics, security, and safety of software systems
- DARPA software cybersecurity programs
- Fundamental challenges and technology shortcomings
- New vision:
  - Models to reason about hard problems of software
  - Automation to amplify human intelligence
  - The XCSG graph schema and the Atlas Platform
- Lab 1

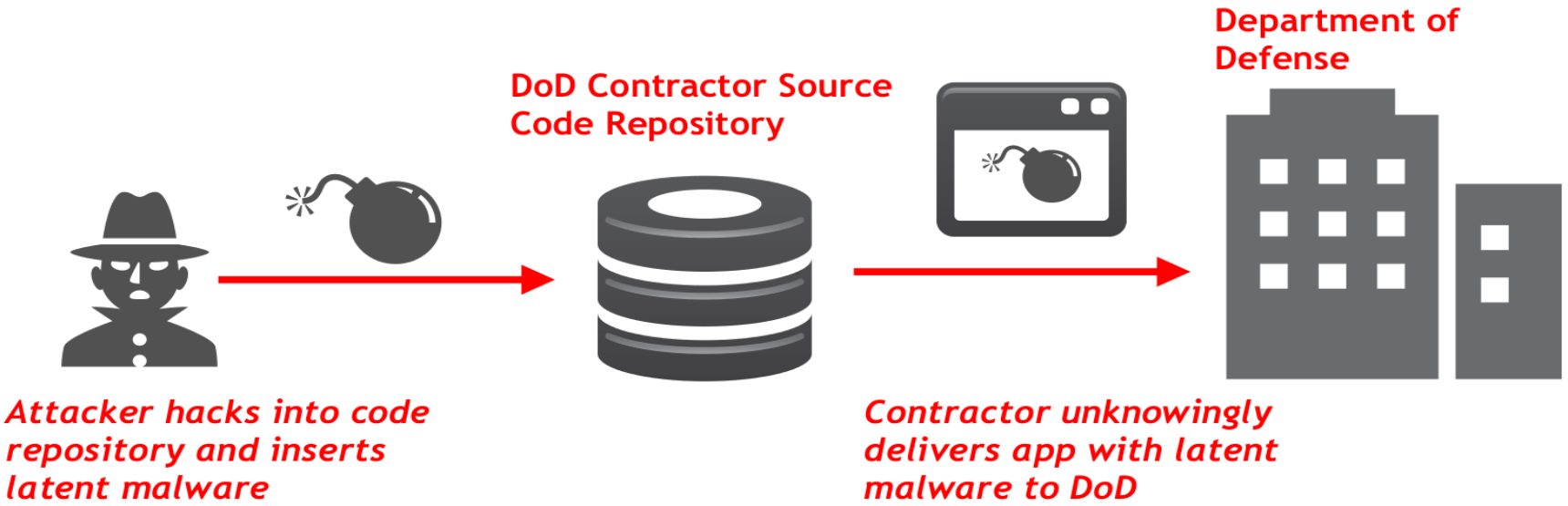
DARPA APAC Project

- **Automated Program Analysis for Cybersecurity (APAC)**: Detect sophisticated vulnerabilities in Android apps
- **Requirement**: Analyze Java code, the resource and GUI files, and the Android APIs used by the app
- **Performance Goal**: Scalable and accurate detection of Android malware

DARPA STAC Project

- **Space/Time Analysis for Cybersecurity (STAC)**: Detect algorithmic complexity (AC) and side channel (SC) vulnerabilities by analyzing variations in space-time complexities along different execution paths
- **Requirement**: Analyze Java byte code taking into account the library calls
- **Performance Goal**: Scalable and accurate detection of AC and SC vulnerabilities

DoD Scenario

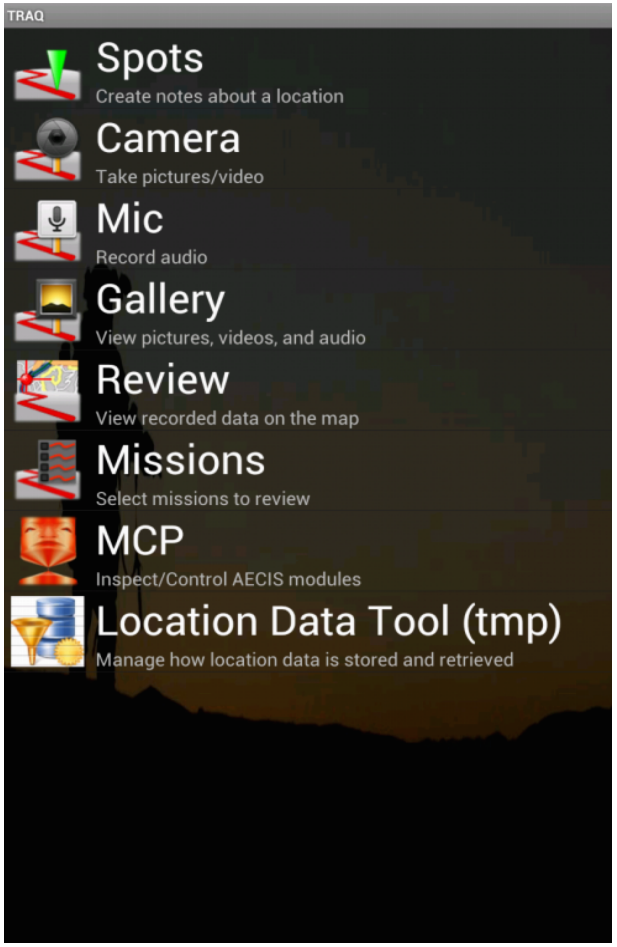


Hardened devices, untrusted contractors, expert adversaries

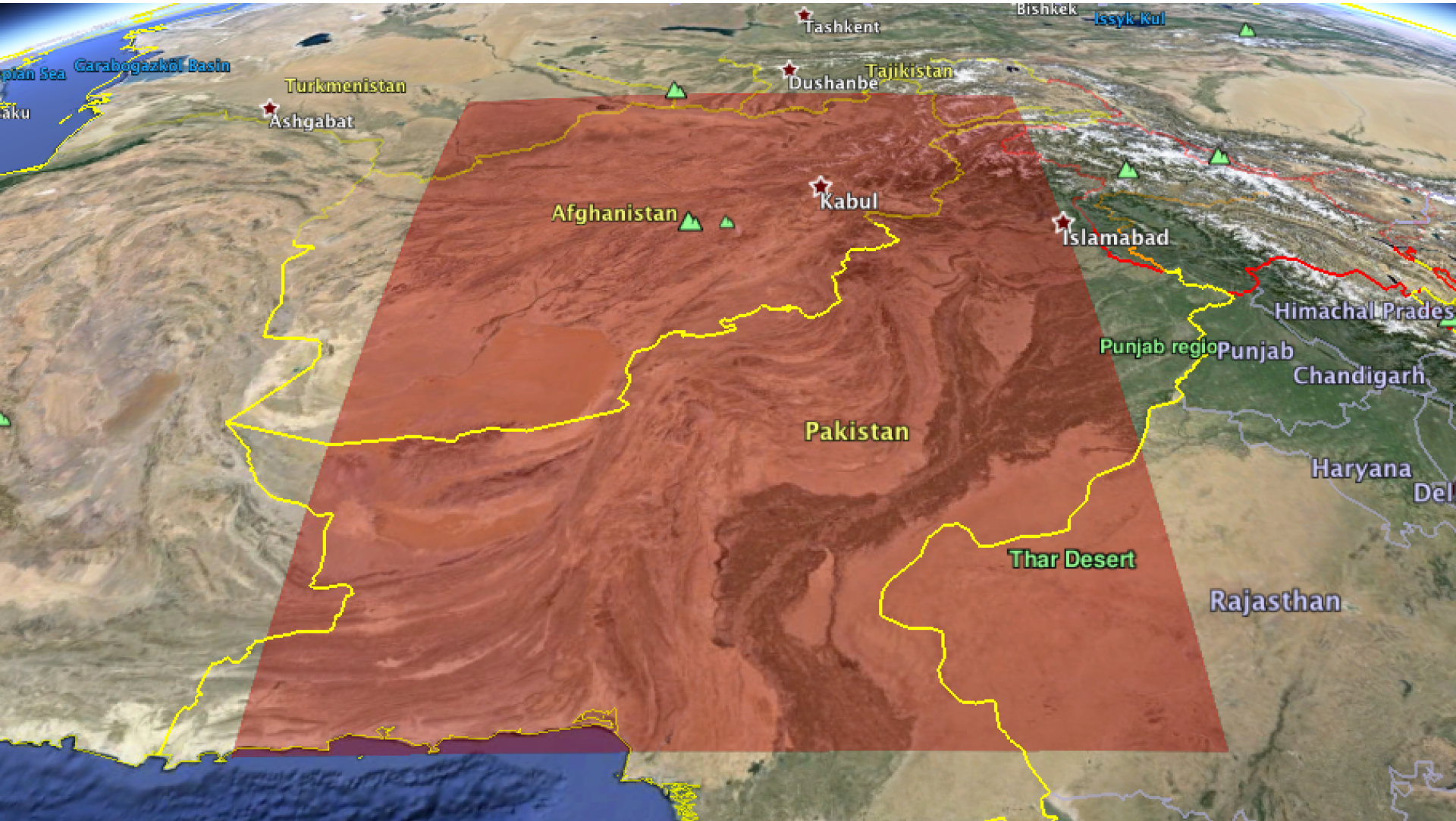
TRAQ (Transformative Apps)

- 55K lines of code
- Data gathering and relaying tool for military
  - Strategic mission planning/review
  - Audio and video recording
  - Geo-tagged camera snapshots
  - Real-time map updates based on GPS

- **Challenge:** Can we detect malicious code that might be in this application?



