Learn to Build Automated Software Analysis Tools with Graph Paradigm and Interactive Visual Framework

GIAN, September 12-16, 2016

Suresh C. Kothari
Richardson Professor
Department of Electrical and Computer Engineering

Ben Holland, Iowa State University

Module VI: Software Analysis Lab Exercises and Projects

Acknowledgement: **Team members at Iowa State University and EnSoft, DARPA contracts FA8750-12-2-0126 & FA8750-15-2-0080**

Software Lab Exercises Projects

Hopefully, you can use this material in your research and teaching, build on it, and share what you have added. THANK YOU.

- Goals:
  - Learn to solve complex software problems by applying critical thinking.
  - Practice the graph paradigm with Atlas to apply critical thinking to large software.
- Sample Lab Exercises and Projects:
  - Lab exercises: (1) Experiments using Atlas to understand and apply program analysis with graph paradigm, (2) Implement software metrics and analyze results using Java code, (3) Detect a class of software safety and security vulnerabilities
  - Project – Use graph models to verify the Linux kernel for a safety property
- Planned activities:
  - Understand the importance of the posed problem and its challenges.
  - Identify the relevant fundamental principles and develop a problem solving strategy.
  - Learn by performing experiments.
  - Learn to communicate the solution precisely and succinctly.

Lab Exercise 1: Understand and apply program analysis

- Goal
  - Hands-on experience to reinforce the course material.
  - Prepare participants to make effective use of the graph paradigm in their research, education, and practice.
- Planned Activities
  - Use the analysis and visualization capabilities interactively.
  - Write and execute short program analysis queries.
  - Introduce the eXtensible Common Software Graph schema for C and Java.

Lab Exercise 2: Understand and apply program analysis

- Goal: Learn how to compute software metrics using Atlas and analyze the pros and cons of a particular metric
- The lab will have experiments to study:
  - What indicators does well designed software have?
  - What indicators does poorly designed software have?

Software Architecture (SA)

- A good SA is essential for easy software maintenance (modification of the software product after delivery to correct faults, to improve performance or other attributes) and adding new features.
- Our focus: the aspect of SA that deals with architecture of the modules (packages and classes) and their interconnections.

Motivating Questions….

- How difficult can (even a simple) a code change be?
  - Does it cause a cascade of subsequent changes in dependent modules? (**Rigidity**)
  - Does it break software in multiple places – even in areas not conceptually related to the original code change? (**Fragility**)
- How easy is it to reuse a software module?
  - What is the overhead in separating the desirable parts from undesirable parts? Is the overhead so much that rewriting the software would be easier? (**Immobility**)
- Is the right way the hard way?
  - Do the design constraints make it hard to follow the right way, and instead an improperly designed "hack" is easy to implement. (**Viscosity**)

References

- Software Package Metrics
  - Design Principles and Design Patterns, Robert C. Martin[1], 2000. Article available at: http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf
  - Wikipedia: http://en.wikipedia.org/wiki/Software_package_metrics
  - Book "**Agile Software Development:** Principles, Patterns, and Practices": http://books.google.com/books/about/Agile_Software_Development.html?id=0HYhAQAAIAAJ

[1]Martin is a software professional since 1970 and an international software consultant since 1990. In 2001, he initiated the meeting of the group that created agile software development from extreme programming techniques.
See http://en.wikipedia.org/wiki/Robert_Cecil_Martin

Package Coupling Principles

- Applications are networks of interrelated packages.
- The interrelationship should obey certain principles for efficient software maintenance.
  - "Package Coupling Principles"
- We capture packages interrelations by package dependency metrics.
- Package X is dependent on package Y if X cannot be compiled without Y.

Package Coupling Principles

- Package X is dependent on package Y if
  - Some code in X read/writes into a field declared in some class in Y
  - A class in X extends/implements a class in Y
  - Some code in X calls a method in Y

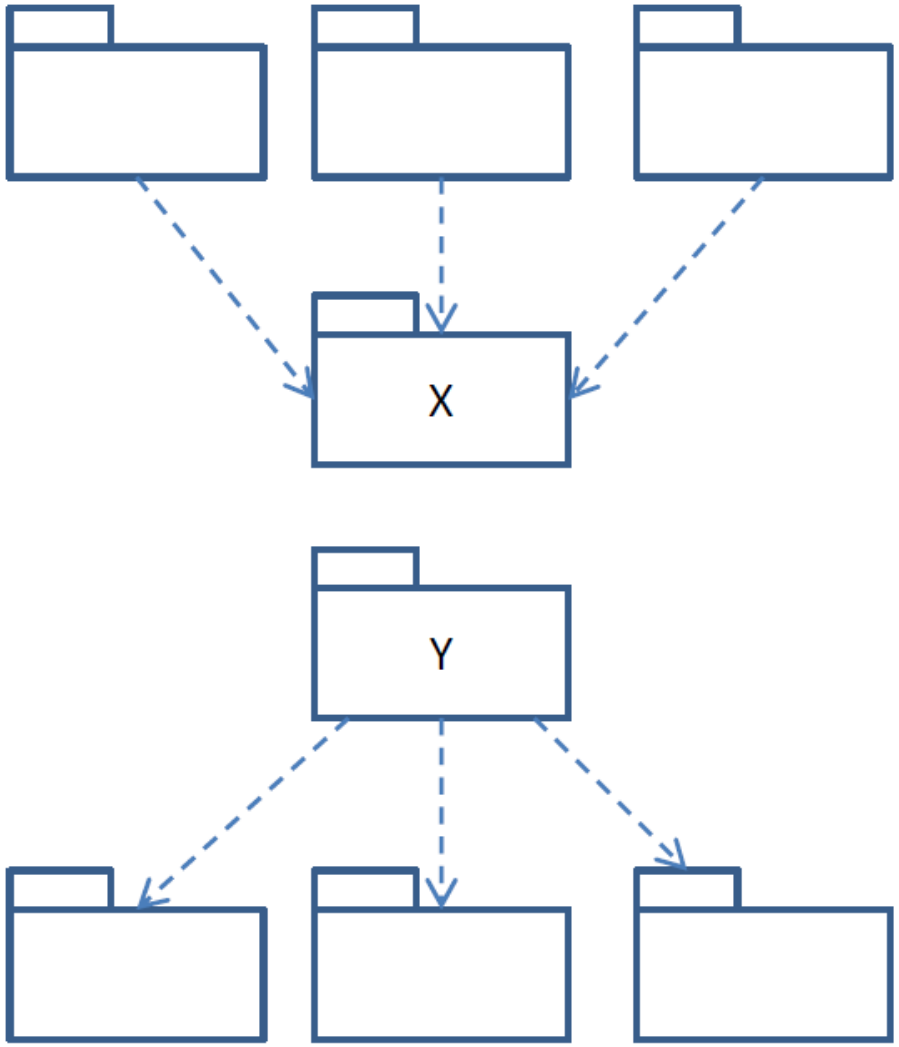
The Acyclic Dependencies Principle

- *The dependencies between packages must not form cycles.*
  - Code-changes are usually localized to a package because a good design groups related classes together in a package. The changed package should be compiled and tested with the packages it depends on. If there is a cyclic dependency, a change in any one package in the cycle requires the test suite to be built with all the packages in the cycle – clearly not a desirable thing.

The Stable Dependencies Principle (SDP)

- A package should be stable if it has many packages dependent on it.
  - Because it requires a great deal of work to reconcile any changes in the stable package with all the dependent packages.
  - For such a package should be stable, it should not be heavily dependent on other packages.

Dependent vs. Independent Packages

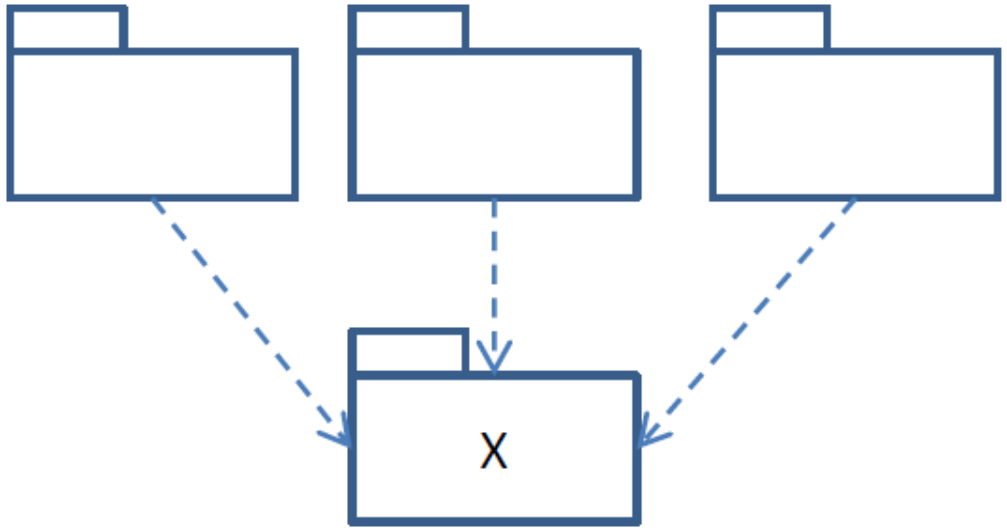


- X is independent
  - It has three packages dependent on it.
  - X depends upon nothing, so it has no external influence to make it change. We say it is *independent.*
- Y is dependent
  - It has no packages depending on it.
  - Y depends on three packages, so changes may come from three external sources. We say that Y is *dependent*.

Stability Metrics

- Efferent Coupling (Ce): The number of classes outside the package that the classes inside the package depend upon (outgoing dependencies).
- Afferent Coupling (Ca): The number of classes outside the package that depend upon the classes inside the package (incoming dependencies).
- Instability (I): Ce/(Ca+Ce)
  - Range [0,1]; 0: very stable, 1: very unstable.
- SDP can be rephrased as: “Depend upon packages whose instability metric is lower than yours.
- We can envision the packages structure of our application as a set of interconnected packages with unstable packages at the top, and stable packages on the bottom. In this view, all dependencies point downwards.

The Stable Abstractions Principle (SAP)

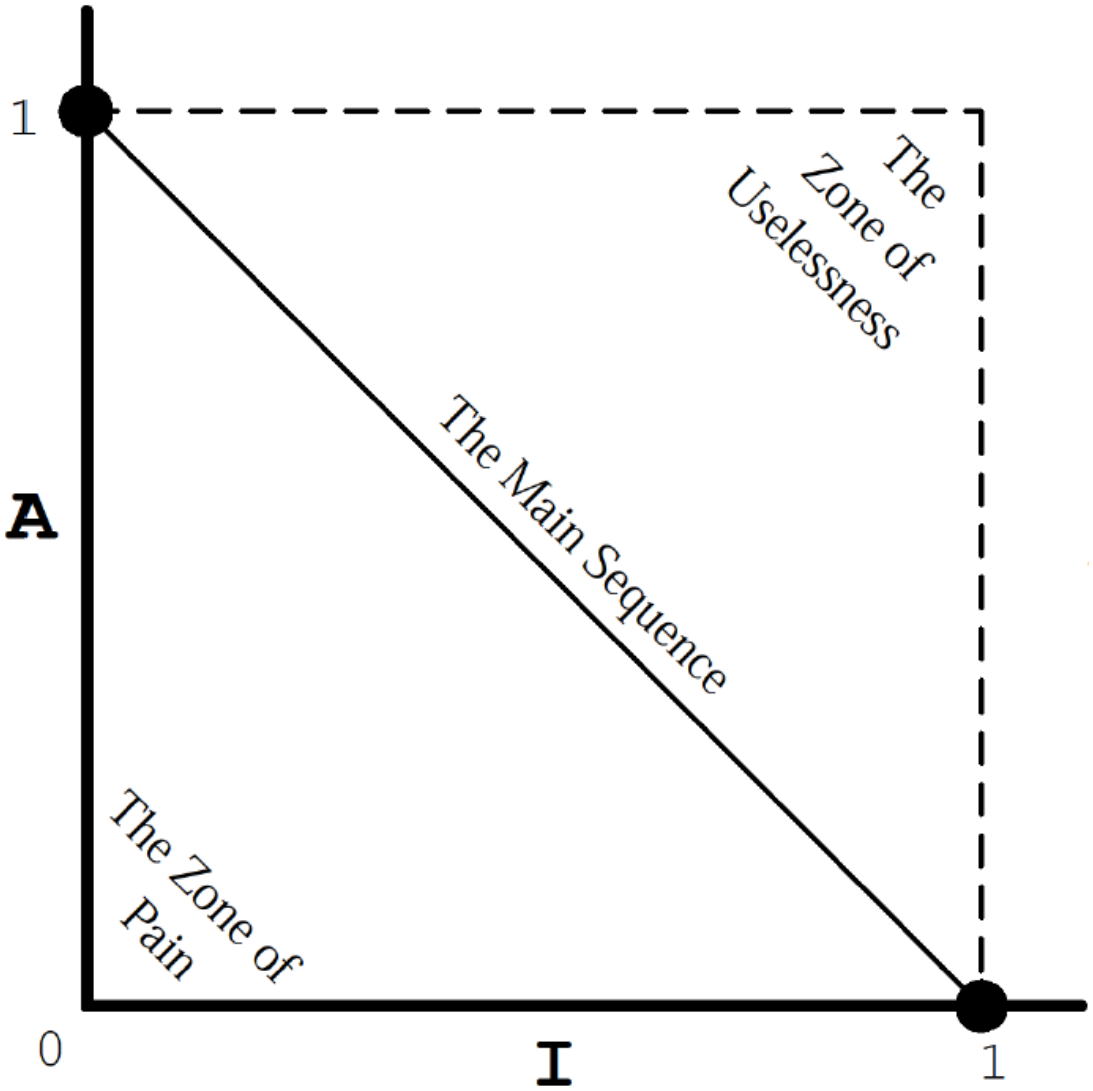


- SAP: Heavily used packages should have a high-degree of abstraction.
- A high-degree of abstraction implies that packages are easy to change or extend.

Abstractness Metrics

- *Nc:* Number of classes in the package.
- *Na:* Number of abstract classes in the package.
- Abstractness (A) = *Na/Nc*
  - Range [0,1]; 0: no abstract classes, 1: all classes are abstract
- SAP can be restated as: I should increase as abstractness decreases. That is, concrete packages should be unstable (flexible) while abstract packages should be stable.

Abstractness vs Instability



- Ideally a package should lie on “The Main Sequence” line
- Distance (D): |A+I-1|
  - Range [0,1]; 0: package directly on the main sequence, 1: farthest away from the main sequence.

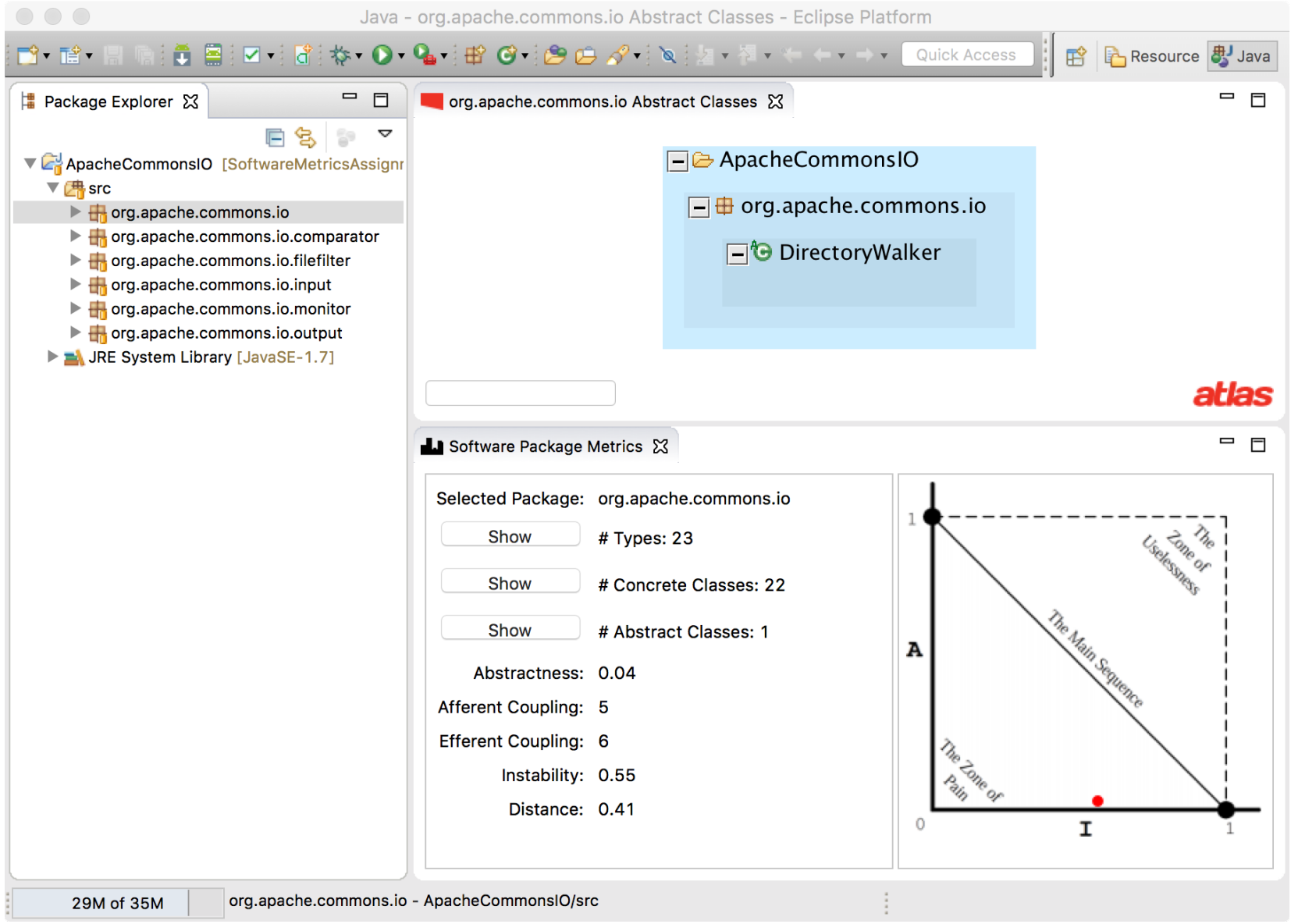
Reflections on Software Metrics

- These metrics are meant to provide a measure of how good the software architecture is.
- But, they are imperfect, and reliance upon them as the sole indicator of a sturdy architecture would be foolhardy.