The malware trigger region



Defense Malware

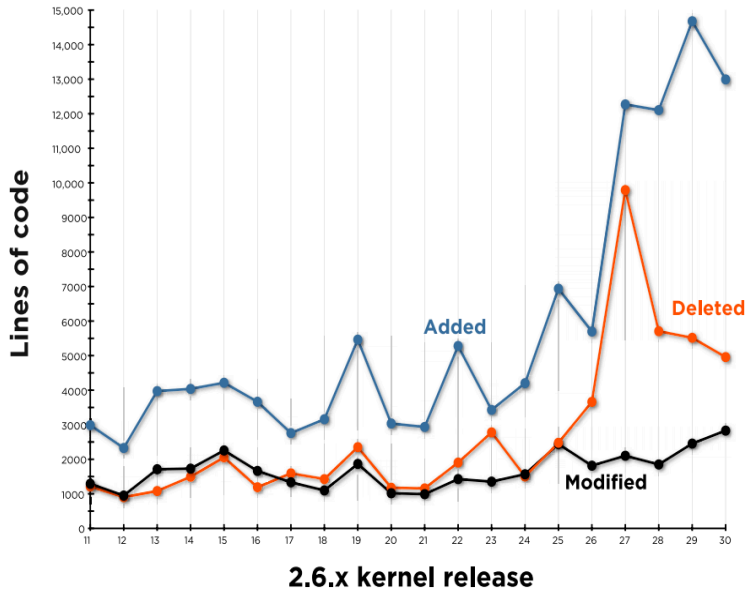
```
@Override
public void onLocationChanged(Location tmpLoc) {
   location = tmpLoc;
   double latitude = location.getLatitude();
   double longitude = location.getLongitude();
   if((longitude >= 62.45 && longitude <= 73.10) &&
        (latitude >= 25.14 && latitude <= 37.88)) {
        location.setLongitude(location.getLongitude() + 9.252);
        location.setLatitude(location.getLatitude() + 5.173);
   }
      ...
}
```

- Malware corrupts GPS when in Afghanistan/Pakistan!
- Small malicious code in a large defense app
- Malware triggered only in certain locations

Agenda

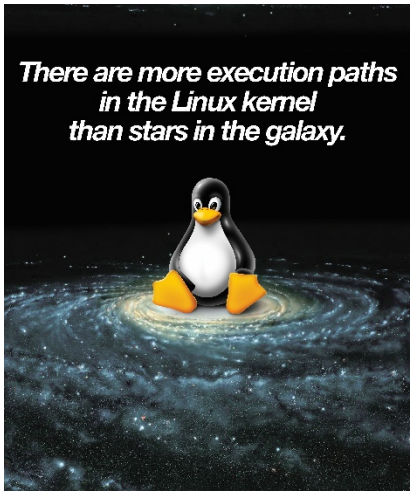
- Economics, security, and safety of software systems
- DARPA software cybersecurity programs
- Fundamental challenges and technology shortcomings
- New vision:
  - Models to reason about hard problems of software
  - Automation to amplify human intelligence
  - The XCSG graph schema and the Atlas Platform
- Lab 1

Challenge of humongous software



Linux 1.0.0 – 176,250 LOC
Linux 3.16 – 17 M LOC

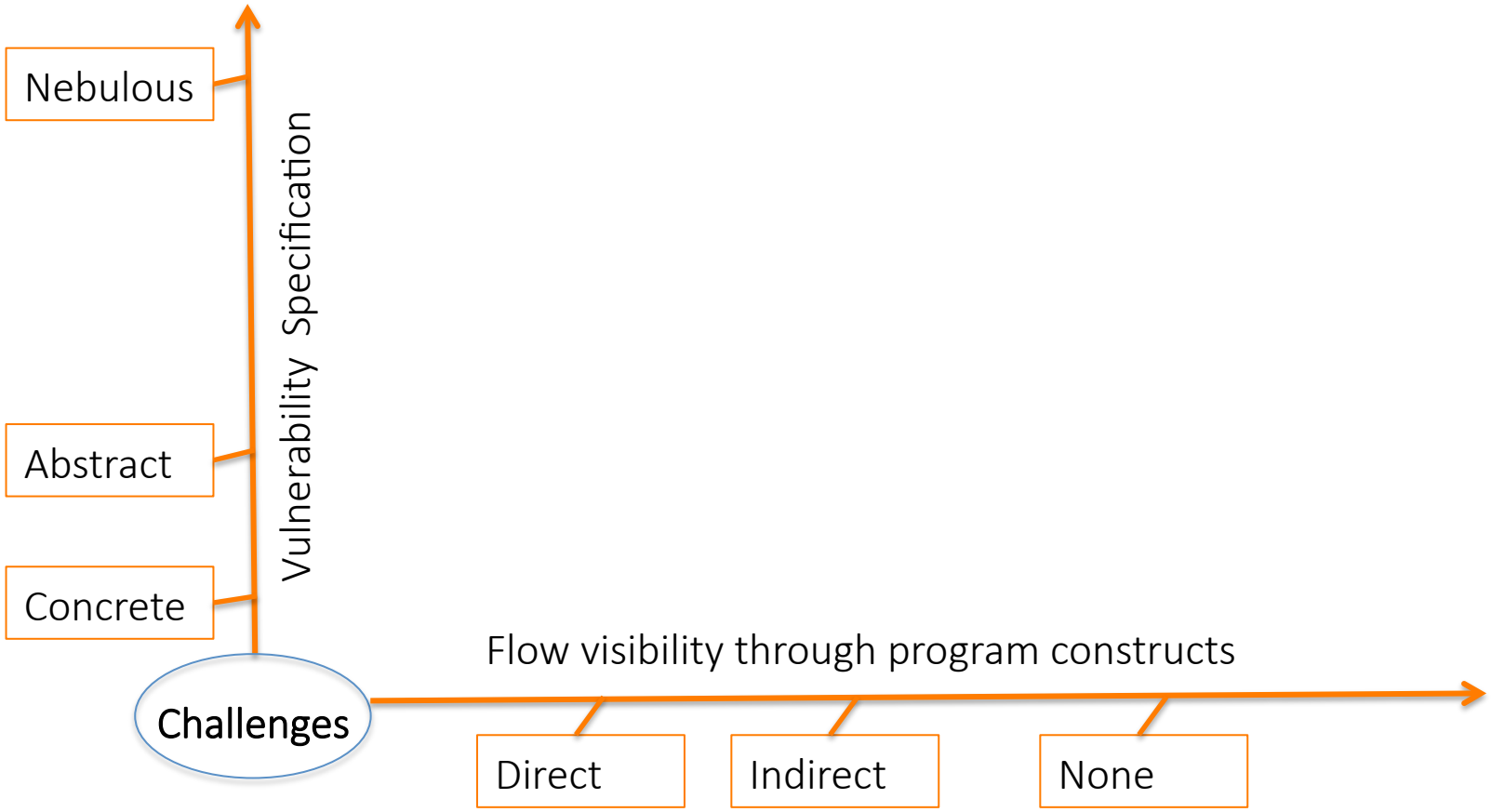
136 feet stack of paper!



Added LOC/day	Deleted LOC/day	Modified LOC/day
12993	4958	2830

Finding a needle in a haystack,
not knowing what the needle looks like…

Challenges for high accuracy



An experiment with 55 antivirus programs

CVE-2012-4681: "Allows remote attackers to execute arbitrary code via a crafted applet that bypasses SecurityManager restrictions…"

A classroom experiment two years after the CVE:

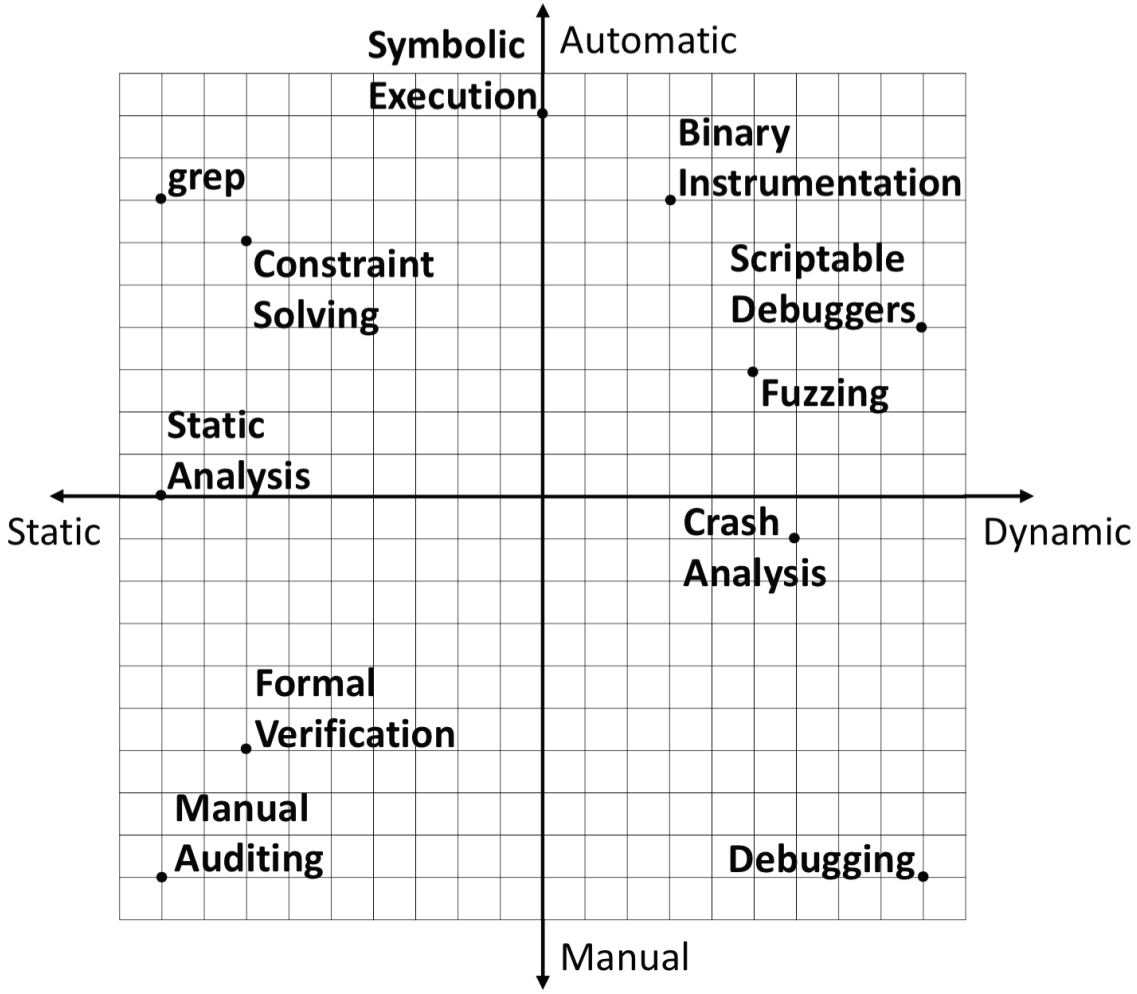
Sample	Notes	Positive detection
Original Sample	http://pastie.org/4594319	30/55
Technique A	Changed Class/Method names	28/55
Techniques A-B	Obfuscate strings	16/55
Techniques A-C	Change Control Flow	16/55
Techniques A-D	Reflective invocations (on sensitive APIs)	3/55
Techniques A-E	Simple XOR Packer	0/55

Pitfalls of superficial characterizations

Superficial Characterization: Vulnerabilities characterized by code snippets or by simplistic categorizations

- Superficial detection techniques easily duped by small code changes. Intrinsic problem remains unsolved
- Knowledge of "hardness" not distilled from problem lists (e.g. Common Weakness Enumeration)
- Simplistic categorizations lump together problems with Very dissimilar "hardness"
- Lack of deep generalizations to solve a broad class of problems sharing the same core

A spectrum of program analysis techniques



Source: Contemporary Automatic Program Analysis, Julian Cohen, Blackhat 2014

Problems with current technologies

- **Reactive Detection:** not acceptable because the very first occurrence of an attack can be catastrophic
- **Testing**: neither targeted nor random testing can work because of ill-defined targets and obscure triggers involving intricate sequences of low-probability events
- **Dynamic Analysis**: cannot be complete due to the exponential explosion of execution paths
- **Static Analysis**: encounters intractable algorithmic problems for achieving scalability and accuracy

Agenda

- Economics, security, and safety of software systems
- DARPA software cybersecurity programs
- Fundamental challenges and technology shortcomings
- New vision:
  - Models to reason about hard problems of software
  - Automation to amplify human intelligence
  - The XCSG graph schema and the Atlas Platform
- Lab 1

A new vision to solve software problems

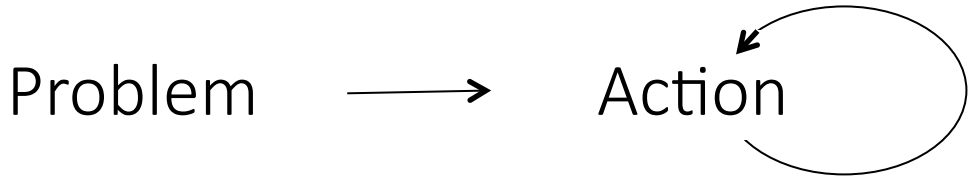
Goal: Solving problems of large software with unprecedented accuracy and substantial reduction of labor

- Design models to:
  - Eliminate noise and encapsulate the essence of the problem
  - Reveal inner workings of software
  - Reduce the programming burden to analyze, validate, and transform software
- Automate to amplify human intelligence to:
  - Understand and visualize software
  - Generate and refine hypothesis
  - Enable effective human-guided problem solving

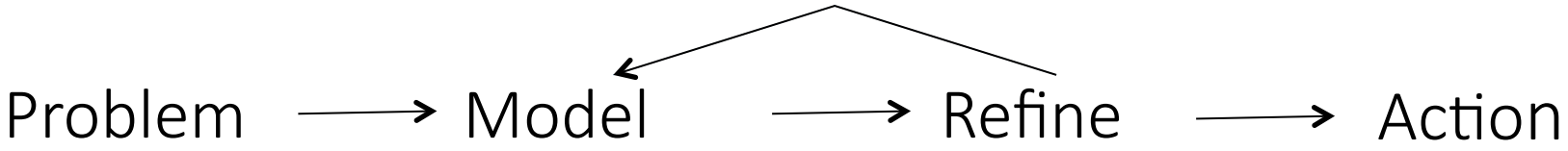
Effective Problem Solving

Act in haste & incur waste

Short-circuited



Effective



Effective problem solving with *models*

Problem 1: John is 3 years older than Jill. Together their ages add up to their mother's age. The mother is 45 years old. How old are John and Jill?

Problem 2: A water tank holds 3 gallons of water more than another water tank. Together the two water tanks can hold 45 gallons of water. What is the capacity of each water tank?

Are these different problems?

$$X - Y = 3$$
$$X + Y = 45$$

Without knowledge of algebra, the two problems look dissimilar, and the solution a wasteful trial-and-error

A brilliant "model" yields insights impossible to attain by any other method - captures essence of the problem, achieves deep generalization, enables effective problem solving

Debugging with a model

```
1. x = 2;
2. y = 3;
3. z = 7;
4. a = x + y;
5. b = x + z;
6. a = 2 * x;
7. c = y + x + z;
8. t = a + b;
9. Print t;
```

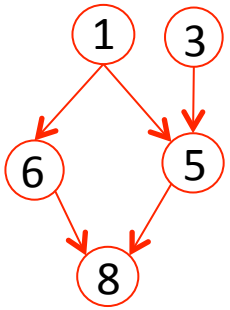
Problem 1: Debug the left program given the printed value is incorrect.

Problem 2: Debug the right program given the printed value is incorrect.

```
1. e = 2;
2. f = 3;
3. g = 7;
4. h = e + f;
5. i = e + g;
6. h = 2 * e;
7. j = f + e + g;
8. k = h + i;
9. Print k;
```

Are these different problems?

NO



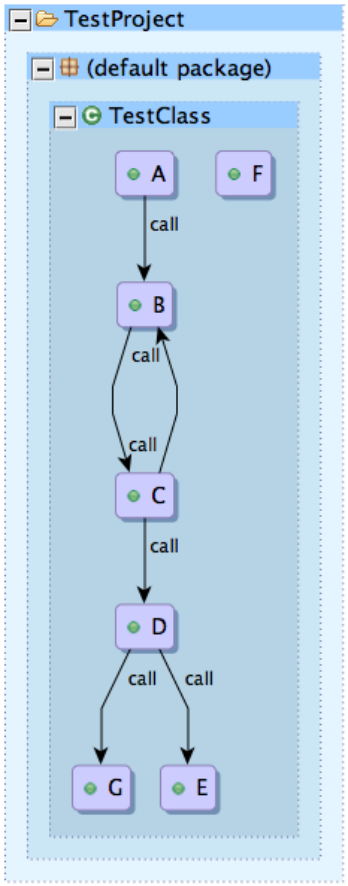
Graphs are effective models of software programs

A platform approach to build tools

- Types of graphs and the computations differ depending on the problem domain
- Platform + Toolboxes
  - Platform provides base graphs for program relationships, and APIs to traverse and transform the graphs
  - Toolbox provides graphs for domain-specific entities (e.g. XML files in Android) and domain-specific reasoning modules (e.g. a reasoning module to detect confidentiality leaks)

The Atlas Platform

```
1  public class TestClass {
2
3      public void A() {
4          B();
5      }
6
7      public void B() {
8          C();
9      }
10
11     public void C() {
12         B();
13         D();
14     }
15
16     public void D() {
17         G();
18         E();
19     }
20
21     public void E() {
22
23     }
24
25     public void F() {
26
27     }
28
29     public void G(){
30
31     }
32
33 }
```



Code

Learning Atlas: http://ensoftatlas.com/wiki/Learning_Atlas
Download: http://www.ensoftcorp.com/atlas/

Model

Mathematical Foundation

- Sets and Relations: as the simplest and the most powerful language of abstraction
- Visualize program artifacts and relations as nodes and edges in a graph
- Express, Analyze, and Transform structural and behavioral knowledge of software using the language of sets and relations
- Use graph algorithms to reason about software
- Benefits: Unprecedented advances in speed, accuracy, and accountability in developing, validating, and managing software

Atlas Features

- eXtended Common Software Graph (XCSG) a unified schema to cover multiple programming languages
- Converters to transform software to XCSG graph
- Database and query language for mining and transforming XCSG graphs
- Visualization of XCSG graphs
- An Interpreter for interactive reasoning
- Query-embedded programming for writing powerful analyses
- Eclipse IDE integration

XCSG Schema

- Key Benefits:
  A. Enables composition of analyses.
  B. Reduces significantly the human effort to create program comprehension, modeling, and reverse engineering tools.
  C. Enables unified program analyzers for multiple source languages.
- Enabling Mechanism:
  A. XCSG-compliant inputs and outputs enable composition of analyses.
  B. By lifting code to high-level graph abstractions, XCSG reduces the human efforts to write code analyzers, transformers, and visualizers.
  C. Serves as a high-level common intermediate language for developing unified analyzers
  D. XCSG incorporates control, data, type and other relationships

Examples of XCSG Analyzers

- Call Graph Analyzer:

```
public Q cg(Q function){
    return edges(XCSG.Call).forward(function)
}
```

- Reverse Call Graph Analyzer:

```
public Q rcg(Q function){
    return edges(XCSG.Call).reverse(function)
}
```

Powerful analyses such as *symbolic evaluation* or *type inference for resolving dynamic dispatches* can be coded in days as opposed months.

Composable Analyses

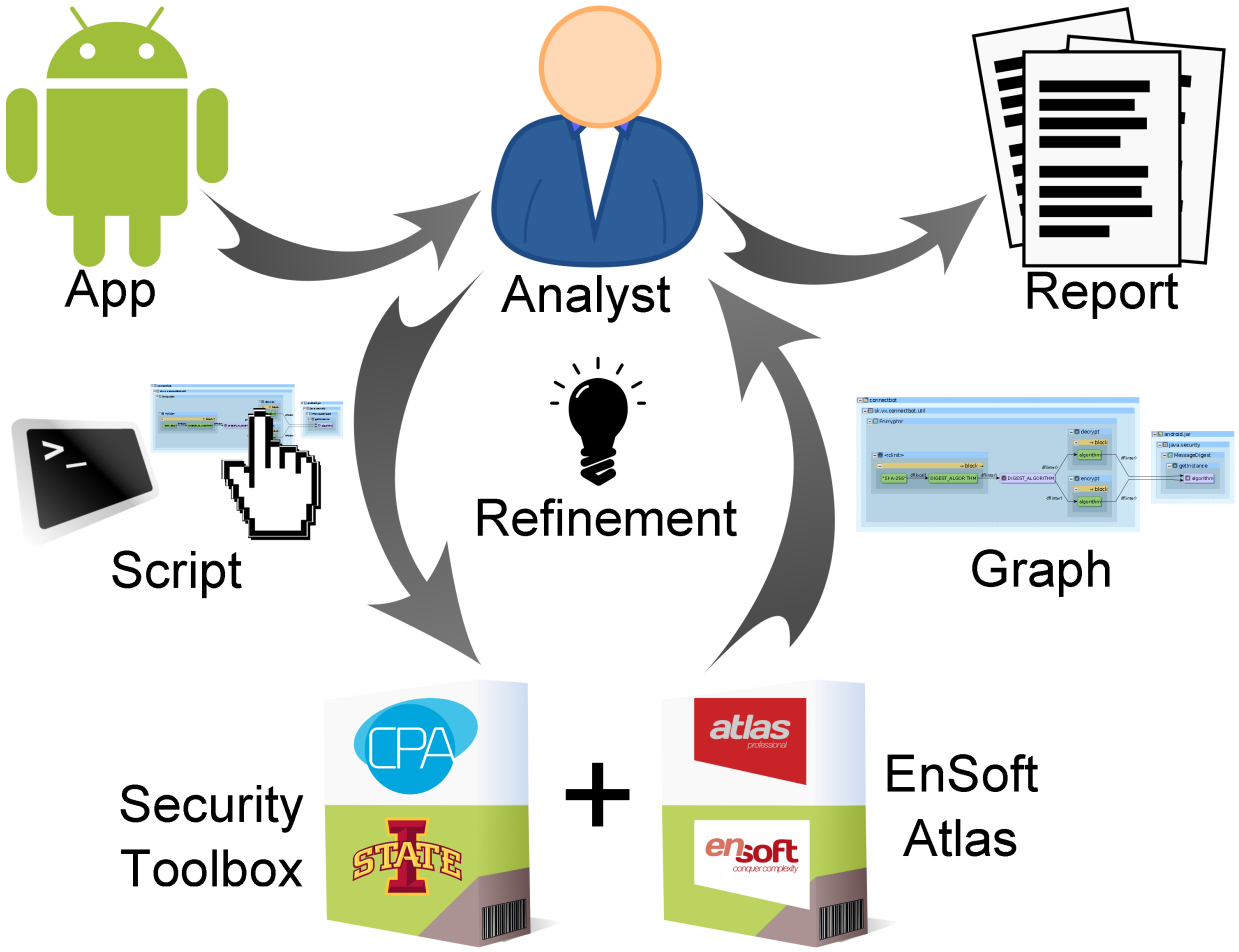
- Goal: Composition of analyses with different sensitivities (object, context, flow, and path); should achieve appropriate accuracy and scale to large software.
- Enabling Technology: eXtensible Common Software Graph (XCSG) – analyses can be composed by having their inputs and outputs adhere to the XCSG schema
- Example:

```
public Q rcg-cg(Q function){
    return rcg(cg(function))
}
```

Domain-specific toolboxes with Atlas

- Atlas enables experiments to find suitable models and model-based algorithms to solve hard software problems
- Applications: (a) validating software for cybersecurity and reliability, (b) auditing modularity of design (c) transforming software from sequential to parallel
- Solution Template:
  1. Use Atlas to model software
  2. Atlas API-programming to refine and extend the model
  3. Apply powerful algorithms and perform interactive experiments to solve difficult problems

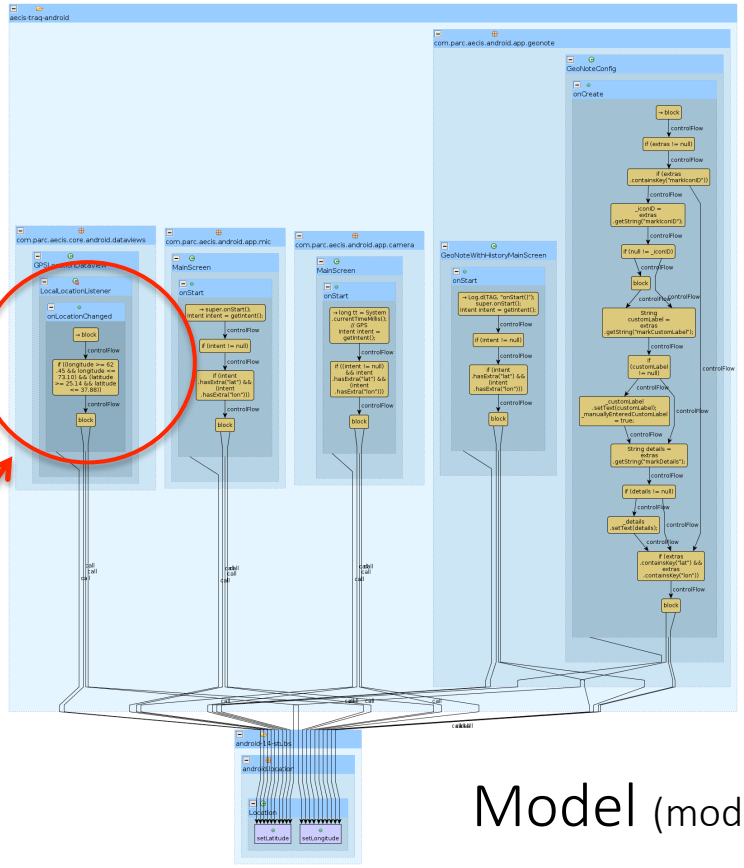
DARPA APAC Project: Atlas + Security Toolbox