Classroom Activity

- Develop a program analysis tool to compute the previously described software metrics
  - Implement TODOs in base project: https://github.com/benjholla/SoftwareMetricsAssignment
- Evaluate the Apache Commons IO library

Classroom Activity



Classroom Activity

- With regard to the Distance (D) from the Main Sequence is the org.apache.commons.io package better or worse compared to the org.apache.commons.io.filefilter package?
- Do you feel these metrics accurately capture the quality of this code?

Classroom Activity: Solution

```
public static Q getClasses(Q graph){
    return graph.nodesTaggedWithAny(XCSG.Java.AbstractClass, XCSG.Java.Class);
}

public static Q getInterfaces(Q graph){
    return graph.nodesTaggedWithAll(XCSG.Java.Interface);
}

public static Q getAbstractClasses(Q graph){
    return graph.nodesTaggedWithAny(XCSG.Java.AbstractClass, XCSG.Java.Interface);
}

public static Q getConcreteClasses(Q graph) {
    return getClasses(graph).difference(getAbstractClasses(graph));
}
```

Classroom Activity: Solution

```
public static double getAbstractness(Node pkg) throws ArithmeticException {
    // Step 1) Get the types declared under the package
    Q packageTypes = getPackageTypes(pkg);
    // Step 2) From the discovered class nodes select the abstract classes
    Q abstractClasses = getAbstractClasses(packageTypes);
    // Step 4) Return the ratio of the number of abstract classes (and interfaces) in the package to the total number of types in the package
    return (double) countNodes(abstractClasses) / (double) countNodes(packageTypes);
}
```

Classroom Activity: Solution

```
public static Q getAfferentCouplings (Node pkg){
    // Step 1) Get the methods declared under the package
    Q packageMethods = getPackageMethods(pkg);
    // Step 2) Create a subgraph of CALL edges from the universe
    Q callEdges = Common.universe().edgesTaggedWithAny(XCSG.Call).retainEdges();
    // Step 3) Within the calls subgraph get the predecessors of the package methods (calling methods)
    Q callingMethods = callEdges.predecessors(packageMethods);
    // Step 4) Return the packages of the calling methods
    return getPackagesOfProgramElements(callingMethods);
}
```

Classroom Activity: Solution

```
public static Q getEfferentCouplings (Node pkg){
    // Step 1) Get the methods declared under the package
    Q packageMethods = getPackageMethods(pkg);
    // Step 2) Create a subgraph of CALL edges from the universe
    Q callEdges = Common.universe().edgesTaggedWithAny(XCSG.Call).retainEdges();
    // Within the calls subgraph get the successors of the package methods (called methods)
    Q calledMethods = callEdges.successors(packageMethods);
    // Step 4) Return the packages of the called methods
   return getPackagesOfProgramElements(calledMethods);
}
```

Classroom Activity: Solution

```
public static double getInstability(Node pkg) {
    double ce = (double) countNodes(getEfferentCouplings(pkg));
    double ca = (double) countNodes(getAfferentCouplings(pkg));
    return ce / (ca + ce);
}

public static double getDistance(Node pkg) {
    double a = getAbstractness(pkg);
    double i = getInstability(pkg);
     return Math.abs(a + i - 1);
}
```

Classroom Activity: Solution

Package	Types	Concrete Classes	Abstract Classes	Abstractness	Afferent Coupling	Efferent Coupling	Instability	Distance	Properties
org.apache.commons.io	23	22	1	0.04	5	6	0.55	0.41	Least Abstract
org.apache.commons.io.comparator	10	9	1	0.1	4	3	0.43	0.47	
org.apache.commons.io.filefilter	25	22	3	0.12	3	6	0.67	0.21	Least Stable
org.apache.commons.io.input	27	24	3	0.11	6	3	0.33	0.56	Most Stable
org.apache.commons.io.monitor	5	4	1	0.2	3	4	0.57	0.23	Most Abstract
org.apache.commons.io.output	19	18	1	0.05	4	3	0.43	0.52	

- With regard to the Distance (D) from the Main Sequence is the org.apache.commons.io package better or worse compared to the org.apache.commons.io.filefilter package?
  - According to the metric, the distance D from the main sequence indicates org.apache.commons.io is worse designed than org.apache.commons.io.filefilter, since the org.apache.commons.io.filefilter package is significantly closer to 0 than the org.apache.commons.io package.

Classroom Activity: Solution

Package	Types	Concrete Classes	Abstract Classes	Abstractness	Afferent Coupling	Efferent Coupling	Instability	Distance	Properties
org.apache.commons.io	23	22	1	0.04	5	6	0.55	0.41	Least Abstract
org.apache.commons.io.comparator	10	9	1	0.1	4	3	0.43	0.47	
org.apache.commons.io.filefilter	25	22	3	0.12	3	6	0.67	0.21	Least Stable
org.apache.commons.io.input	27	24	3	0.11	6	3	0.33	0.56	Most Stable
org.apache.commons.io.monitor	5	4	1	0.2	3	4	0.57	0.23	Most Abstract
org.apache.commons.io.output	19	18	1	0.05	4	3	0.43	0.52	

- Do you feel these metrics accurately capture the quality of this code?
  - Depends. Code metrics are an approximation that capture some properties of the software. Keep in mind that Apache Commons IO is a library. What would happen if we consider the libraries interactions with the JDK? What would happen if we consider the libraries interactions with an application using the library?

Lab Exercise 3: Detect a class of software vulnerabilities

- Goal
  - Bring out the challenges of intricate data and control-flow dependencies.
  - Demonstrate how graph models can address the challenges.
- Planned Activities
  - Perform experiments to understand the challenges of multiplicative growth of execution paths and path feasibility.
  - Identify the fundamentals necessary to address the challenges.
  - Apply the principles to design graph models for effective detection and verification.
  - Practice critical thinking through experiments to design the graph models.

A software vulnerability problem

- Problem: Check whether the memory allocation in the XINU disk driver code leads to memory leak.
- Given: relevant disk driver routines – about 200 lines of C code.

Problem Details

Problem Overview: The given code includes some disk driver routines. getbuf() and freebuf() are respectively the calls to *allocate* and *deallocate* memory dynamically. The goal is to verify that an *allocation* is followed by *deallocation* on all feasible execution paths. The code presents two allocation instances: (a) a simple instance in dsread() where allocation is followed by deallocation in the same function, (b) a complex instance in dswrite() where allocation is *not* followed by deallocation in the same function. How to verify that allocation in dswrite() is followed by deallocation on all feasible execution paths? How many paths are there? Where could be the matching deallocation, if not in dswrite()?

Side Questions: Why is the deallocation not in dswrite() itself? Is there a good justification to not have the deallocation in dswrite()? what is that justification?

Discussion Points: Why is the problem so hard? What are the prevalent practices to solve the problem? Why do we need a fundamentally different approach to solve the problem?

Memory leak verification

- Problem: Verify that each instance A(x) of memory allocation (event E1) is followed by deallocation D(X) (event E2) on every *feasible* execution path.
  - This is a 2-event problem.
  - Non-occurrence of an event is fault!

Lock/Unlock Pairing verification

- Problem: Verify that each instance L(x) of Lock (event E1) is followed by Unlock U(X) (event E2) on every *feasible* execution path.
  - This is a 2-event problem.
  - Non-occurrence of an event is fault!

Anti-Matching Pair (AMP) verification

- Problem: Verify that event E1(X) is *not* followed by event E2(X) on every *feasible* execution path.
  - This is a 2-event problem.
  - Occurrence of an event is fault!

A concrete example of AMP

- Sensitive Source Event : Occurrence of an API with the permission to *read* phone state. Let us call this E1(X) where X is the pointer to the phone state.
- Malicious Sink: Occurrence of an API with the permission to *access* Internet. Let us call this E2(X) where X is the pointer to the phone state.
- Problem: Verify that event E1(X) is *not* followed by event E2(X) on every *feasible* execution path.

MP: False positives and false negatives

REALITY	VERIFICATION RESULT	ACCURACY
E1(x) followed by E2(x) on all paths	Violation YES	FP
E1(x) followed by E2(x) on all paths	Violation NO	Correct
E1(x) not followed by E2(x) on some path	Violation YES	Correct
E1(x) not followed by E2(x) on some path	Violation NO	FN

AMP: False positives and false negatives

REALITY	VERIFICATION RESULT	ACCURACY
E1(x) not followed by E2(x) on all paths	Violation YES	FP
E1(x) not followed by E2(x) on all paths	Violation NO	Correct
E1(x) followed by E2(x) on some path	Violation YES	Correct
E1(x) followed by E2(x) on some path	Violation NO	FN

Sophisticated vulnerabilities

- The vulnerability has a catastrophic unaccepted behavior.
- A reactive fix is not a good solution.
- The trigger for enacting the vulnerability is obscure and rare.
- Testing by itself is not a viable verification option.
- Discovering the vulnerability is like looking for the needle in the haystack, not knowing what the needle looks like.
- The vulnerabilities are often hidden due to invisible control and/or data flows in the program.

The commonly used reliability testing, software reliability measures, analysis of factors causing defects, and the cybersecurity precautions are generally not applicable or effective in the case of critical vulnerabilities.

Defining problem hardness

- The hardness of verifying one instance can be very different than the hardness of verifying another instance.
- Without knowing the hardness, we are working in the dark to advance the research.
- What defines the hardness?

Memory Leak Example 1

```
dsread(devptr, buff, block)
     struct   devsw    *devptr;
     char *buff;
     DBADDR   block;
{
     struct   dreq *drptr;
     int  stat;
     char ps;

     disable(ps);
     drptr = (struct dreq *) getbuf(dskrbp);
     drptr->drdba = block;
     drptr->drpid = currpid;
     drptr->drbuff = buff;
     drptr->drop = DREAD;
     if ( (stat=dskenq(drptr, devptr->dvioblk)) == DONQ) {
          stat = drptr->drstat;
     }
     freebuf(drptr);
     restore(ps);
     return(stat);
}
```

Verification Easy!

Memory Leak Example 2

```
dswrite(devptr, buff, block)
        struct   devsw    *devptr;
        char *buff;
        DBADDR   block;
{
        struct   dreq *drptr;
        char ps;

        disable(ps);
        drptr = (struct dreq *) getbuf(dskrbp);
        drptr->drbuff = buff;
        drptr->drdba = block;
        drptr->drpid = currpid;
        drptr->drop = DWRITE;
        dskenq(drptr, devptr->dvioblk);
        restore(ps);
        return(OK);
}
```

Verification Hard!

Let us get to the fundamentals of what makes it hard.

Let's try to Characterize *Hardness*

```
dswrite(devptr, buff, block)
     struct   devsw    *devptr;
     char *buff;
     DBADDR   block;
{
     struct   dreq *drptr;
     char ps;

     disable(ps);
     drptr = (struct dreq *) getbuf(dskrbp);
     drptr->drbuff = buff;
     drptr->drdba = block;
     drptr->drpid = currpid;
     drptr->drop = DWRITE;
     dskenq(drptr, devptr->dvioblk);
     restore(ps);
     return(OK);
}
```

Interprocedural – not a good enough characterization of hardness, Why?

Let's Look a Little Deeper - 1

4 paths

```
dskenq(drptr, dsptr)
     struct   dreq *drptr;
     struct   dsblk    *dsptr;
     { struct dreq *p, *q;
     if ( (q=dsptr->dreqlst) == DRNULL ) {
          dsptr->dreqlst = drptr;
1         dskstrt(dsptr);
          return();}
     for () {
          if ((st=dskqopt(p, q, drptr)!=SYSERR))
2                 return();
          if (   ) {
               q->drnext = drptr;
3              return();} }
     q->drnext = drptr;
4    return();   }
```

Path 1: drptr passed to dskstrt(), which does not deallocate –a memory leak?

Let's Look a Little Deeper - 2

```
dskenq(drptr, dsptr)
     struct   dreq *drptr;
     struct   dsblk    *dsptr;
     { struct dreq *p, *q;
     if ( (q=dsptr->dreqlst) == DRNULL ) {
          dsptr->dreqlst = drptr;
1         dskstrt(dsptr);
          return();}
     for () {
          if ((st=dskqopt(p, q, drptr)!=SYSERR))
2                 return();
          if (   ) {
               q->drnext = drptr;
3              return();} }
     q->drnext = drptr;
4    return();   }
```

More paths ➡ Harder

Hardness should account for # paths.

Paths 3 & 4: drptr not passed to *callee* –memory leak?

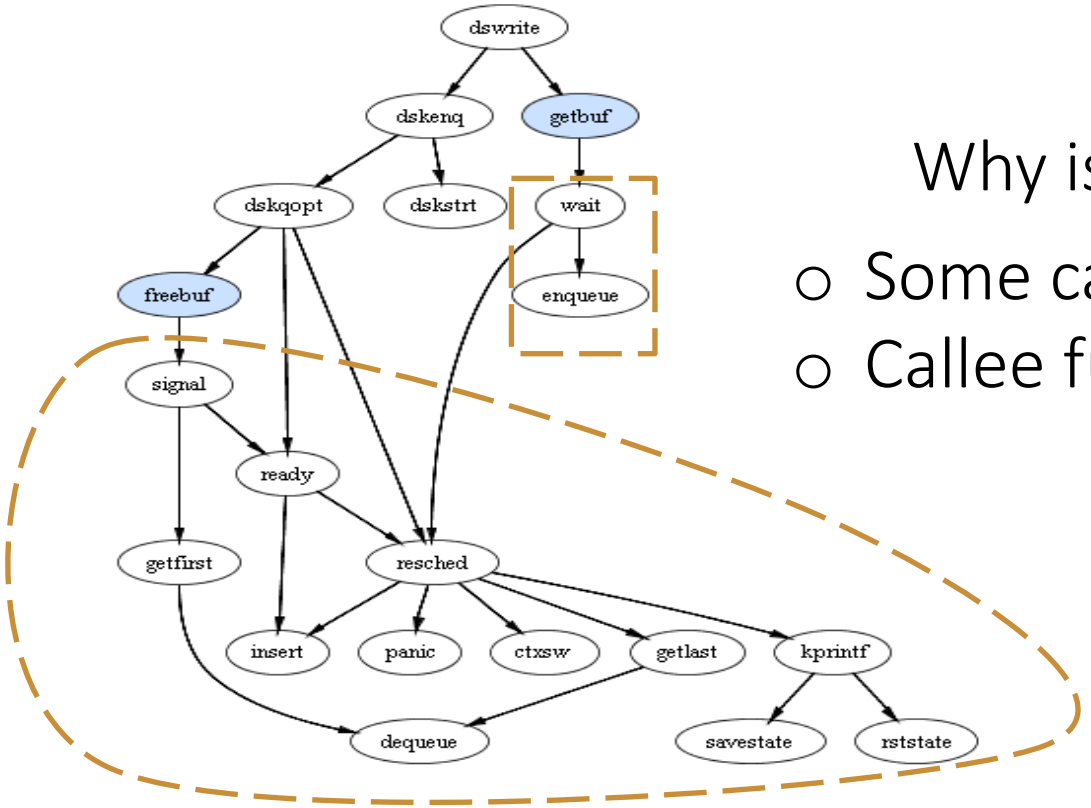
Let's Look a Little Deeper - 3

```
dskenq(drptr, dsptr)
     struct   dreq *drptr;
     struct   dsblk    *dsptr;
     { struct dreq *p, *q;
     if ( (q=dsptr->dreqlst) == DRNULL ) {
          dsptr->dreqlst = drptr;
1         dskstrt(dsptr);
          return();}
     for () {
          if ((st=dskqopt(p, q, drptr)!=SYSERR))
2                 return();
          if (   ) {
               q->drnext = drptr;
3              return();} }
     q->drnext = drptr;
4    return();   }
```

drptr gets passed back to the caller(s) on paths 3 and 4.

Let's Look a Little Deeper - 4

Counting the number of *callee* functions not a good metric to quantify hardness.



Why is it not a good metric?