DARPA Challenge App Engagements

- Phase 1
  - 77 Android applications developed by the Red teams
  - 62 contained novel malware able to evade current automatic detection techniques
  - Our performance: Correctly classified 66 (85.7%) apps, found unintended malicious behaviors in 6 (7.8%), missed planted malware in 5 (6.5%)
  - ISU-EnSoft team was the top performer among the six Blue teams
- Phase 2: Ranked among top 3 performers

Defense malware revealed by model



Malware

```
@Override
public void onLocationChanged(Location tmpLoc) {
   location = tmpLoc;
   double latitude = location.getLatitude();
   double longitude = location.getLongitude();
   if((longitude >= 62.45 && longitude <= 73.10) &&
      (latitude >= 25.14 && latitude <= 37.88)) {
        location.setLongitude(location.getLongitude() + 9.252);
        location.setLatitude(location.getLatitude() + 5.173);
   }
                ...
}
```

~10 Lines of Code

Model (modifications of GPS coordinates under conditions)

Agenda

- Economics, security, and safety of software systems
- DARPA software cybersecurity programs
- A parade of vulnerabilities and malware
- Fundamental challenges and technology shortcomings
- New vision:
  - Models to reason about hard problems of software
  - Automation to amplify human intelligence
  - The XCSG graph schema and the Atlas Platform
- Lab 1

BREAK

- Bathroom/coffee/etc.
- Introduce yourself to your neighbor(s)
  - What's your name?
  - What program analysis challenges are you interested in solving?
- What questions do you have for us?

Lab 1: Graph Queries

- Goals
  - Successfully write graph traversals
    - Example: Forward, Reverse, Set Operations
  - Understand basics of Atlas program graphs
    - Structure, Control Flow, Data Flow
    - Attributes, Tags
  - Explore query composability/reusability
  - Locate additional resources
  - Start auditing a large application

Interactive reasoning with graph models

- eXtensible Common Software Graph (XCSG) schema
  - Heterogeneous, attributed, directed graph data structure as an abstraction to represent the essential aspects of the program's syntax and semantics (structure, control flow, and data flow), which are required to reason about software.
  - Expressive query language for users to write composable analyzers
  - Results computed in the form of subgraphs defined by the query, which can be visualized or used as input in to other queries

Thought Experiment - Given the following program what graph(s) could we produce?

```
public class MyClass {
   public static void A() {
      B();
   }
   public static void B() {
      C();
   }
   public static void C() {
      B();
      D();
   }
   public static void D() {
      G();
      E();
   }
   public static void E() {}
   public static void F() {}
   public static void G() {}
}
```

Control Flow (summary)

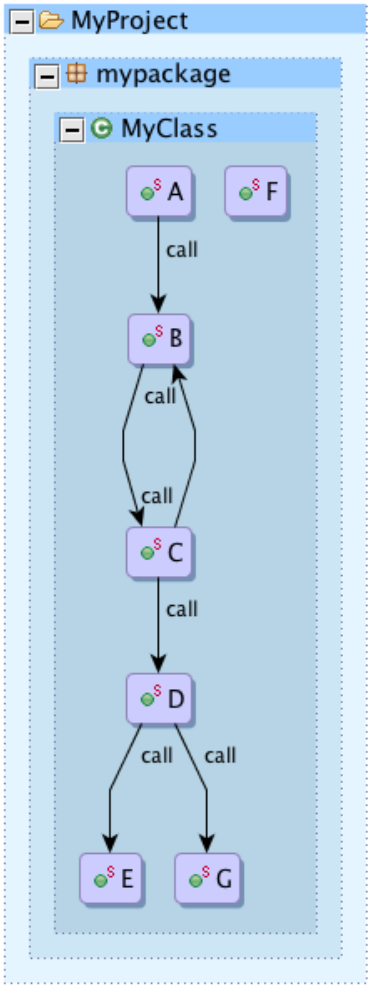
A *calls* B
B *calls* C
C *calls* B
C *calls* D
D *calls* G
D *calls* E

Structure

MyProject *contains* mypackage
mypackage *contains* MyClass
MyClass *contains* methods:
A, B, C, D, E, F, G

Data Flow?

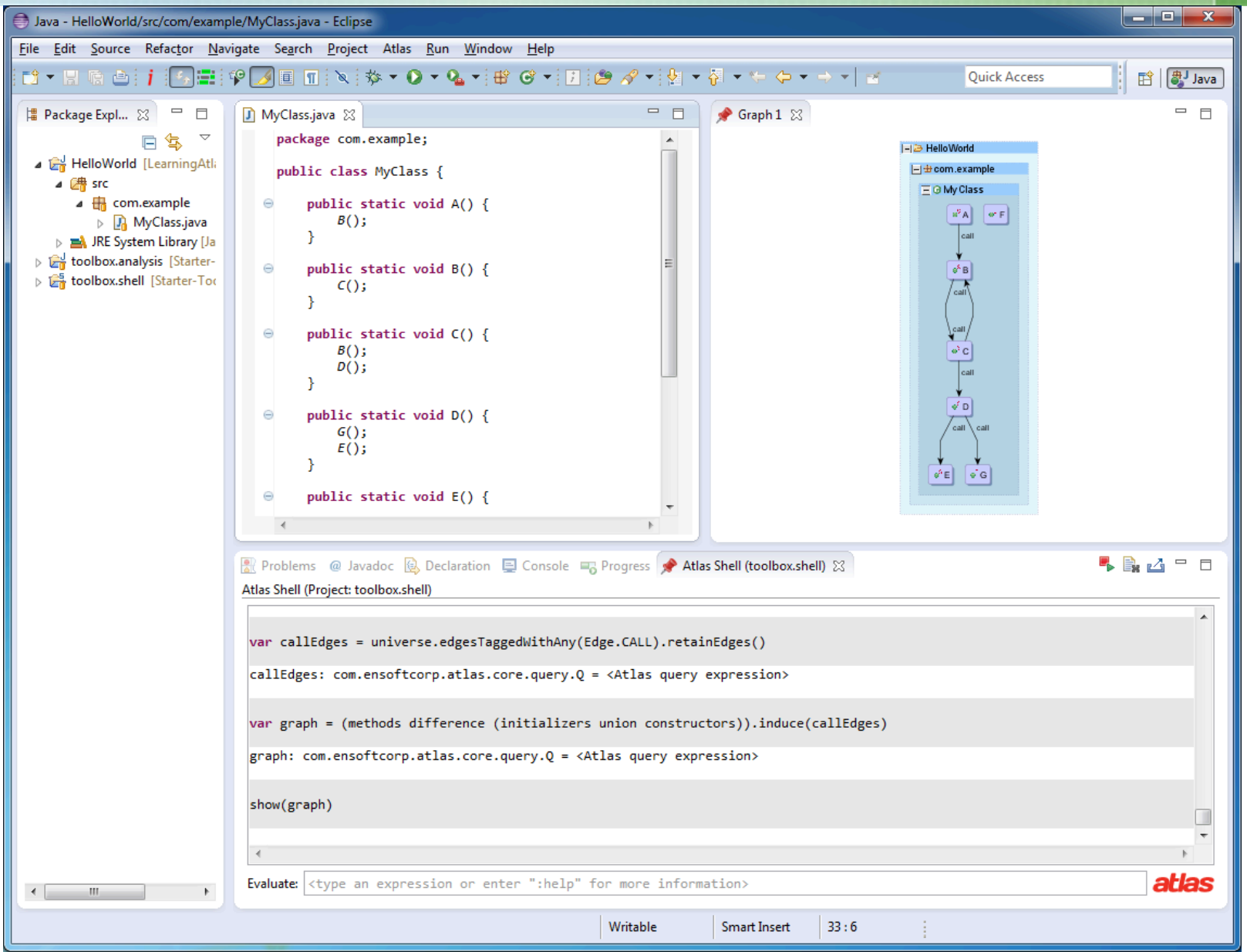
No data in this program.



Basic Queries

1. Map the Workspace project “HelloWorld”
2. Execute the following queries on the Atlas Shell
   (We will discuss what they mean later)

```
var containsEdges = universe.edgesTaggedWithAny(XCSG.Contains).retainEdges()
var app = containsEdges.forward(universe.project("HelloWorld"))
var appMethods = app.nodesTaggedWithAny(XCSG.Method)
var initializers = app.methods("<init>") union app.methods("<clinit>")
var constructors = app.nodesTaggedWithAny(XCSG.Constructor)
var callEdges = universe.edgesTaggedWithAny(XCSG.Call).retainEdges()
var graph = (appMethods difference (initializers union constructors)).induce(callEdges)
show(graph)
```


Java - HelloWorld/src/com/example/MyClass.java - Eclipse
File Edit Source Refactor Navigate Search Project Atlas Run Window Help
Quick Access
Java
Package Expl...
HelloWorld [LearningAtl
src
com.example
MyClass.java
JRE System Library [Ja
toolbox.analysis [Starter-
toolbox.shell [Starter-To
MyClass.java
package com.example;
public class MyClass {
public static void A() {
B();
}
public static void B() {
C();
}
public static void C() {
B();
D();
}
public static void D() {
G();
E();
}
public static void E() {
Graph 1
HelloWorld
com.example
My Class
A
F
B
C
D
E
G
call
Problems
Javadoc
Declaration
Console
Progress
Atlas Shell (toolbox.shell)
Atlas Shell (Project: toolbox.shell)
var callEdges = universe.edgesTaggedWithAny(Edge.CALL).retainEdges()
callEdges: com.ensoftcorp.atlas.core.query.Q = <Atlas query expression>
var graph = (methods difference (initializers union constructors)).induce(callEdges)
graph: com.ensoftcorp.atlas.core.query.Q = <Atlas query expression>
show(graph)
Evaluate:
<type an expression or enter ":help" for more information>
atlas
Writable
Smart Insert
33 : 6

Basic Queries

Declare a few variables to represent different methods in our example graph. Note that you can also use the "selected" variable after clicking on the Atlas graph or corresponding source file element.

var A = app.methods("A")
var B = app.methods("B")
var C = app.methods("C")
var D = app.methods("D")
var E = app.methods("E")
var F = app.methods("F")
var G = app.methods("G")

…alternatively…

var A = selected
var B = selected
…

Note: In the following examples you will need to pass the result to the "show" method on the Atlas Shell to view the results.

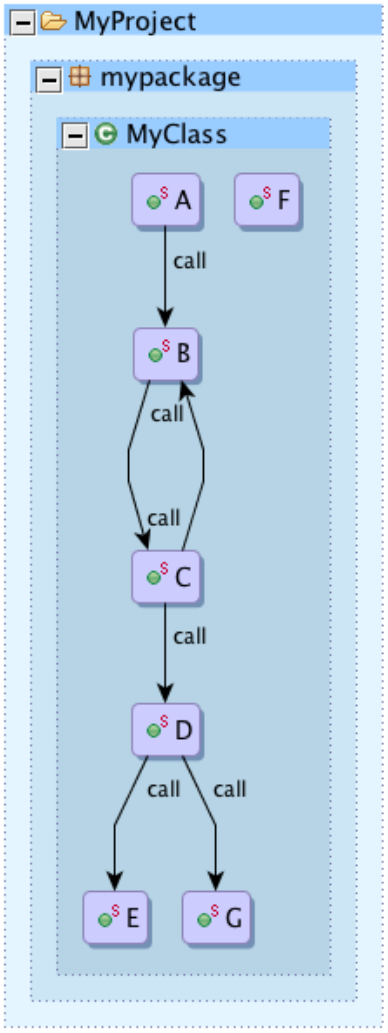
Example: show(graph.forward(D))

Forward Traversals

graph.forward(origin)
Selects the graph reachable from the given nodes using a forward transitive traversal. This query includes the origin in the resulting graph.
graph.forward(D) will output the graph D→E and D→G.
graph.forward(C) will output the graph C→B→C, C→D→E and C→D→G.

graph.forwardStep(origin)
Selects the graph reachable from the given nodes along a path length of 1 in the forward direction. This query includes the origin in the resulting graph.
graph.forwardStep(D) will output the graph D→E and D→G.
graph.forwardStep(C) will output the graph C→B and C→D.
graph.forwardStep(F) will output a graph with only F.

graph.successors(origin)
Selects the successors reachable from the given nodes along a path length of 1. By definition this query does not include the origin unless a node succeeds itself (has a self pointer). The final result never includes edges.
graph.successors(C) will output a graph with only D and B.
graph.successors(F) will output an empty graph.

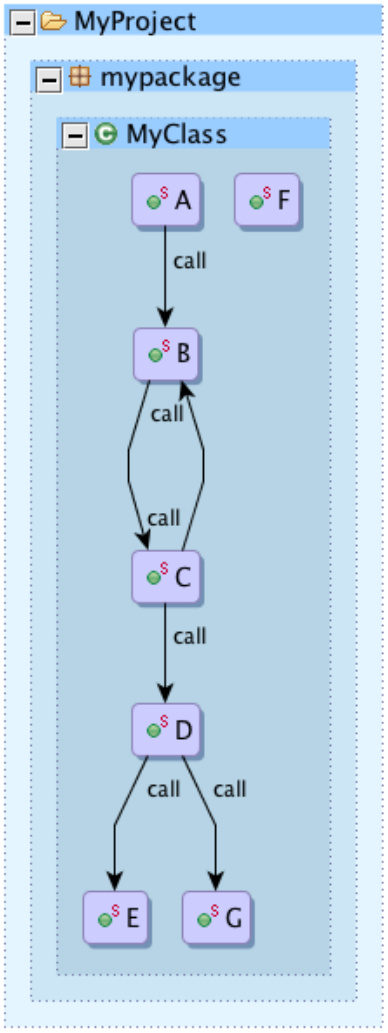


Reverse Traversals

graph.reverse(origin)
Selects the graph reachable from the given nodes using reverse transitive traversal. This query includes the origin in the resulting graph.
graph.reverse(D) will output the graph D←C←B←A and D←C←B←C.
graph.reverse(C) will output the graph C←B←A and C←B←C.

graph.reverseStep(origin)
Selects the graph reachable from the given nodes along a path length of 1 in the reverse direction. This query includes the origin in the resulting graph.
graph.reverseStep(D) will output the graph D←C.
graph.reverseStep(C) will output the graph C←B.

graph.predecessors(origin)
Selects the predecessors reachable from the given nodes along a path length of 1. By definition this query does not include the origin unless a node precedes itself (has a self-pointer). The final result never includes edges.
graph.predecessors(C) will output a graph with only B.
graph.predecessors(F) will output an empty graph.



Set Operations

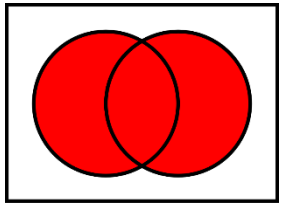
graph.union(otherGraphs...)

Yields the union of this graph and the given graphs. That is, the resulting graph's nodes are the union of all nodes, and likewise for edges.

B.union(C) outputs a graph with nodes B and C.

A.union(B, C) outputs a graph with nodes A, B, and C.

graph.union(C) outputs the entire graph.

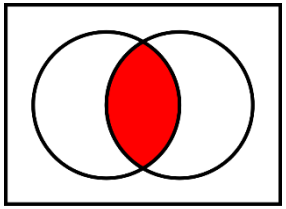


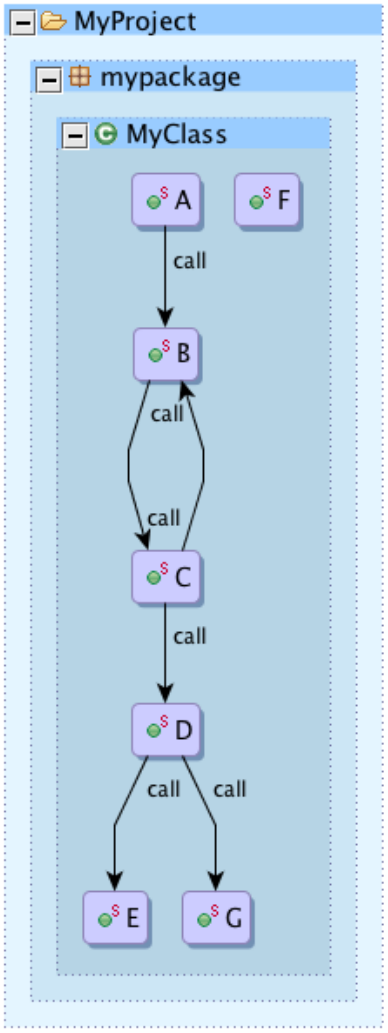
graph.intersection(otherGraphs...)

Yields the intersection of this graph and the given graphs. That is, the resulting graph's nodes are the intersection of all node sets, and likewise for edges.

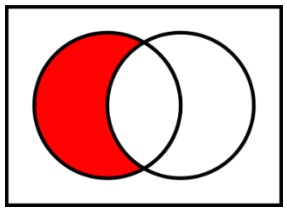
A.intersection(B) outputs an empty graph.

graph.intersection(C) outputs a graph with only the node C.





Set Operations (Continued)



graph.difference(otherGraphs...)

Select the current graph, excluding the given graphs. Note that, because an edge is only in a graph if its nodes are in a graph, removing an edge will necessarily remove the nodes it connects as well. Removing either node would remove the edge as well.

B.difference(C) outputs a graph with only node B.

B.difference(A, B) outputs an empty graph.

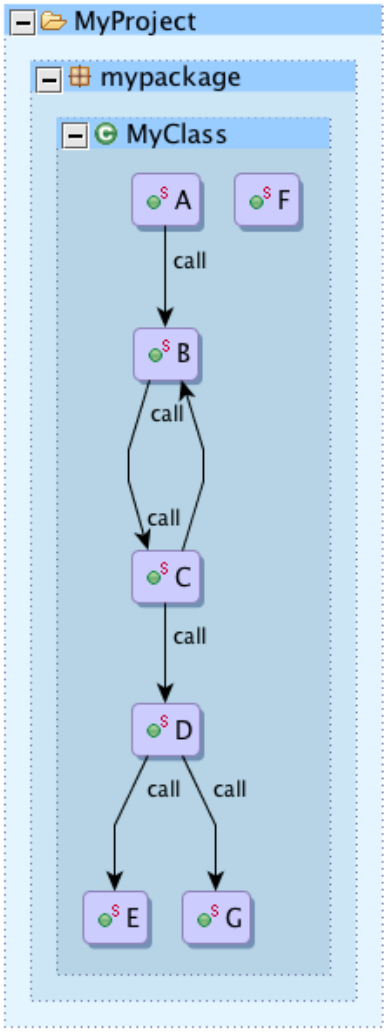
graph.difference(C) outputs the figure above without the node C and any edges entering or leaving node C.

graph.differenceEdges(otherGraphs...)