- Some callee functions *not* relevant.
- Callee functions are not the only relevant functions.

Call graph is neither *necessary* nor *sufficient* to identify *relevant functions.*

Let's Look a Little Deeper - 5

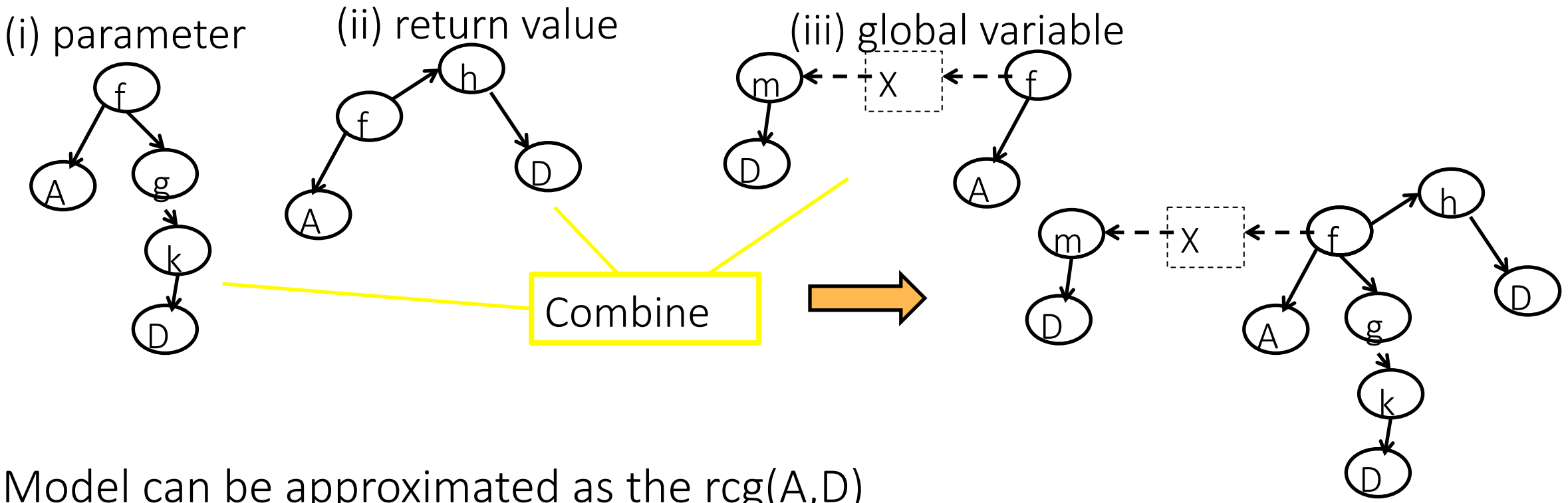
Secondly, it is not just about how many functions interact but also about how functions interact.

Fundamental Interaction Mechanisms:

1. Function `f` passes a pointer as a parameter to a callee function `g`.
2. Function `g` passes a pointer as a parameter or return value back to a caller function `g`.
3. Function `f` assigns a pointer to a global variable `D`.

Does one mechanism make verification harder than the other? Why?

A comprehensive model of communication among functions



Model can be approximated as the rcg(A,D)

A: memory allocation function, D: deallocation function

rcg(A,D): the *reverse call graph* with A and D as leaves

Exploration experiments using Atlas

At this point, let us explore using Atlas.

Explorations will be about:

1. Identifying the relevant functions and their interactions.
2. Addressing the challenge of multitude of paths and the need to check their feasibility.

Some queries for the exploration

Background: Memory is allocated for a structure of type dreq and drptr points to a structure of type dreq. getbuf and freebuf respectively *allocate* and *deallocate* memory.

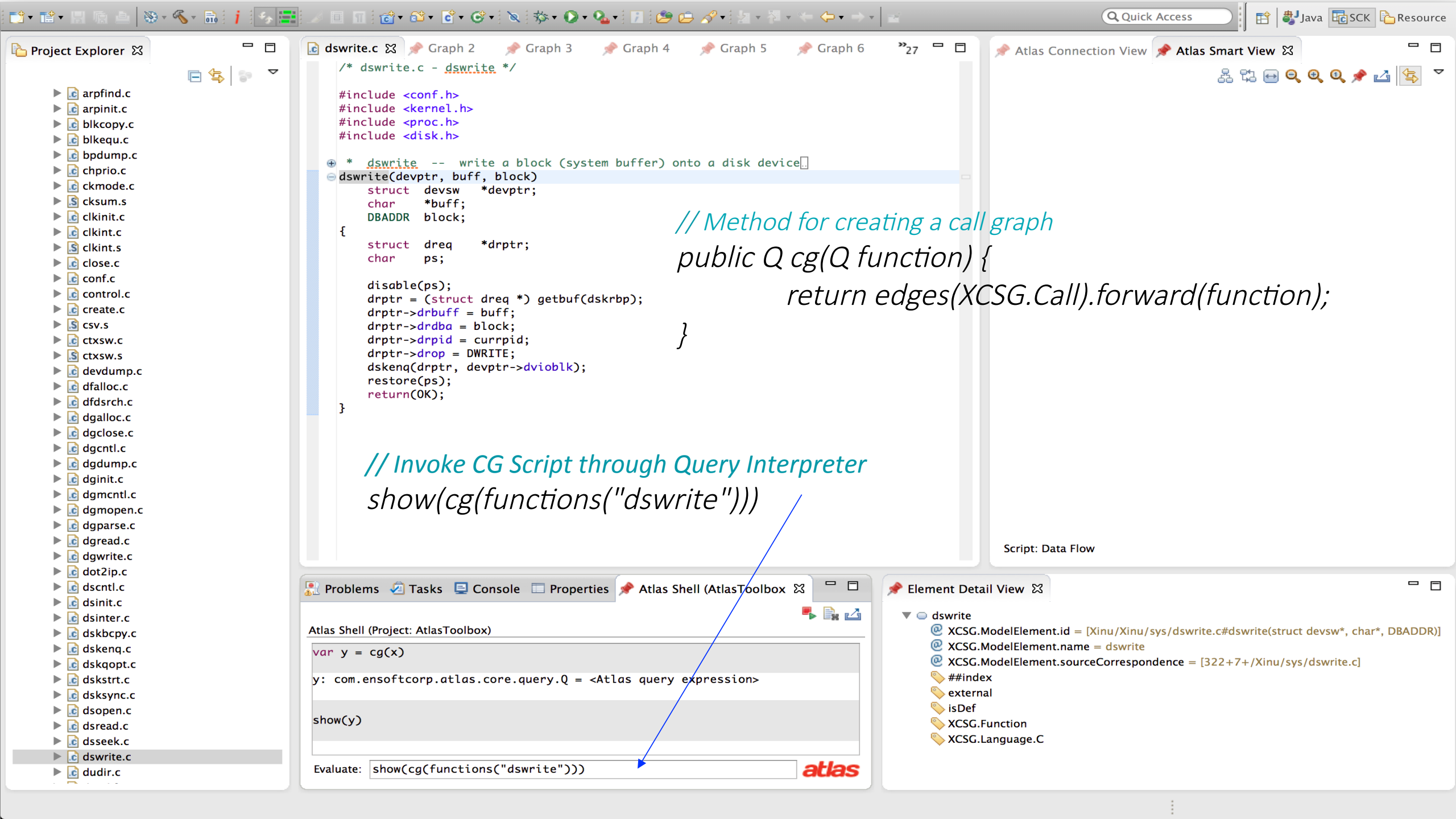
Queries:

1. Find functions which pass drptr to callee as a parameter, along with call relationships.
2. Find functions which pass drptr back to caller as a parameter, along with call relationships.
3. Find functions that write drptr to a global variable.
4. Find functions that read drptr from a global variable.
5. Find functions that have a call chain reaching to either getbuf or freebuf.
6. Find the union of the above, along with call relationships.
7. Find functions that have: (i) a call chain reaching to either getbuf or freebuf, and (ii) reference (read or write) drptr.

A. Will #6 and #7 give the same answer? Why or why not?
B. Is either of the answer necessary and/or sufficient for verifying the memory leak?

Interactive use of Atlas

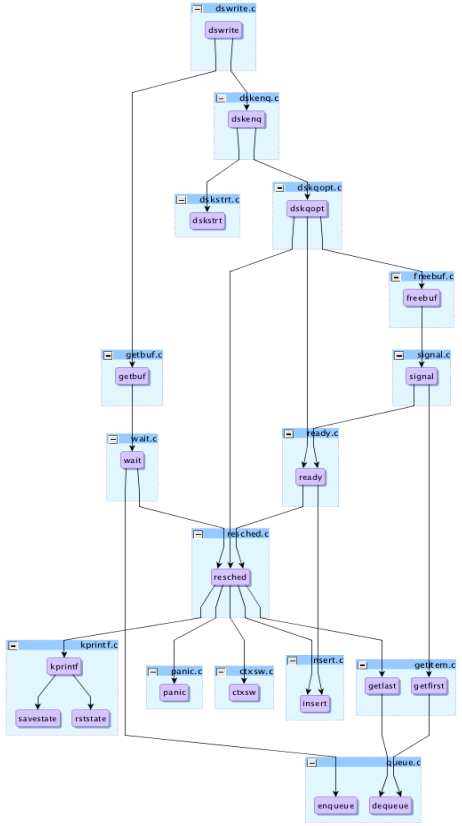
- We explore visual models interactively.
- We will show an example of how interactive experiments can be useful for discovering new ideas and invent new visual models.
- The Lab covers the Atlas features:
  - Querying through Interpreter
  - Querying by referencing selecting elements
  - Smart Views
- The lab covers the visual models: Call Graph (CG), Reverse Call Graph (RCG), Control Flow Graph (CFG), Data Flow Graph (DFG), Program Slices, and Matching Pair Graph (MPG).


Project Explorer
arpfind.c
arpinit.c
blkcopy.c
blkequ.c
bpdump.c
chprio.c
ckmode.c
cksum.s
clkinit.c
clkint.c
clkint.s
close.c
conf.c
control.c
create.c
csv.s
ctxsw.c
ctxsw.s
devdump.c
dfalloc.c
dfdsrch.c
dgalloc.c
dgclose.c
dgcntl.c
dgdump.c
dginit.c
dgmcntl.c
dgmopen.c
dgparse.c
dgread.c
dgwrite.c
dot2ip.c
dscntl.c
dsinit.c
dsinter.c
dskbcpy.c
dskenq.c
dskqopt.c
dskstrt.c
dsksync.c
dsopen.c
dsread.c
dsseek.c
dswrite.c
dudir.c
dswrite.c
Graph 2
Graph 3
Graph 4
Graph 5
Graph 6
/* dswrite.c - dswrite */
#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <disk.h>
* dswrite -- write a block (system buffer) onto a disk device
dswrite(devptr, buff, block)
struct devsw *devptr;
char *buff;
DBADDR block;
{
struct dreq *drptr;
char ps;
disable(ps);
drptr = (struct dreq *) getbuf(dskrbp);
drptr->drbuff = buff;
drptr->drdba = block;
drptr->drpid = currpid;
drptr->drop = DWRITE;
dskenq(drptr, devptr->dvioblk);
restore(ps);
return(OK);
}
// Method for creating a call graph
public Q cg(Q function) {
return edges(XCSG.Call).forward(function);
}
// Invoke CG Script through Query Interpreter
show(cg(functions("dswrite")))
Atlas Connection View
Atlas Smart View
Script: Data Flow
Problems
Tasks
Console
Properties
Atlas Shell (AtlasToolbox)
Atlas Shell (Project: AtlasToolbox)
var y = cg(x)
y: com.ensoftcorp.atlas.core.query.Q = <Atlas query expression>
show(y)
Evaluate:
show(cg(functions("dswrite")))
atlas
Element Detail View
dswrite
XCSG.ModelElement.id = [Xinu/Xinu/sys/dswrite.c#dswrite(struct devsw*, char*, DBADDR)]
XCSG.ModelElement.name = dswrite
XCSG.ModelElement.sourceCorrespondence = [322+7+/Xinu/sys/dswrite.c]
##index
external
isDef
XCSG.Function
XCSG.Language.C

Project Explorer

- arpfind.c
- arpinit.c
- blkcopy.c
- blkequ.c
- bpdump.c
- chprio.c
- ckmode.c
- cksum.s
- clkinit.c
- clkint.c
- clkint.s
- close.c
- conf.c
- control.c
- create.c
- csv.s
- ctxsw.c
- ctxsw.s
- devdump.c
- dfalloc.c
- dfdsrch.c
- dgalloc.c
- dgclose.c
- dgcntl.c
- dgdump.c
- dginit.c
- dgmcntl.c
- dgmopen.c
- dgparse.c
- dgread.c
- dgwrite.c
- dot2ip.c
- dscntl.c
- dsinit.c
- dsinter.c
- dskbcpy.c
- dskenq.c
- dskqopt.c
- dskstrt.c
- dsksync.c
- dsopen.c
- dsread.c
- dsseek.c
- dswrite.c
- dudir.c

Graph 3 | Graph 4 | Graph 5 | Graph 6 | x_net.c | Graph 7 | 28



Resulting Call Graph (CG)

Atlas Connection View | Atlas Smart View

Script: Data Flow

Problems | Tasks | Console | Properties | Atlas Shell (AtlasToolbox

Atlas Shell (Project: AtlasToolbox)

```
show(y)

show(cg(functions("dswrite")))
```

Evaluate: <type an expression or enter ":help" for more information>

Element Detail View

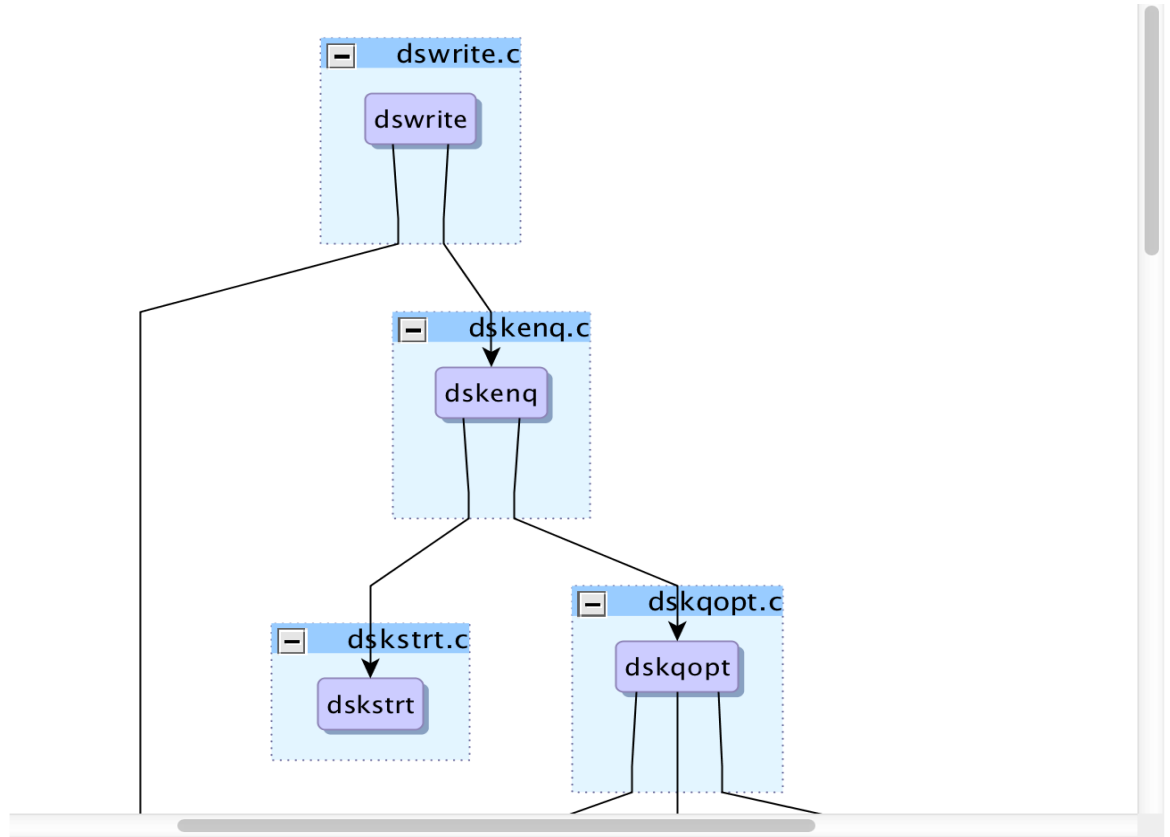
- dswrite
  - XCSG.ModelElement.id = [Xinu/Xinu/sys/dswrite.c#dswrite(struct devsw*, char*, DBADDR)]
  - XCSG.ModelElement.name = dswrite
  - XCSG.ModelElement.sourceCorrespondence = [322+7+/Xinu/sys/dswrite.c]
  - ##index
  - external
  - isDef
  - XCSG.Function
  - XCSG.Language.C

Quick Access | Java | SCK | Resource

Project Explorer

- arpfind.c
- arpinit.c
- blkcopy.c
- blkequ.c
- bpdump.c
- chprio.c
- ckmode.c
- cksum.s
- clkinit.c
- clkint.c
- clkint.s
- close.c
- conf.c
- control.c
- create.c
- csv.s
- ctxsw.c
- ctxsw.s
- devdump.c
- dfalloc.c
- dfdsrch.c
- dgalloc.c
- dgclose.c
- dgcntl.c
- dgdump.c
- dginit.c
- dgmcntl.c
- dgmopen.c
- dgparse.c
- dgread.c
- dgwrite.c
- dot2ip.c
- dscntl.c
- dsinit.c
- dsinter.c
- dskbcpy.c
- dskenq.c
- dskqopt.c
- dskstrt.c
- dsksync.c
- dsopen.c
- dsread.c
- dsseek.c
- dswrite.c
- dudir.c

Graph 3 | Graph 4 | Graph 5 | Graph 6 | x_net.c | Graph 7 | 28



Atlas Connection View | Atlas Smart View

Zoom on the CG

Script: Data Flow

Problems | Tasks | Console | Properties | Atlas Shell (AtlasToolbox

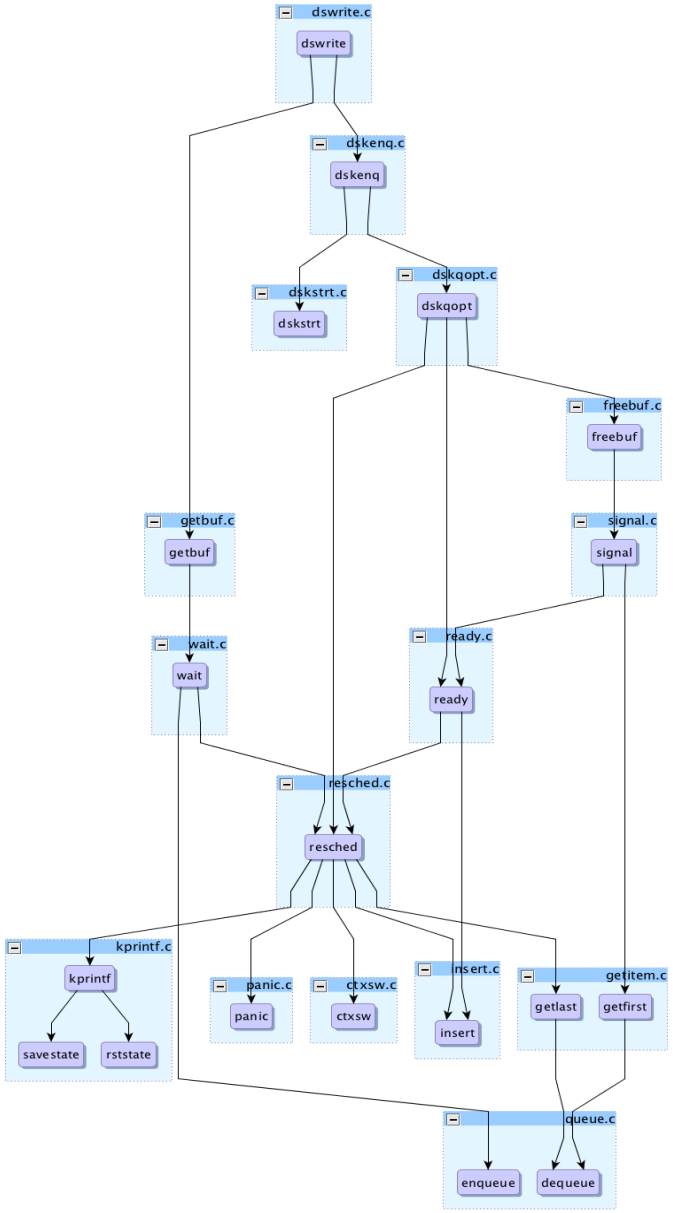
Atlas Shell (Project: AtlasToolbox)

```
show(y)

show(cg(functions("dswrite")))
```

Evaluate: <type an expression or enter ":help" for more information>

Element Detail View

- dswrite
  - XCSG.ModelElement.id = [Xinu/Xinu/sys/dswrite.c#dswrite(struct devsw*, char*, DBADDR)]
  - XCSG.ModelElement.name = dswrite
  - XCSG.ModelElement.sourceCorrespondence = [322+7+/Xinu/sys/dswrite.c]
  - ##index
  - external
  - isDef
  - XCSG.Function
  - XCSG.Language.C

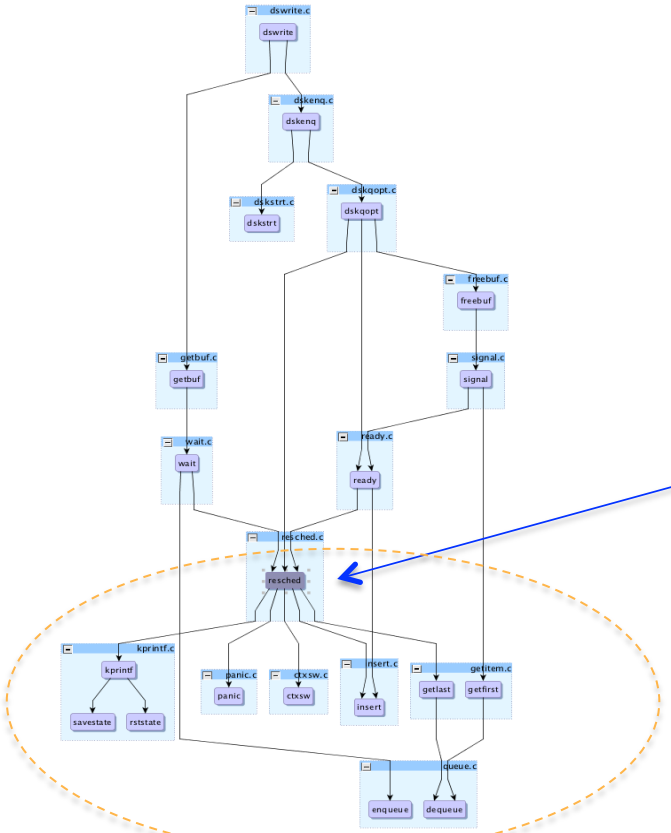


Expand the window

Project Explorer

- arpfind.c
- arpinit.c
- blkcopy.c
- blkequ.c
- bpdump.c
- chprio.c
- ckmode.c
- cksum.s
- clkinit.c
- clkint.c
- clkint.s
- close.c
- conf.c
- control.c
- create.c
- csv.s
- ctxsw.c
- ctxsw.s
- devdump.c
- dfalloc.c
- dfdsrch.c
- dgalloc.c
- dgclose.c
- dgcntl.c
- dgdump.c
- dginit.c
- dgmcntl.c
- dgmopen.c
- dgparse.c
- dgread.c
- dgwrite.c
- dot2ip.c
- dscntl.c
- dsinit.c
- dsinter.c
- dskbcpy.c
- dskenq.c
- dskqopt.c
- dskstrt.c
- dsksync.c
- dsopen.c
- dsread.c
- dsseek.c
- dswrite.c
- dudir.c

Graph 7 | Graph 8 | Graph 9 | Graph 10 | Graph 11



Atlas Connection View | Atlas Smart View

Queries by selecting model elements

Select a node from CG

show(cg(selected))

Script: Data Flow

Problems | Tasks | Console | Properties | Atlas Shell (AtlasToolbox)

Atlas Shell (Project: AtlasToolbox)

```
show(cg(selected))

show(cg(functions("dswrite")))
```

Evaluate: <type an expression or enter ":help" for more information>

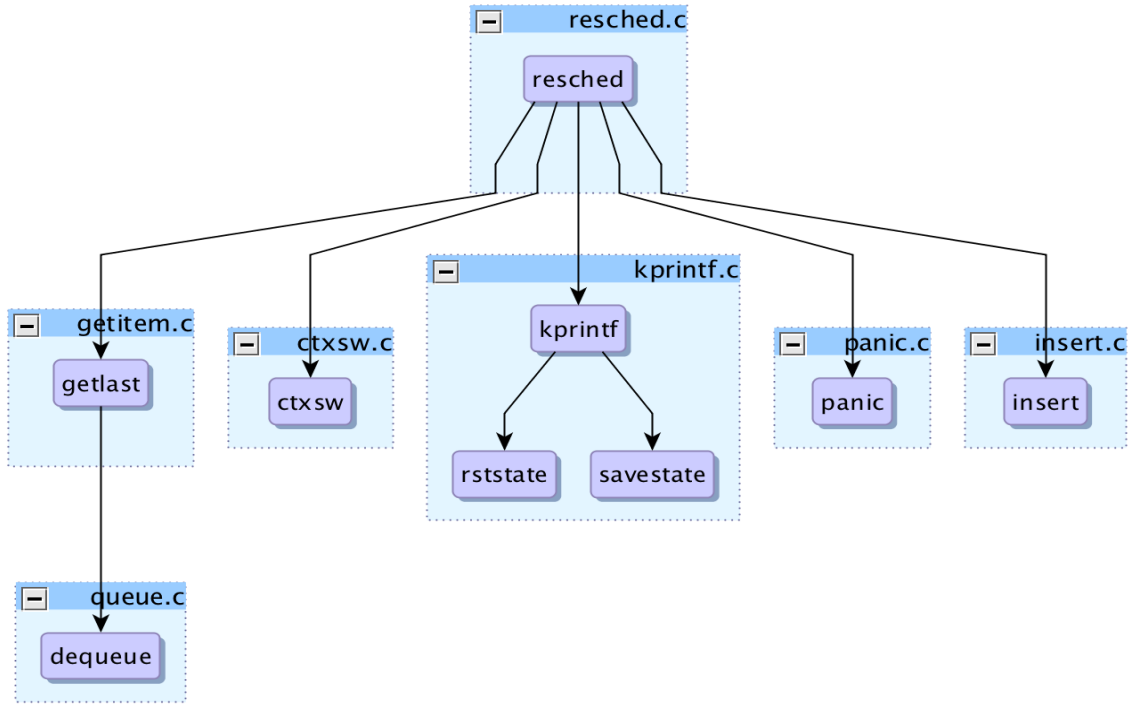
Element Detail View

- resched
  - XCSG.ModelElement.id = [Xinu/Xinu/sys/resched.c#resched()]
  - XCSG.ModelElement.name = resched
  - XCSG.ModelElement.sourceCorrespondence = [489+7+/Xinu/sys/resched.c]
  - ##index
  - external
  - isDef
  - XCSG.Function
  - XCSG.Language.C

Project Explorer

- arpfind.c
- arpinit.c
- blkcopy.c
- blkequ.c
- bpdump.c
- chprio.c
- ckmode.c
- cksum.s
- clkinit.c
- clkint.c
- clkint.s
- close.c
- conf.c
- control.c
- create.c
- csv.s
- ctxsw.c
- ctxsw.s
- devdump.c
- dfalloc.c
- dfdsrch.c
- dgalloc.c
- dgclose.c
- dgcntl.c
- dgdump.c
- dginit.c
- dgmcntl.c
- dgmopen.c
- dgparse.c
- dgread.c
- dgwrite.c
- dot2ip.c
- dscntl.c
- dsinit.c
- dsinter.c
- dskbcpy.c
- dskenq.c
- dskqopt.c
- dskstrt.c
- dsksync.c
- dsopen.c
- dsread.c
- dsseek.c
- dswrite.c
- dudir.c

Graph 8 | Graph 9 | Graph 10 | Graph 11 | Graph 12 | 34



Atlas Connection View | Atlas Smart View

CG of the selected node

Script: Data Flow

Problems | Tasks | Console | Properties | Atlas Shell (AtlasToolbox)

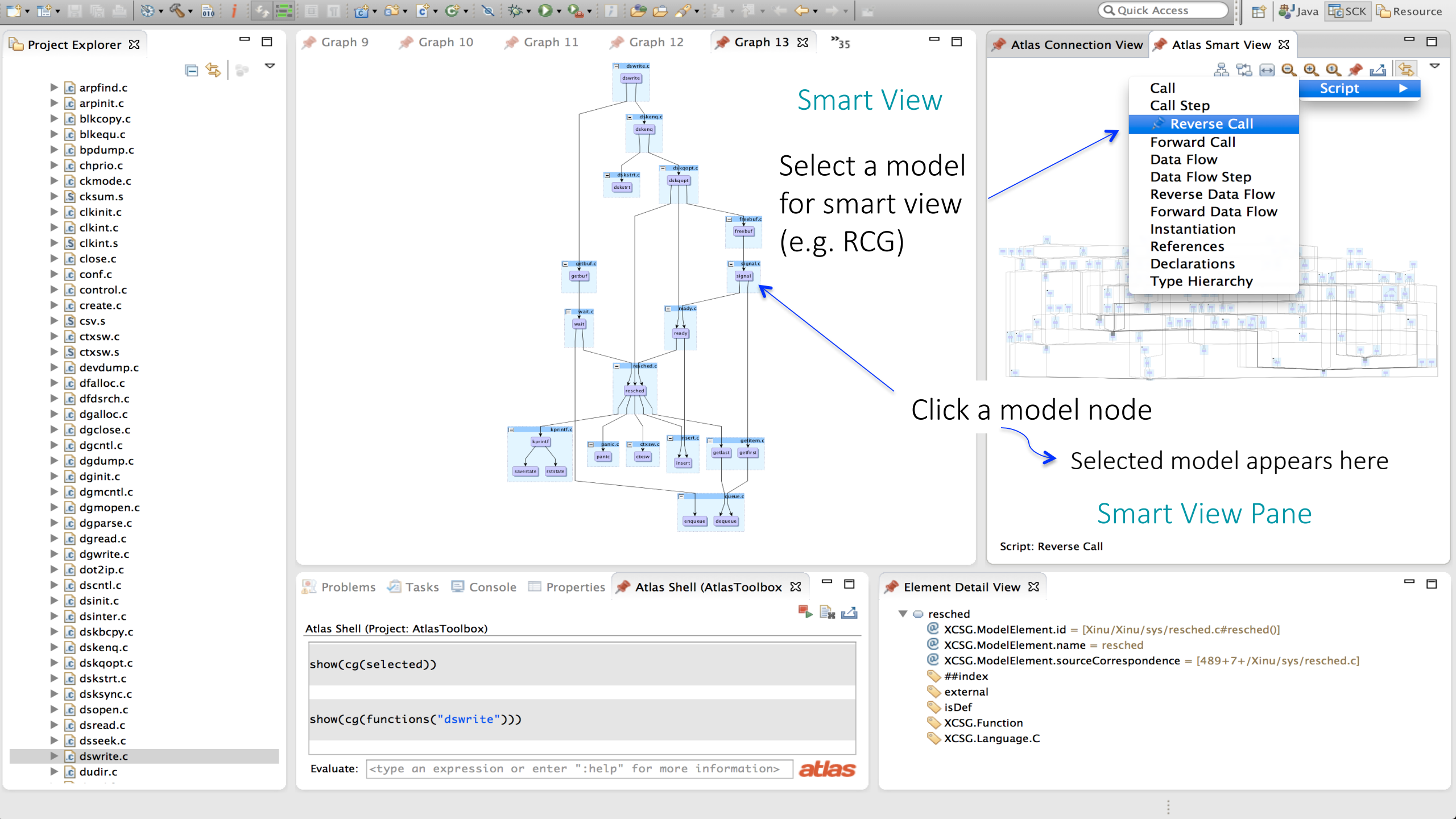
Atlas Shell (Project: AtlasToolbox)

```
show(cg(functions("dswrite")))

show(cg(selected))
```

Evaluate: <type an expression or enter ":help" for more information>

Element Detail View

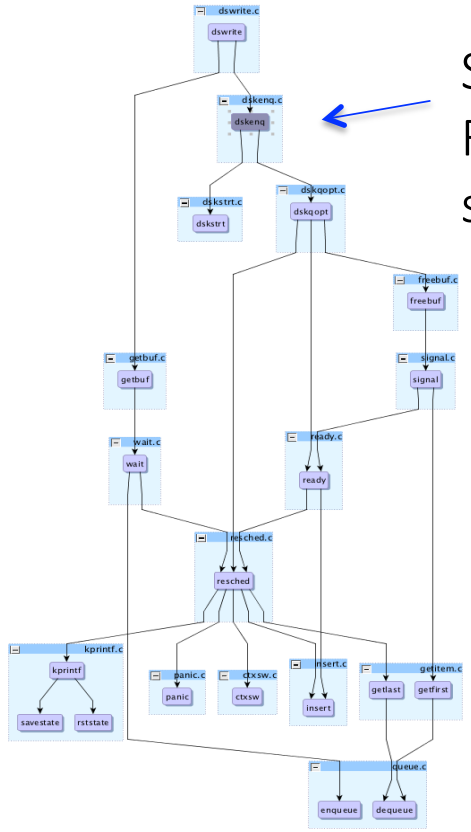
- resched
  - XCSG.ModelElement.id = [Xinu/Xinu/sys/resched.c#resched()]
  - XCSG.ModelElement.name = resched
  - XCSG.ModelElement.sourceCorrespondence = [489+7+/Xinu/sys/resched.c]
  - ##index
  - external
  - isDef
  - XCSG.Function
  - XCSG.Language.C