Selects the current graph, excluding the edges from the given graphs.

graph.differenceEdges(graph) outputs a graph with only the nodes A,B,C,D,E,F,G.

graph.differenceEdges(graph.forwardStep(B)) outputs the graph A→B, C→B, C→D, D→E, D→G, and F (the edge B→C was removed from the original graph).

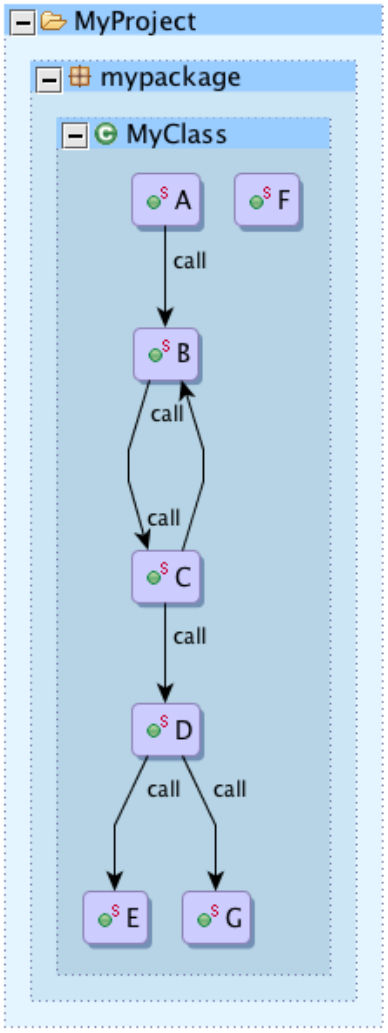


Between Traversals

graph.between(from, to)
Selects the subgraph such that the given nodes in to are reachable from the nodes in from using forward traversal. Note this query selects all paths in the graph between the "from" and "to" nodes.
graph.between(C, A) will output an empty graph.
graph.between(C, E) will output the graph C→D→E, C→B→C.

graph.betweenStep(from, to)
Selects the subgraph such that the given nodes in to are reachable from the nodes in from using forward step traversal.
graph.betweenStep(C, D) will output the graph from C→D.
graph.betweenStep(D, C) will output an empty graph.
graph.betweenStep(C, E) will output an empty graph.

Note: We can intuitively say that if there is a direct edge from the '*from*' node to the '*to*' node then betweenStep() will output the nodes with the edge between them. A possible implementation of betweenStep could be: *graph.forwardStep(from).intersection(graph.reverseStep(to))*



Graph Operations

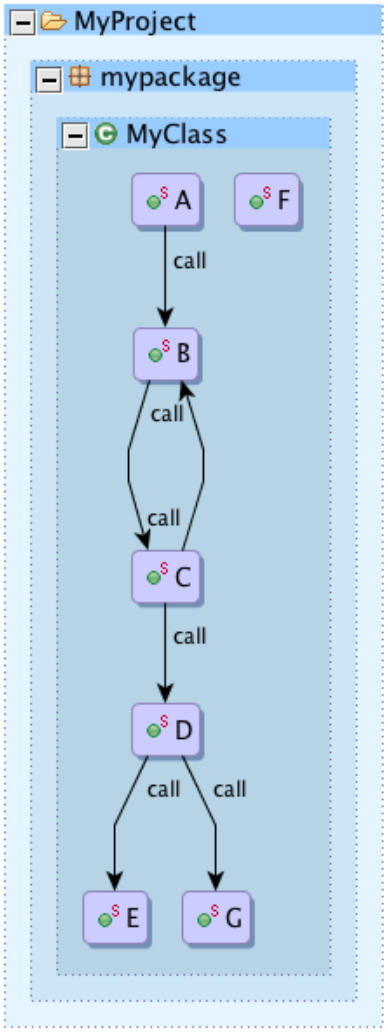
graph.leaves()
Selects the nodes from the given graph with no successors.
graph.leaves() outputs a graph with the nodes E, F, and G.

graph.roots()
Selects the nodes from the given graph with no predecessors.
graph.roots() outputs a graph with the nodes A, F.

graph.retainNodes()
Selects all nodes from the graph, ignoring edges.
graph.retainNodes() will output a graph with only the nodes A,B,C,D,E,F,G.

graph.retainEdges()
Retain only edges, retaining only those nodes directly connected to a retained edge.
graph.retainEdges() will return the figure above without the node F because the node F does not have a call edge.

subgraph.induce(graph)
Adds edges from the given graph to the subgraph.
var subgraph = B.union(C)
subgraph.induce(graph) will induce the edges between B and C and gives the output graph B→C→B.



Graph Elements

- In Atlas a *Q* object can be thought of as a recipe to a constraint satisfaction problem (CSP). Building and chaining together *Q*'s costs you almost nothing, but when you ask to see what is in the *Q* (by showing or evaluating the *Q*) Atlas must evaluate the query and execute the graph traversals.
- The evaluated result is a *Graph*. A *Graph* is a set of *GraphElement* objects. In Atlas both a node and an edge are *GraphElement* objects.

```
Graph methodsGraph = graph.eval();
AtlasSet<GraphElement> methodNodes = methodsGraph.nodes();
AtlasSet<GraphElement> methodEdges = methodsGraph.edges();
```

Attributes

- *GraphElement*s may have many attributes
- An attribute is a key that corresponds to a value in the *GraphElement* attribute map.
  - An attribute that is common to almost all nodes and edges is *XCSG.name*.

```
for(GraphElement methodNode : methodNodes){
   String name = (String) methodNode.attr().get(XCSG.name);
}
```

  - Another common attribute is the source correspondence that stores the file and character offset of where the node corresponds to in source code. Try double clicking on a node or edge to jump to the corresponding source code!

Selecting on Attributes

- Attributes can be used to select *GraphElement*s (nodes/edges) out of a graph.
  - For example from the graph we can select all method nodes with the attribute key *XCSG.name* that have the value "main".

    Q mainMethods = graph.selectNode(XCSG.name, "main");

  - We could also select all array's with 3 dimensions.

    Q 3DimArrays = graph.selectNode(XCSG.arrayDimension, 3);

Tags

- *GraphElement*s may have many attributes
- Intuitively, tags can be thought of as special attributes where the value is always a Boolean true.
- The presence of a tag denotes that a node or edge is a member of a set.
  - For example all nodes in the Atlas map that are methods are tagged with *XCSG.Method*.
- A tag is just a string value, but Atlas provides several default tags such as *XCSG.Method*, which have defined constants that should be used to make your code cleaner (and safer from possible Schema changes in the future!).

Selecting on Tags

graph.nodesTaggedWithAny(...)

Selects the graph where nodes are tagged with at least one of the given tags. The final result contains only nodes.

graph.nodesTaggedWithAny(XCSG.Method) will output a graph with only the Method nodes A,B,C,D,E,F,G.

graph.nodesTaggedWithAny(XCSG.Class) will output an empty graph because graph does not contain Class nodes.

graph.nodesTaggedWithAny(XCSG.Method, XCSG.Class) will output a graph with only the Method nodes A,B,C,D,E,F,G.

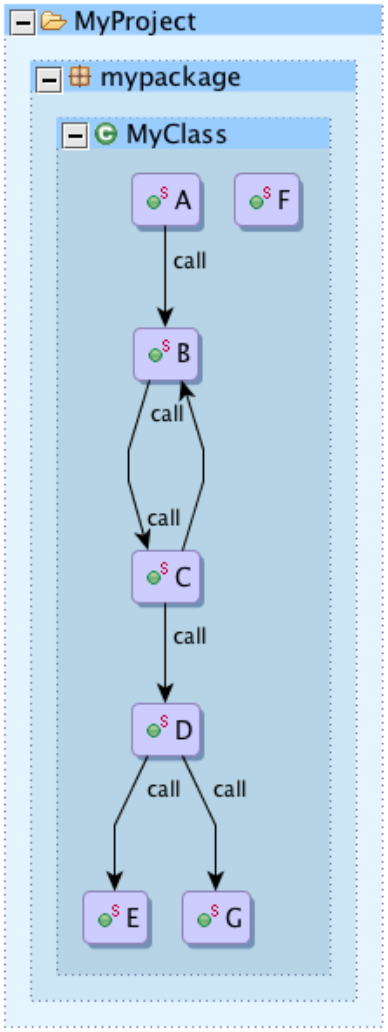
graph.nodesTaggedWithAll(...)