Project Explorer
arpfind.c
arpinit.c
blkcopy.c
blkequ.c
bpdump.c
chprio.c
ckmode.c
cksum.s
clkinit.c
clkint.c
clkint.s
close.c
conf.c
control.c
create.c
csv.s
ctxsw.c
ctxsw.s
devdump.c
dfalloc.c
dfdsrch.c
dgalloc.c
dgclose.c
dgcntl.c
dgdump.c
dginit.c
dgmcntl.c
dgmopen.c
dgparse.c
dgread.c
dgwrite.c
dot2ip.c
dscntl.c
dsinit.c
dsinter.c
dskbcpy.c
dskenq.c
dskqopt.c
dskstrt.c
dsksync.c
dsopen.c
dsread.c
dsseek.c
dswrite.c
dudir.c
Graph 9
Graph 10
Graph 11
Graph 12
Graph 13
Smart View
Select a model for smart view (e.g. RCG)
Atlas Connection View
Atlas Smart View
Script
Call
Call Step
Reverse Call
Forward Call
Data Flow
Data Flow Step
Reverse Data Flow
Forward Data Flow
Instantiation
References
Declarations
Type Hierarchy
Click a model node
Selected model appears here
Smart View Pane
Script: Reverse Call
Problems
Tasks
Console
Properties
Atlas Shell (AtlasToolbox
Atlas Shell (Project: AtlasToolbox)
show(cg(selected))
show(cg(functions("dswrite")))
Evaluate:
<type an expression or enter ":help" for more information>
atlas
Element Detail View
resched
XCSG.ModelElement.id = [Xinu/Xinu/sys/resched.c#resched()]
XCSG.ModelElement.name = resched
XCSG.ModelElement.sourceCorrespondence = [489+7+/Xinu/sys/resched.c]
##index
external
isDef
XCSG.Function
XCSG.Language.C

Project Explorer

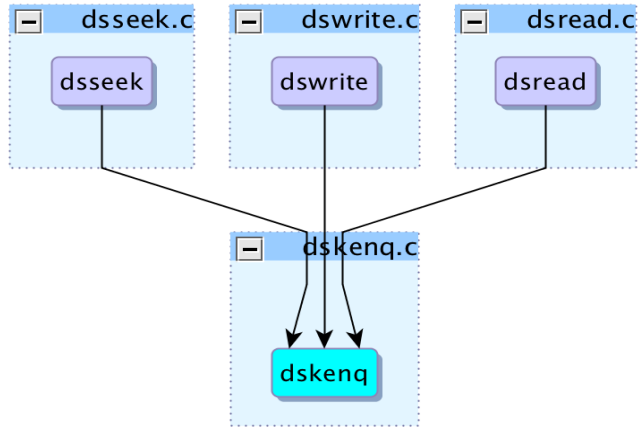
- arpfind.c
- arpinit.c
- blkcopy.c
- blkequ.c
- bpdump.c
- chprio.c
- ckmode.c
- cksum.s
- clkinit.c
- clkint.c
- clkint.s
- close.c
- conf.c
- control.c
- create.c
- csv.s
- ctxsw.c
- ctxsw.s
- devdump.c
- dfalloc.c
- dfdsrch.c
- dgalloc.c
- dgclose.c
- dgcntl.c
- dgdump.c
- dginit.c
- dgmcntl.c
- dgmopen.c
- dgparse.c
- dgread.c
- dgwrite.c
- dot2ip.c
- dscntl.c
- dsinit.c
- dsinter.c
- dskbcpy.c
- dskenq.c
- dskqopt.c
- dskstrt.c
- dsksync.c
- dsopen.c
- dsread.c
- dsseek.c
- dswrite.c
- dudir.c

Graph 9 | Graph 10 | Graph 11 | Graph 12 | Graph 13



Smart view generates RCGs on the fly as we select different nodes

Atlas Connection View | Atlas Smart View



Script: Reverse Call

Problems | Tasks | Console | Properties | Atlas Shell (AtlasToolbox

Atlas Shell (Project: AtlasToolbox)

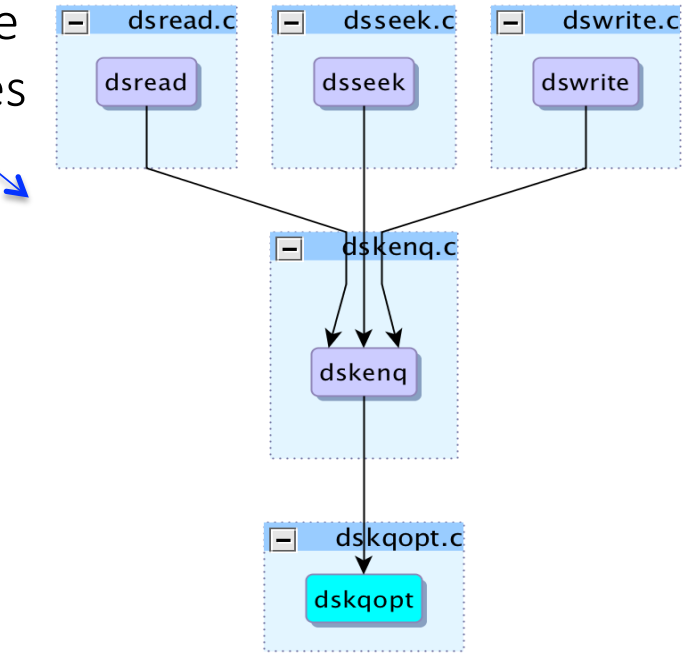
```
show(cg(selected))

show(cg(functions("dswrite")))
```

Evaluate: <type an expression or enter ":help" for more information>

Element Detail View

- dskenq
  - XCSG.ModelElement.id = [Xinu/Xinu/sys/dskenq.c#dskenq(struct dreq*, struct dsblk*)]
  - XCSG.ModelElement.name = dskenq
  - XCSG.ModelElement.sourceCorrespondence = [306+6+/Xinu/sys/dskenq.c]
  - ##index
  - external
  - isDef
  - XCSG.Function
  - XCSG.Language.C

Smart view generates RCGs on the fly as we select different nodes



Script: Reverse Call

Atlas Shell (Project: AtlasToolbox)

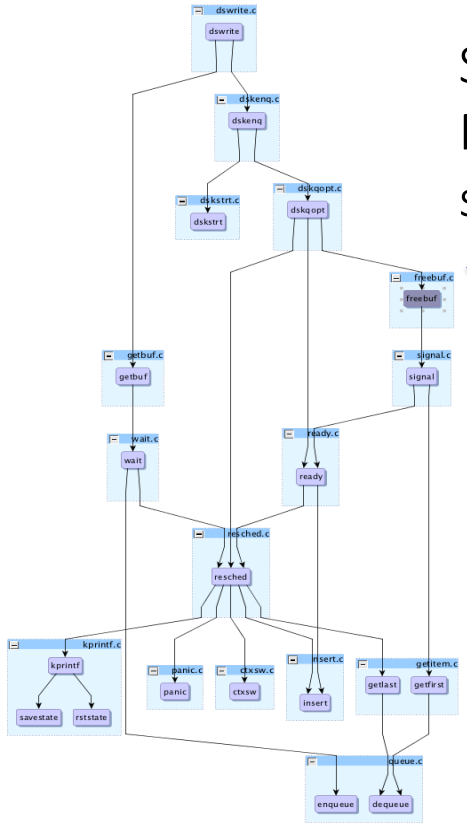
```
show(cg(selected))

show(cg(functions("dswrite")))
```

Evaluate: <type an expression or enter ":help" for more information>

Project Explorer

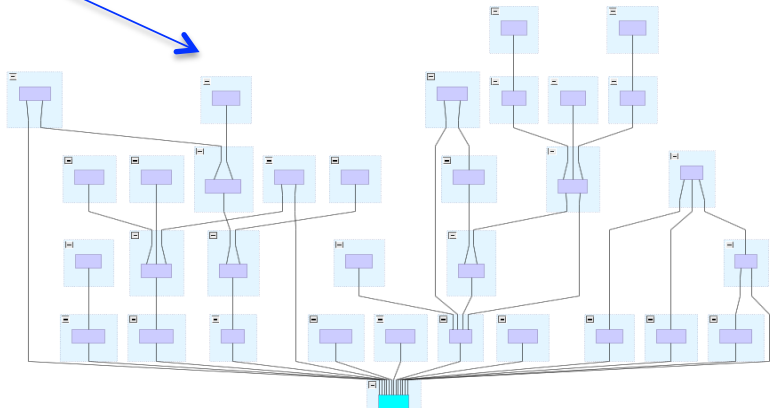
- arpfind.c
- arpinit.c
- blkcopy.c
- blkequ.c
- bpdump.c
- chprio.c
- ckmode.c
- cksum.s
- clkinit.c
- clkint.c
- clkint.s
- close.c
- conf.c
- control.c
- create.c
- csv.s
- ctxsw.c
- ctxsw.s
- devdump.c
- dfalloc.c
- dfdsrch.c
- dgalloc.c
- dgclose.c
- dgcntl.c
- dgdump.c
- dginit.c
- dgmcntl.c
- dgmopen.c
- dgparse.c
- dgread.c
- dgwrite.c
- dot2ip.c
- dscntl.c
- dsinit.c
- dsinter.c
- dskbcpy.c
- dskenq.c
- dskqopt.c
- dskstrt.c
- dsksync.c
- dsopen.c
- dsread.c
- dsseek.c
- dswrite.c
- dudir.c

Graph 9 | Graph 10 | Graph 11 | Graph 12 | Graph 13



Smart view generates RCGs on the fly as we select different nodes

Atlas Connection View | Atlas Smart View



Experiment and invent new visual models:

1. As we are viewing RCGs for different nodes, we discover a node with a large RCG.
2. What does a large RCG tell about the node?
3. How can we use that knowledge to refine visual models?

Problems | Tasks | Console | Properties | Atlas Shell (AtlasToolbox

Atlas Shell (Project: AtlasToolbox)

```
show(cg(selected))

show(cg(functions("dswrite")))
```

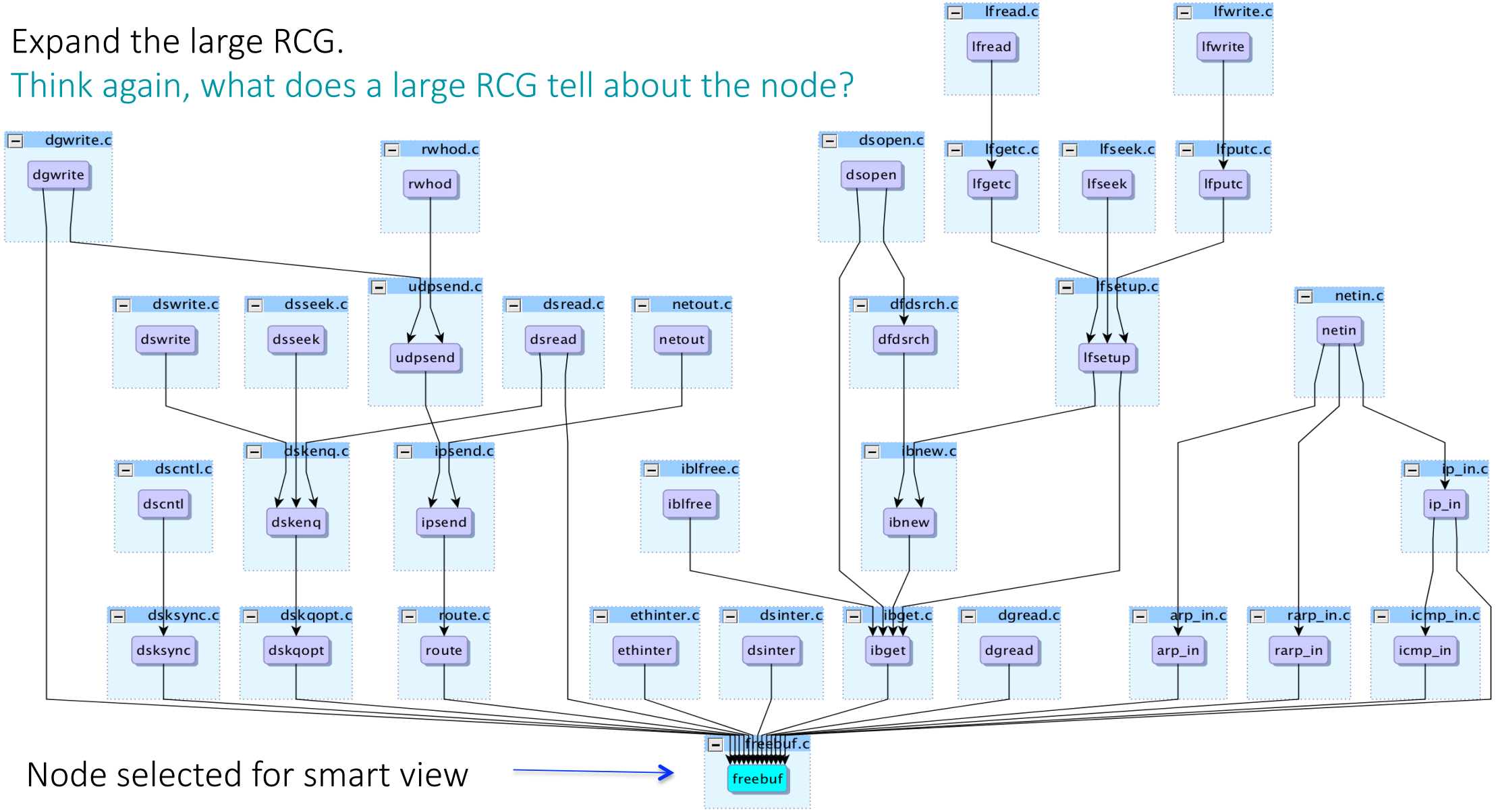
Evaluate: <type an expression or enter ":help" for more information>

Element Detail View

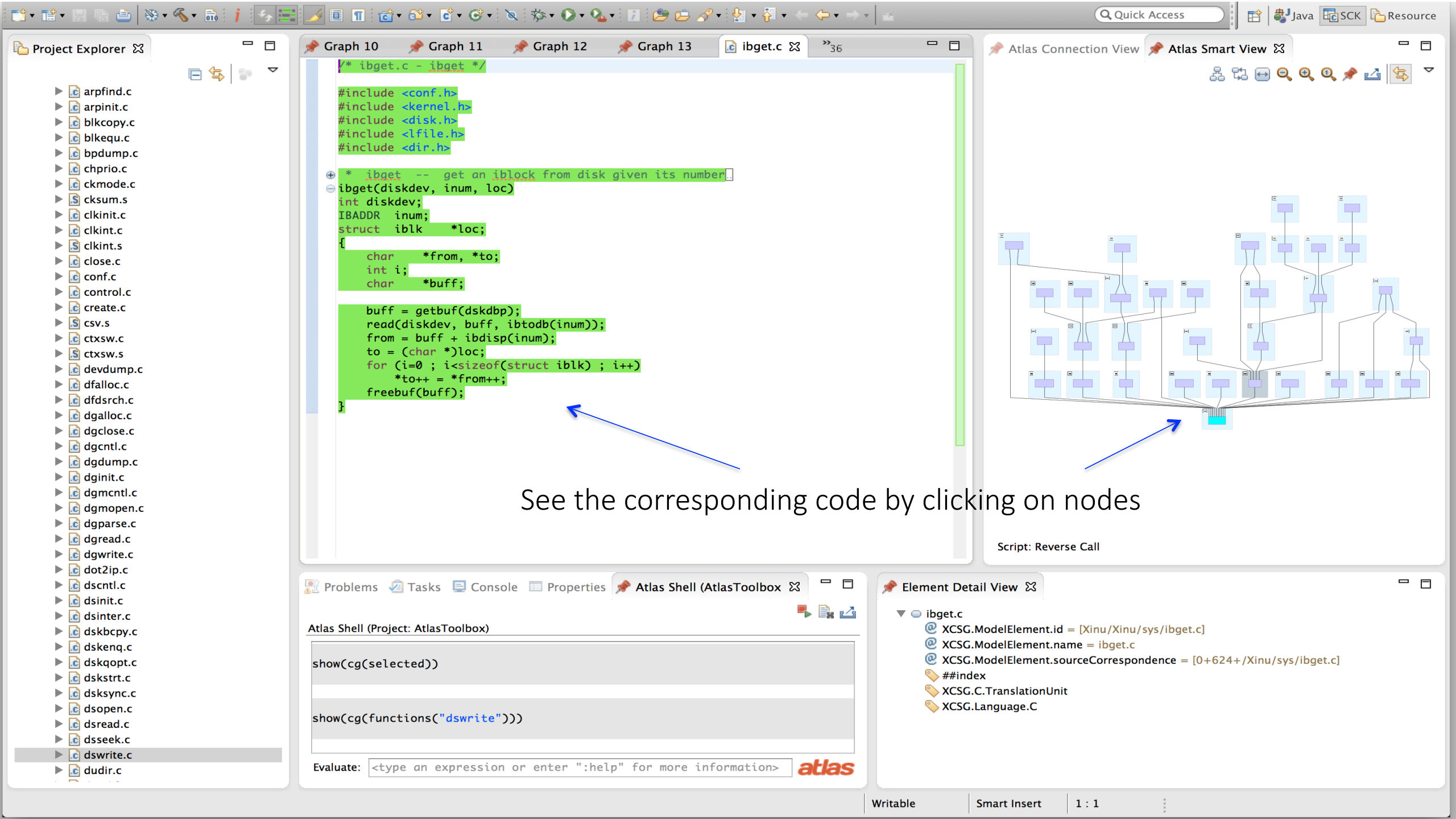
- freebuf
  - XCSG.ModelElement.id = [Xinu/Xinu/sys/freebuf.c#freebuf(int*)]
  - XCSG.ModelElement.name = freebuf
  - XCSG.ModelElement.sourceCorrespondence = [339+7+/Xinu/sys/freebuf.c]
  - ##index
  - external
  - isDef
  - XCSG.Function
  - XCSG.Language.C

Expand the large RCG.

Think again, what does a large RCG tell about the node?



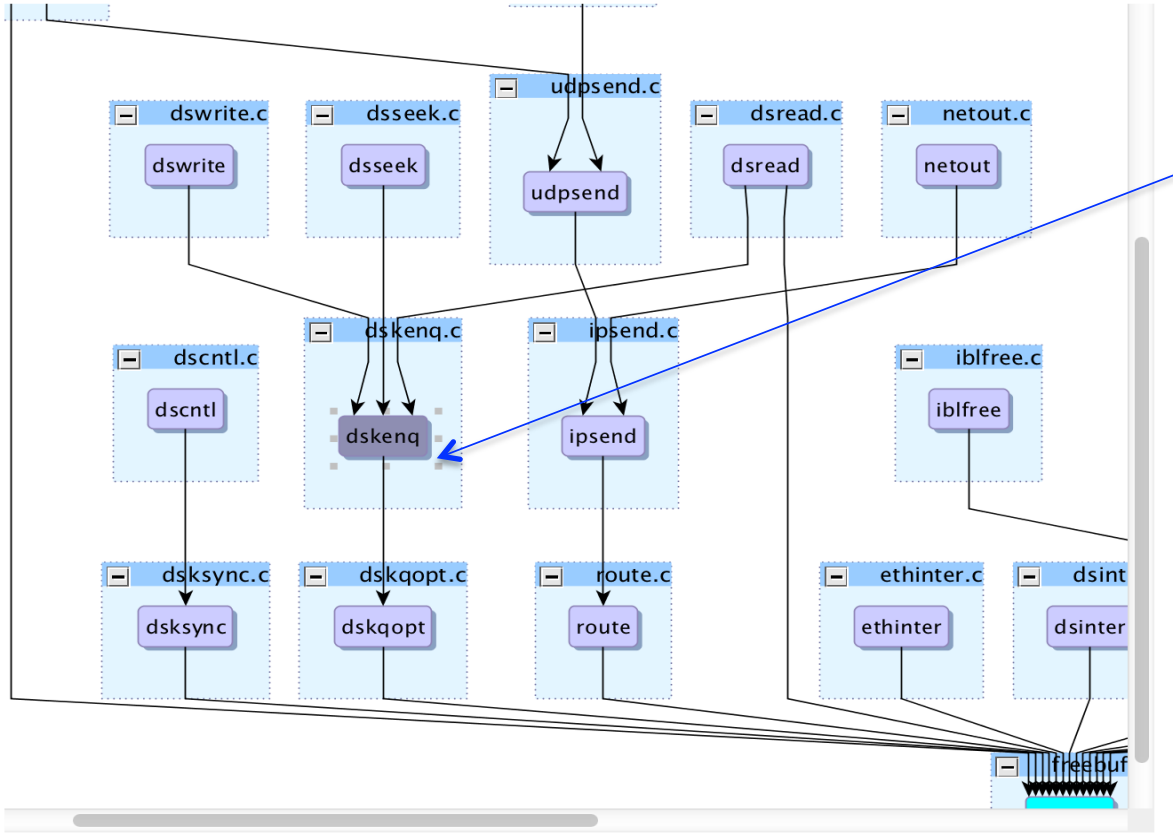
Node selected for smart view


Project Explorer
arpfind.c
arpinit.c
blkcopy.c
blkequ.c
bpdump.c
chprio.c
ckmode.c
cksum.s
clkinit.c
clkint.c
clkint.s
close.c
conf.c
control.c
create.c
csv.s
ctxsw.c
ctxsw.s
devdump.c
dfalloc.c
dfdsrch.c
dgalloc.c
dgclose.c
dgcntl.c
dgdump.c
dginit.c
dgmcntl.c
dgmopen.c
dgparse.c
dgread.c
dgwrite.c
dot2ip.c
dscntl.c
dsinit.c
dsinter.c
dskbcpy.c
dskenq.c
dskqopt.c
dskstrt.c
dsksync.c
dsopen.c
dsread.c
dsseek.c
dswrite.c
dudir.c
Graph 10
Graph 11
Graph 12
Graph 13
ibget.c
/* ibget.c - ibget */
#include <conf.h>
#include <kernel.h>
#include <disk.h>
#include <lfile.h>
#include <dir.h>
* ibget -- get an iblock from disk given its number
ibget(diskdev, inum, loc)
int diskdev;
IBADDR inum;
struct iblk *loc;
{
char *from, *to;
int i;
char *buff;
buff = getbuf(dskdbp);
read(diskdev, buff, ibtodb(inum));
from = buff + ibdisp(inum);
to = (char *)loc;
for (i=0 ; i<sizeof(struct iblk) ; i++)
*to++ = *from++;
freebuf(buff);
}
Atlas Connection View
Atlas Smart View
Script: Reverse Call
See the corresponding code by clicking on nodes
Problems
Tasks
Console
Properties
Atlas Shell (AtlasToolbox
Atlas Shell (Project: AtlasToolbox)
show(cg(selected))
show(cg(functions("dswrite")))
Evaluate:
<type an expression or enter ":help" for more information>
atlas
Element Detail View
ibget.c
XCSG.ModelElement.id = [Xinu/Xinu/sys/ibget.c]
XCSG.ModelElement.name = ibget.c
XCSG.ModelElement.sourceCorrespondence = [0+624+/Xinu/sys/ibget.c]
##index
XCSG.C.TranslationUnit
XCSG.Language.C
Writable
Smart Insert
1 : 1

Project Explorer

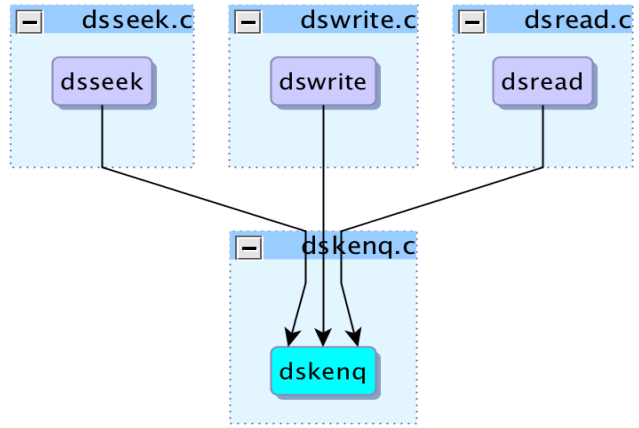
- arpfind.c
- arpinit.c
- blkcopy.c
- blkequ.c
- bpdump.c
- chprio.c
- ckmode.c
- cksum.s
- clkinit.c
- clkint.c
- clkint.s
- close.c
- conf.c
- control.c
- create.c
- csv.s
- ctxsw.c
- ctxsw.s
- devdump.c
- dfalloc.c
- dfdsrch.c
- dgalloc.c
- dgclose.c
- dgcntl.c
- dgdump.c
- dginit.c
- dgmcntl.c
- dgmopen.c
- dgparse.c
- dgread.c
- dgwrite.c
- dot2ip.c
- dscntl.c
- dsinit.c
- dsinter.c
- dskbcpy.c
- dskenq.c
- dskqopt.c
- dskstrt.c
- dsksync.c
- dsopen.c
- dsread.c
- dsseek.c
- dswrite.c
- dudir.c

Graph 11 | Graph 12 | Graph 13 | ibget.c | Graph 14 | 37



Atlas Connection View | Atlas Smart View

Select a node from CG



show(cfg(selected))

Script: Reverse Call

Problems | Tasks | Console | Properties | Atlas Shell (AtlasToolbox)

Atlas Shell (Project: AtlasToolbox)

```
show(cg(selected))

show(cg(functions("dswrite")))
```

Evaluate: show(cfg(selected))

atlas

Element Detail View

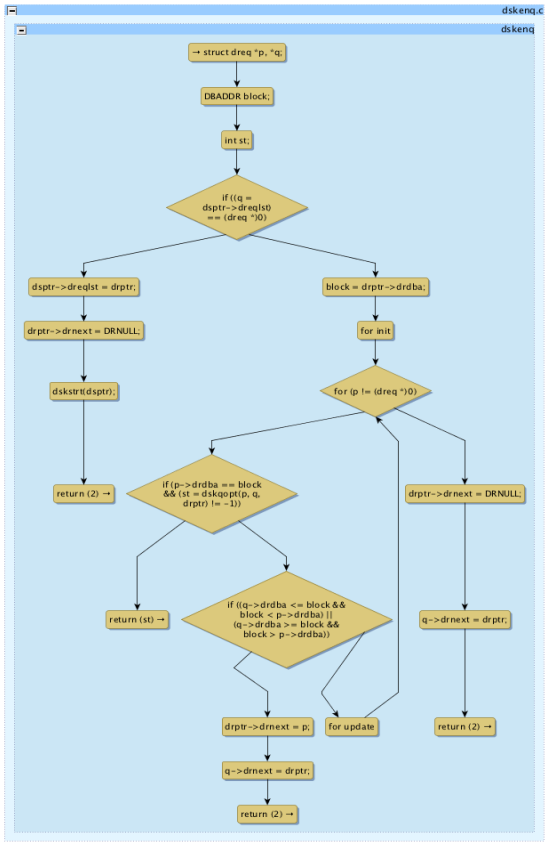
- dskenq
  - XCSG.ModelElement.id = [Xinu/Xinu/sys/dskenq.c#dskenq(struct dreq*, struct dsblk*)]
  - XCSG.ModelElement.name = dskenq
  - XCSG.ModelElement.sourceCorrespondence = [306+6+/Xinu/sys/dskenq.c]
  - ##index
  - external
  - isDef
  - XCSG.Function
  - XCSG.Language.C

Quick Access | Java | SCK | Resource

Project Explorer

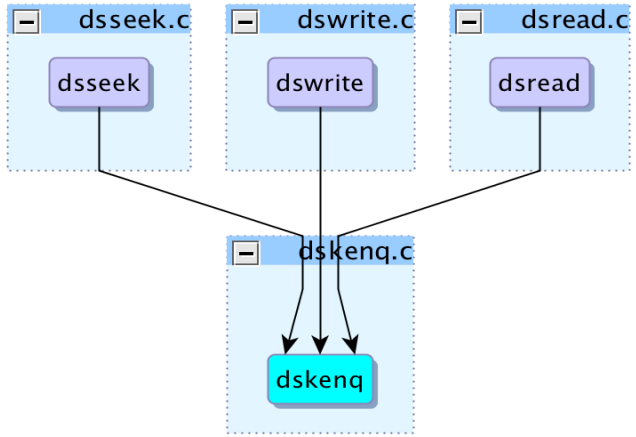
- arpfind.c
- arpinit.c
- blkcopy.c
- blkequ.c
- bpdump.c
- chprio.c
- ckmode.c
- cksum.s
- clkinit.c
- clkint.c
- clkint.s
- close.c
- conf.c
- control.c
- create.c
- csv.s
- ctxsw.c
- ctxsw.s
- devdump.c
- dfalloc.c
- dfdsrch.c
- dgalloc.c
- dgclose.c
- dgcntl.c
- dgdump.c
- dginit.c
- dgmcntl.c
- dgmopen.c
- dgparse.c
- dgread.c
- dgwrite.c
- dot2ip.c
- dscntl.c
- dsinit.c
- dsinter.c
- dskbcpy.c
- dskenq.c
- dskqopt.c
- dskstrt.c
- dsksync.c
- dsopen.c
- dsread.c
- dsseek.c
- dswrite.c
- dudir.c

Graph 12 | Graph 13 | ibget.c | Graph 14 | Graph 15 | 38



Control Flow Graph (CFG) of selected node

Atlas Connection View | Atlas Smart View



Script: Reverse Call

Problems | Tasks | Console | Properties | Atlas Shell (AtlasToolbox)

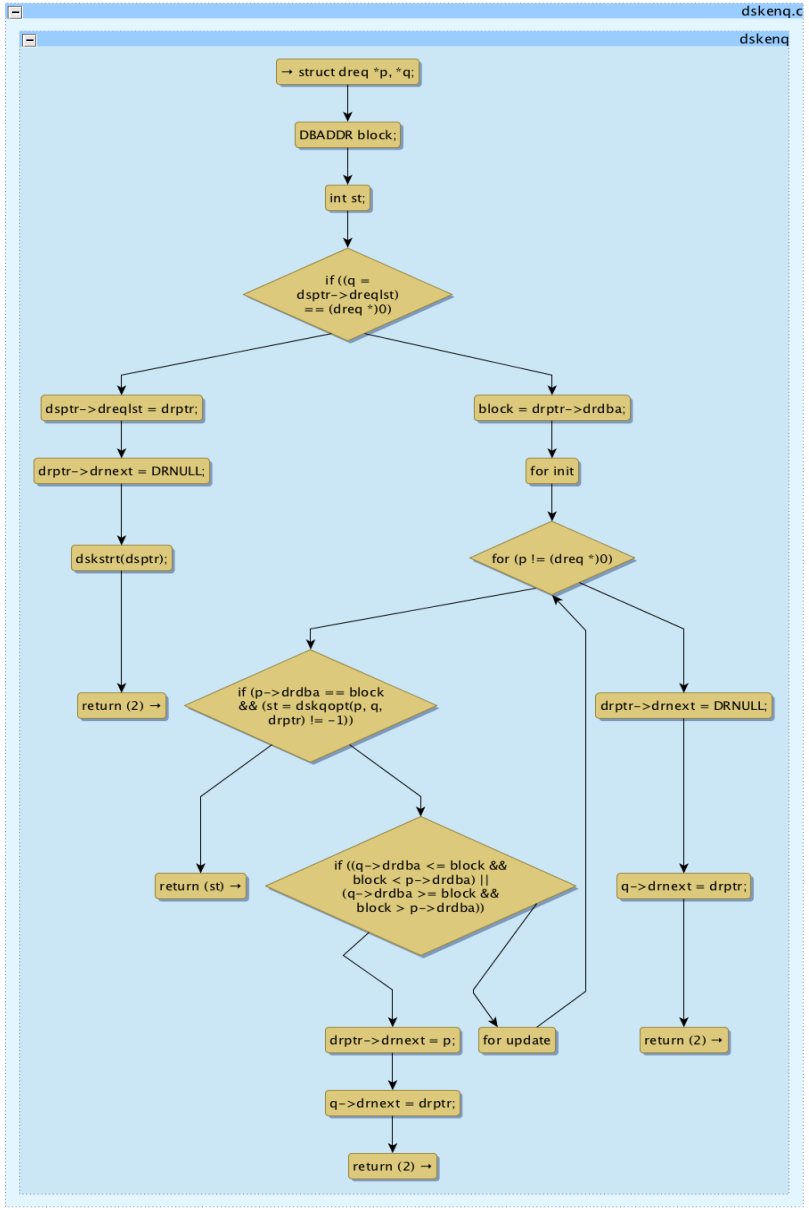
Atlas Shell (Project: AtlasToolbox)

```
show(cg(functions("dswrite")))

show(cfg(selected))
```

Evaluate: <type an expression or enter ":help" for more information>

Element Detail View

- dskenq
  - XCSG.ModelElement.id = [Xinu/Xinu/sys/dskenq.c#dskenq(struct dreq*, struct dsblk*)]
  - XCSG.ModelElement.name = dskenq
  - XCSG.ModelElement.sourceCorrespondence = [306+6+/Xinu/sys/dskenq.c]
  - ##index
  - external
  - isDef
  - XCSG.Function
  - XCSG.Language.C



Expand the window

Project Explorer

- bpdump.c
- chprio.c
- ckmode.c
- cksum.s
- clkinit.c
- clkint.c
- clkint.s
- close.c
- conf.c
- control.c
- create.c
- csv.s
- ctxsw.c
- ctxsw.s
- devdump.c
- dfalloc.c
- dfdsrch.c
- dgalloc.c
- dgclose.c
- dgcntl.c
- dgdump.c
- dginit.c
- dgmcntl.c
- dgmopen.c
- dgparse.c
- dgread.c
- dgwrite.c
- dot2ip.c
- dscntl.c
- dsinit.c
- dsinter.c
- dskbcpy.c
- dskenq.c
- dskqopt.c
- dskstrt.c
- dsksync.c
- dsopen.c
- dsread.c
- dsseek.c
- dswrite.c
- dudir.c
- dumkfs.c
- ethinit.c
- ethinter.c
- ethread.c