Selects the subgraph where nodes are tagged with all of the given tags. The final result contains only nodes.

graph.nodesTaggedWithAll(XCSG.Method) will output a graph with only the Method nodes A,B,C,D,E,F,G.

graph.nodesTaggedWithAll(XCSG.Class) will output an empty graph because graph does not contain Class nodes.

graph.nodesTaggedWithAll(XCSG.Method, XCSG.Class) will output an empty graph because graph does not contain nodes that are both Class and Method nodes.



Selecting on Tags (Continued)

graph.edgesTaggedWithAny(...)

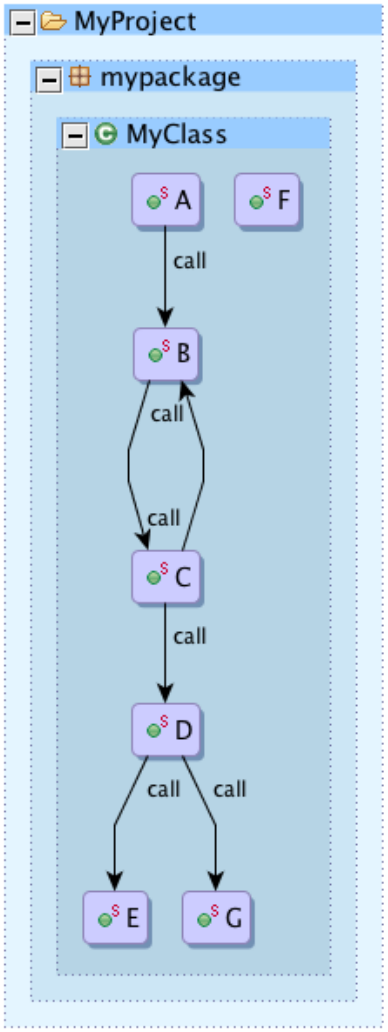
Selects the subgraph where edges are tagged with at least one of tags. Since an edge is defined as a relationship between two nodes this query returns a graph with nodes and edges. Note this will also it will return all the nodes in graph.

graph.edgesTaggedWithAny(XCSG.Call) will generate the graph shown in the figure above.

graph.edgesTaggedWithAll(...)

Selects the subgraph where edges are tagged with all of the given tags. Since an edge is defined as a relationship between two nodes this query returns a graph with nodes and edges. Note this will also it will return all the nodes in graph.

graph.edgesTaggedWithAll(XCSG.Call) will generate the graph shown in the figure above.



Chaining Queries

- We can chain queries together to form more complex queries
- *Q* objects may contain multiple nodes and edges (so an origin can include multiple starting points).
- Let's take a look at the queries we started the example with again…

1. First create a subgraph (called containsEdges) of the universe that only contains nodes and edges connected by a *contains* relationship. The Atlas map is heterogeneous, meaning there are many edge and node types. Here we are specifying that we want edges that represent a *contains* relationship from a parent node to a child node. Note that we called retainEdges() here because we want to throw away nodes that are not connected by a *contains* edge, otherwise those nodes may still be in the resulting containsEdges graph.

var containsEdges = universe.edgesTaggedWithAny(XCSG.Contains).retainEdges()

Chaining Queries (Continued)

2. We then define yet another subgraph (called app) which contains nodes and edges declared under the HelloWorld project.

var app = containsEdges.forward(universe.project("HelloWorld"))

3. From the app subgraph we select all nodes that are methods.

var appMethods = app.nodesTaggedWithAny(XCSG.Method)

4. From the app subgraph we select all nodes that are methods that are named "<init>" (instance initializer methods) or "<clinit>" (static initializer methods). We are using a query method called methods(String methodName) that selects methods that have a name that matches the given string. We will explore more query methods in the next section.

var initializers = app.methods("<init>") union app.methods("<clinit>")

Note that since this is Scala the ".", "(", ")", and ";" characters are implied. So the above could also be written as the following:

var initializers = app.methods("<init>").union(app.methods("<clinit>"));

Chaining Queries (Continued)

5. From the app subgraph we select all nodes that are constructors.

var constructors = app.nodesTaggedWithAny(XCSG.Constructor)

6. From the universe create a subgraph (called callEdges) that only contains nodes and edges connected by a call relationship.

var callEdges = universe.edgesTaggedWithAny(XCSG.Call).retainEdges()

7. Define graph to be the methods in the app ignoring initializers and constructors with call edges added in where they exist.

var graph = (appMethods difference (initializers union constructors)).induce(callEdges)

8. Display the graph.

show(graph)

Atlas Schema

- To become proficient in wielding Atlas, you should have:
  - Firm understanding of Extensible Common Software Graph (XCSG) schema
  - Firm understanding of the language you are analyzing
    - (Java source, Jimple, vs. C)

- Example: How do we detect an inner class with XCSG?
  - No tag for inner class, inner class is defined by a *contains* relationship.

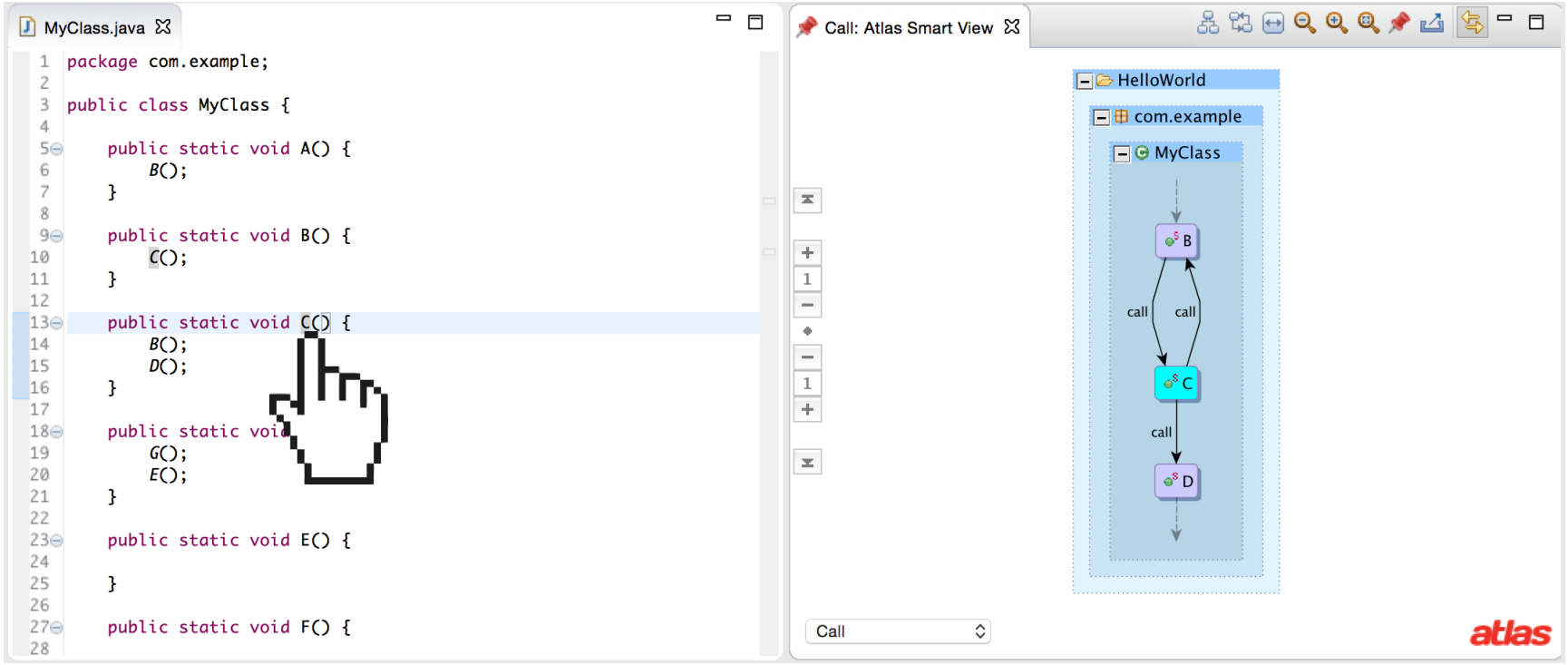
    containsEdges = universe.edgesTaggedWithAny(XCSG.Contains)
    topLevelClasses = containsEdges.successors(universe.nodesTaggedWithAny(XCSG.Package))
    innerClasses = containsEdges.forward(topLevelClasses).difference(topLevelClasses)
  - What about Java vs. Jimple?
    - No concept of inner classes in bytecode

Atlas Schema

- Resources
  - http://ensoftatlas.com/wiki/Extensible_Common_Software_Graph
  - Eclipse → Show Views → Other… → Atlas → Element Detail View
  - Atlas Shell (test out queries on the fly!)
  - Atlas Smart Views (interactive graphs)

Atlas Smart Views

- Observation: Some queries are very common tasks and could be automated!



ConnectBotBad

ConnectBot is a powerful open-source Secure Shell (SSH) client. It can manage simultaneous SSH sessions, create secure tunnels, and copy/paste between other applications. This client allows you to connect to Secure Shell servers that typically run on UNIX-based servers.

Challenge: Find any malware(s) that may exist in the source code as efficiently as possible.

Tip: Work smarter not harder…