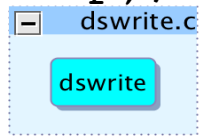
dswrite.c | Graph 2 | Graph 3 | Graph 4 | Graph 5 | Graph 6

```
/* dswrite.c - dswrite */

#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <disk.h>

 *  dswrite  --  write a block (system buffer) onto a disk device
dswrite(devptr, buff, block)
    struct  devsw   *devptr;
    char    *buff;
    DBADDR  block;
{
    struct  dreq    *drptr;
    char    ps;

    disable(ps);
    drptr = (struct dreq *) getbuf(dskrbp);
    drptr->drbuff = buff;
    drptr->drdba = block;
    drptr->drpid = currpid;
    drptr->drop = DWRITE;
    dskenq(drptr, devptr->dvioblk);
    restore(ps);
    return(OK);
}
```

drptr = (struct dreq *) getbuf(dskrbp);

dskenq(drptr, devptr->dvioblk);

Atlas Connection View | Atlas Smart View



Script: Reverse Call

var dswritedfg = dfg("dswrite", "getbuf")

Problems | Tasks | Console | Properties | Atlas Shell (AtlasToolbox)

Atlas Shell (Project: AtlasToolbox)

```
show(projection)

show(forwardSlice(projection,selected))
```

Evaluate: `var dswritedfg = dfg("dswrite", "getbuf")`

Element Detail View

- dswrite
  - XCSG.ModelElement.id = [Xinu/Xinu/sys/dswrite.c#dswrite(struct devsw*, char*, DBADDR)]
  - XCSG.ModelElement.name = dswrite
  - XCSG.ModelElement.sourceCorrespondence = [322+7+/Xinu/sys/dswrite.c]
  - ##index
  - external
  - isDef
  - XCSG.Function
  - XCSG.Language.C

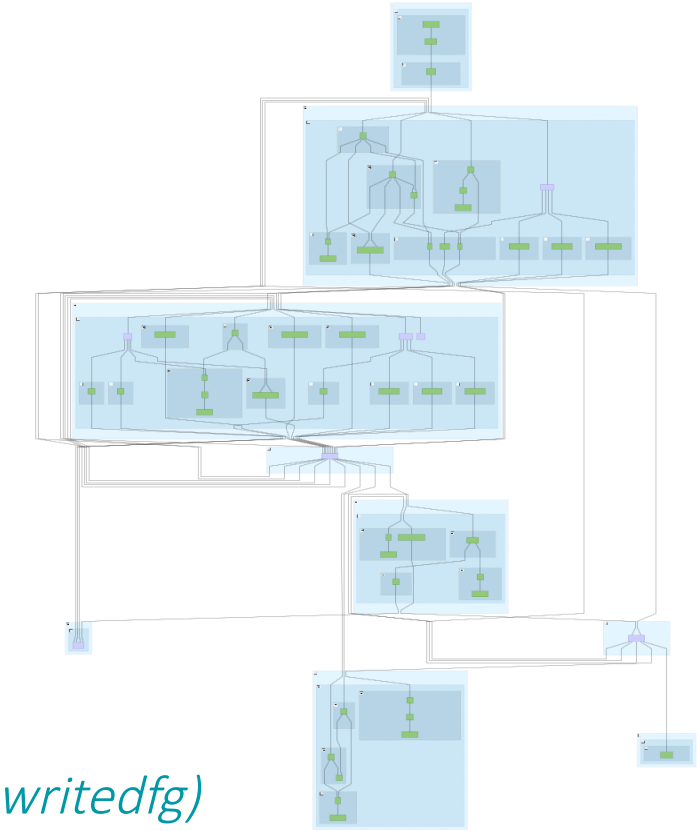
Writable | Smart Insert | 12 : 1

Quick Access | Java | SCK | Resource

Project Explorer

- bpdump.c
- chprio.c
- ckmode.c
- cksum.s
- clkinit.c
- clkint.c
- clkint.s
- close.c
- conf.c
- control.c
- create.c
- csv.s
- ctxsw.c
- ctxsw.s
- devdump.c
- dfalloc.c
- dfdsrch.c
- dgalloc.c
- dgclose.c
- dgcntl.c
- dgdump.c
- dginit.c
- dgmcntl.c
- dgmopen.c
- dgparse.c
- dgread.c
- dgwrite.c
- dot2ip.c
- dscntl.c
- dsinit.c
- dsinter.c
- dskbcpy.c
- dskenq.c
- dskqopt.c
- dskstrt.c
- dsksync.c
- dsopen.c
- dsread.c
- dsseek.c
- dswrite.c
- dudir.c
- dumkfs.c
- ethinit.c
- ethinter.c
- ethread.c

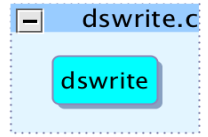
Graph 20 | Graph 21 | Graph 22 | Graph 23 | Graph 25 | »47



show(dswritedfg)

Atlas Connection View | Atlas Smart View

Dataflow Graph (DFG) of the pointer returned by getbuf()



Script: Reverse Call

Problems | Tasks | Console | Properties | Atlas Shell (AtlasToolbox

Atlas Shell (Project: AtlasToolbox)

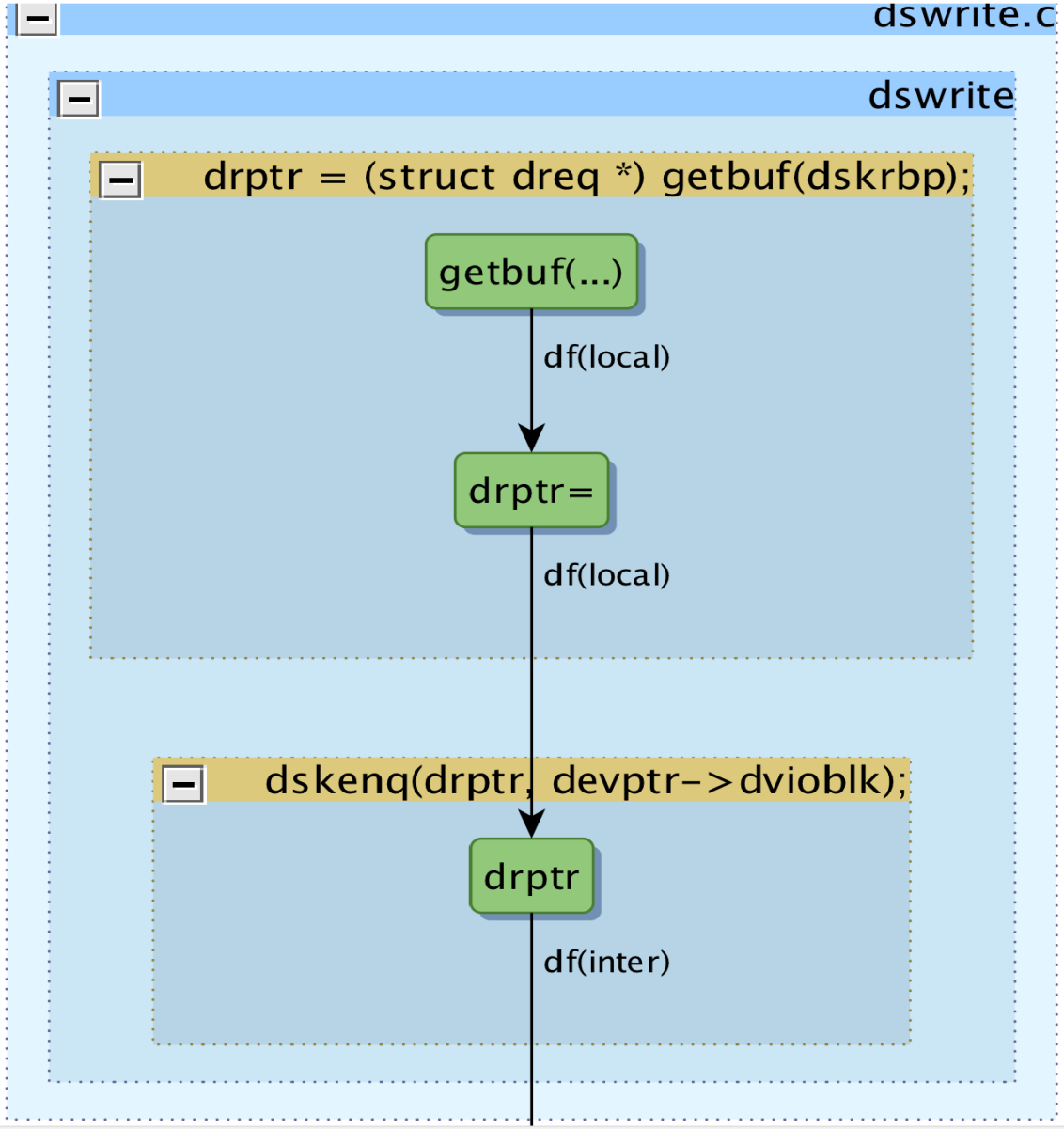
```
var dswritedfg = dfg("dswrite", "getbuf")

dswritedfg: com.ensoftcorp.atlas.core.query.Q = <Atlas query expression>

show(dswritedfg)
```

Evaluate: <type an expression or enter ":help" for more information>

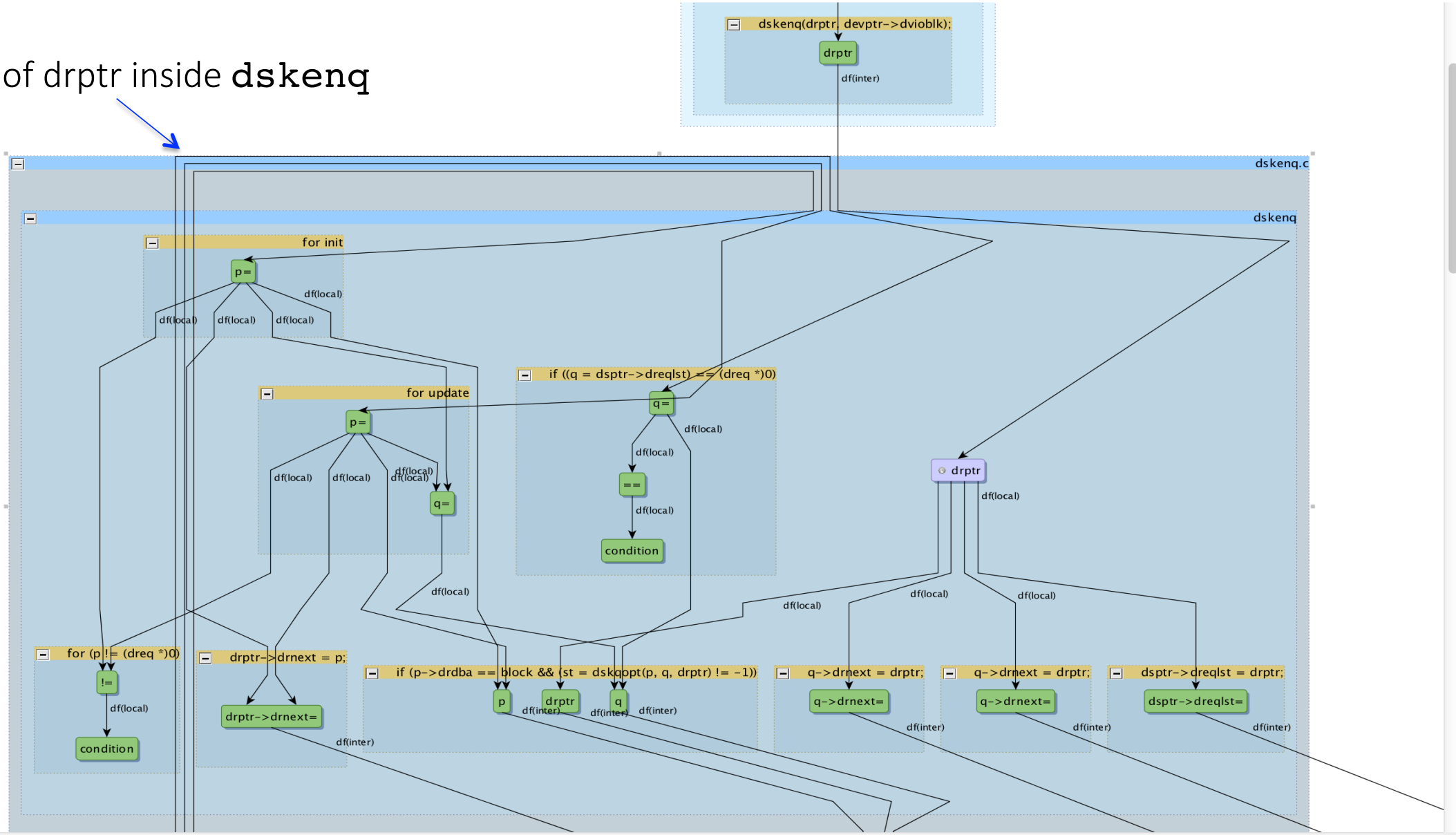
Element Detail View

- dswrite
  - XCSG.ModelElement.id = [Xinu/Xinu/sys/dswrite.c#dswrite(struct devsw*, char*, DBADDR)]
  - XCSG.ModelElement.name = dswrite
  - XCSG.ModelElement.sourceCorrespondence = [322+7+/Xinu/sys/dswrite.c]
  - ##index
  - external
  - isDef
  - XCSG.Function
  - XCSG.Language.C

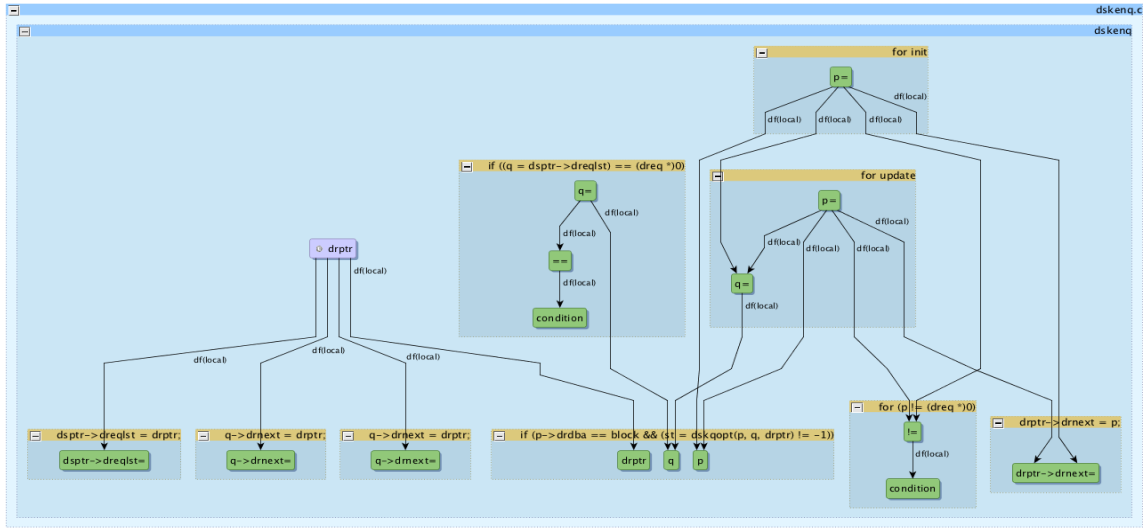
drptr returned by getbuf() passed as parameter to dskenq()

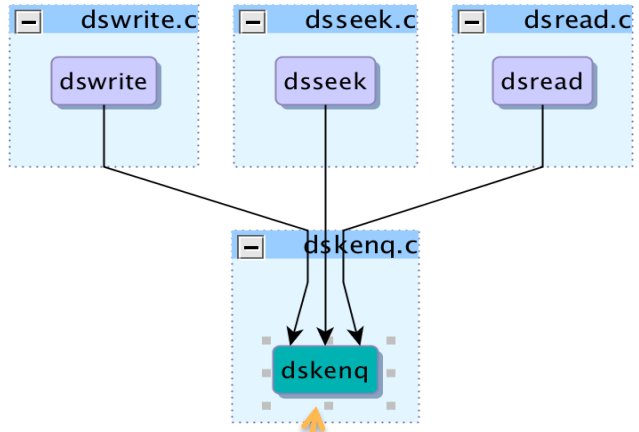


Dataflow of drptr inside **dskenq**



Dataflow within `dskenq` – obtained by projecting the DFG





var projection = projectdfg(dswritedfg, selected)
show(projection)

Script: Reverse Call

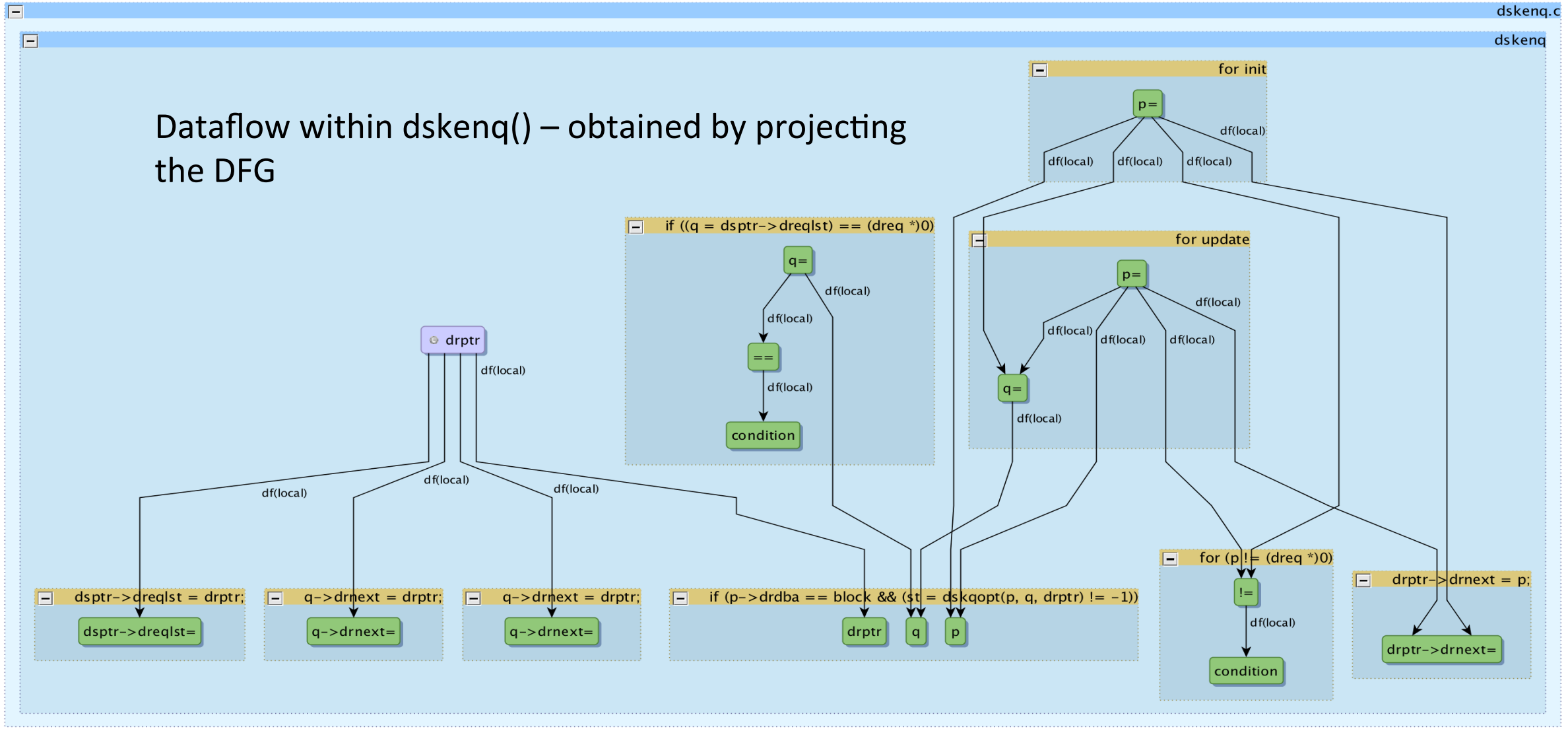
Atlas Shell (Project: AtlasToolbox)

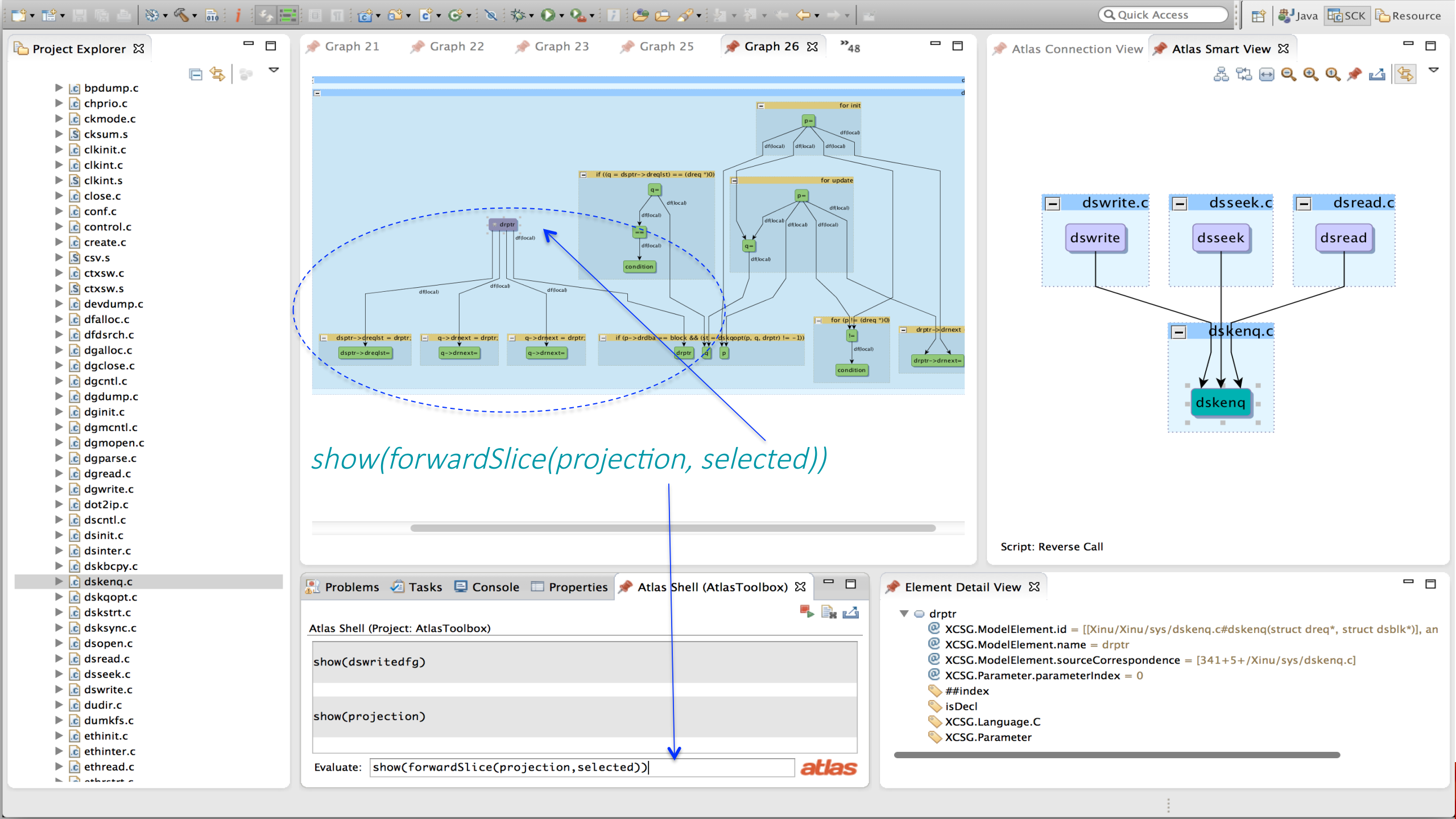
```
show(dswritedfg)

show(projection)
```

Evaluate: <type an expression or enter ":help" for more information>

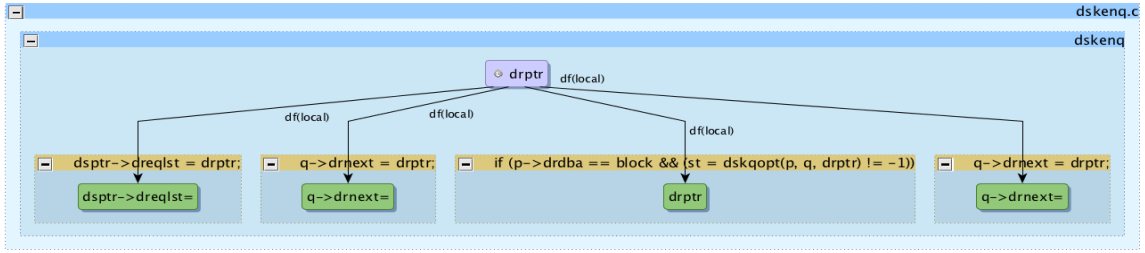
Element Detail View

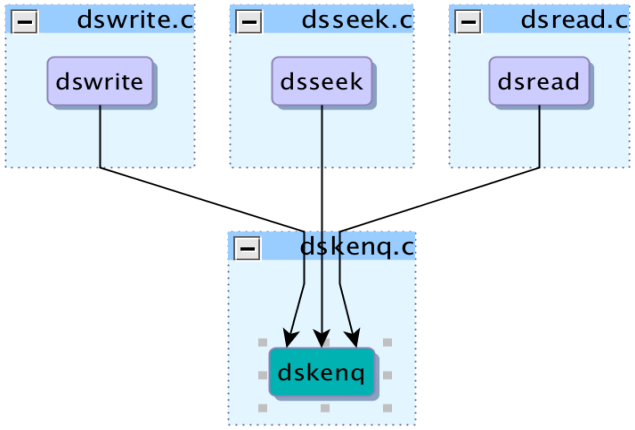
- dskenq
  - XCSG.ModelElement.id = [Xinu/Xinu/sys/dskenq.c#dskenq(struct dreq*, struct dsblk*)]
  - XCSG.ModelElement.name = dskenq
  - XCSG.ModelElement.sourceCorrespondence = [306+6+/Xinu/sys/dskenq.c]
  - ##index
  - external
  - isDef
  - XCSG.Function
  - XCSG.Language.C


Graph 14
Graph 15
Graph 16
Graph 17
Graph 18
Graph 19
Graph 20
Graph 21
Graph 22
Graph 23
Graph 25
Graph 26
dskenq.c
dskenq
Dataflow within dskenq() – obtained by projecting the DFG
for init
p=
df(local)
df(local)
df(local)
df(local)
if ((q = dsptr->dreqlst) == (dreq *)0)
q=
df(local)
df(local)
==
df(local)
condition
for update
p=
df(local)
df(local)
df(local)
df(local)
q=
df(local)
drptr
df(local)
df(local)
df(local)
df(local)
dsptr->dreqlst = drptr;
dsptr->dreqlst=
q->drnext = drptr;
q->drnext=
q->drnext = drptr;
q->drnext=
if (p->drdba == block && (st = dskqopt(p, q, drptr) != -1))
drptr
q
p
for (p != (dreq *)0)
!=
df(local)
condition
drptr->drnext = p;
drptr->drnext=


Project Explorer
bpdump.c
chprio.c
ckmode.c
cksum.s
clkinit.c
clkint.c
clkint.s
close.c
conf.c
control.c
create.c
csv.s
ctxsw.c
ctxsw.s
devdump.c
dfalloc.c
dfdsrch.c
dgalloc.c
dgclose.c
dgcntl.c
dgdump.c
dginit.c
dgmcntl.c
dgmopen.c
dgparse.c
dgread.c
dgwrite.c
dot2ip.c
dscntl.c
dsinit.c
dsinter.c
dskbcpy.c
dskenq.c
dskqopt.c
dskstrt.c
dsksync.c
dsopen.c
dsread.c
dsseek.c
dswrite.c
dudir.c
dumkfs.c
ethinit.c
ethinter.c
ethread.c
Graph 21
Graph 22
Graph 23
Graph 25
Graph 26
48
for init
for update
condition
drptr
dsptr->dreqlst = drptr;
dsptr->dreqlst=
q->drnext = drptr;
q->drnext=
drptr->drnext=
show(forwardSlice(projection, selected))
Atlas Connection View
Atlas Smart View
dswrite.c
dsseek.c
dsread.c
dswrite
dsseek
dsread
dskenq.c
dskenq
Script: Reverse Call
Problems
Tasks
Console
Properties
Atlas Shell (AtlasToolbox)
Atlas Shell (Project: AtlasToolbox)
show(dswritedfg)
show(projection)
Evaluate:
show(forwardSlice(projection,selected))
atlas
Element Detail View
drptr
XCSG.ModelElement.id = [[Xinu/Xinu/sys/dskenq.c#dskenq(struct dreq*, struct dsblk*)], an
XCSG.ModelElement.name = drptr
XCSG.ModelElement.sourceCorrespondence = [341+5+/Xinu/sys/dskenq.c]
XCSG.Parameter.parameterIndex = 0
##index
isDecl
XCSG.Language.C
XCSG.Parameter

Forward Slice of a node from dataflow graph





Script: Reverse Call

Atlas Shell (Project: AtlasToolbox)

```
show(projection)

show(forwardSlice(projection,selected))
```

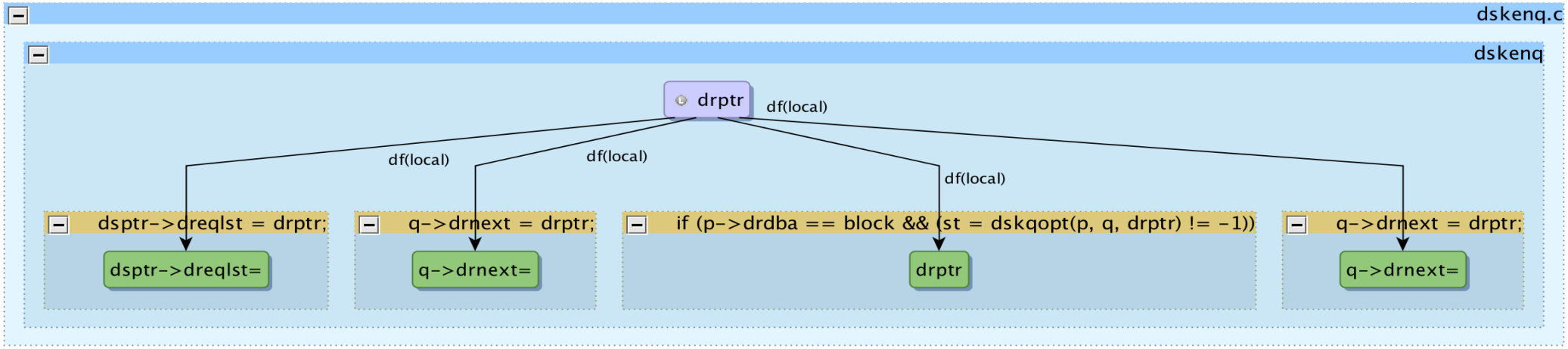
Evaluate: <type an expression or enter ":help" for more information>

Element Detail View

- drptr
  - XCSG.ModelElement.id = [[Xinu/Xinu/sys/dskenq.c#dskenq(struct dreq*, struct dsblk*)], an
  - XCSG.ModelElement.name = drptr
  - XCSG.ModelElement.sourceCorrespondence = [341+5+/Xinu/sys/dskenq.c]
  - XCSG.Parameter.parameterIndex = 0
  - ##index
  - isDecl
  - XCSG.Language.C
  - XCSG.Parameter

Graph 15 | Graph 16 | Graph 17 | Graph 18 | Graph 19 | Graph 20 | Graph 21 | Graph 22 | Graph 23 | Graph 25 | Graph 26 | Graph 27

Expanded the widow



Two Aspects of Hardness

- Interprocedural Interactions get complicated because of the varied ways the functions communicate with each other.
- Within a function, it gets complicated because of many paths, and with different complications within paths.

Quest for Relevant Functions

- Relevant: functions necessary and sufficient to verify a problem instance (e.g. each memory allocation).
- Butterfly Effect: A small change at one point causes unforeseen effects at far away points.
- How do we find relevant functions?
  - Identify all the communication mechanisms (e.g. pointer to allocated memory gets passed as a parameter).
  - Formulate the constraints for identifying relevant functions (e.g. functions called by getbuf and freebuf should not be in the set of relevant functions).

Fundamentals of data and control flow

- Relevant data: Data (D) relevant to a problem instance (e.g. pointers to an allocated memory).
- Fundamental mechanisms of data flow:
  - f passes D as a parameter to a callee function g.
  - f passes D as a parameter or return to a caller function g.
  - f and g share D through a global variable.

harder

- Fundamental mechanisms of control flow:
  - f calls g directly.
  - f calls g indirectly (e.g. using a function pointer).
  - f and g operate asynchronously (control transfer happens

harder

Formulation of constraints

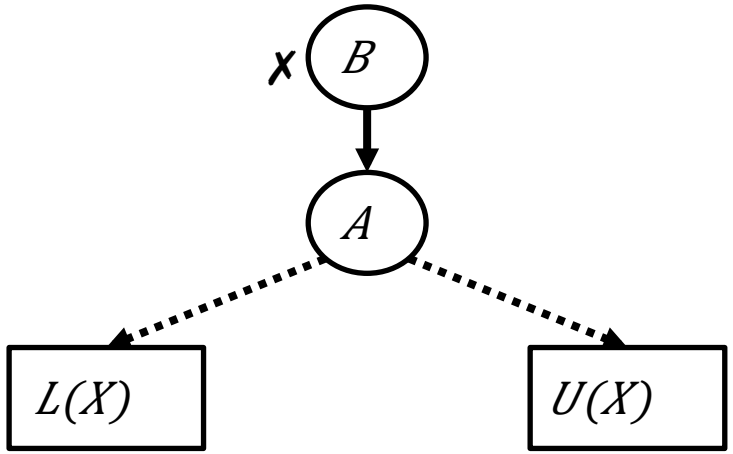
- Data-Centric Constraints: Exclude functions that do not access the data (D) relevant to the problem instance (e.g. exclude functions that do not access pointers to the allocated memory).
- Control-Centric Constraints: Exclude functions that cannot reach to functions relevant to the problem instance (e.g. exclude functions that do not have a call chain reaching to freebuf).
- Design-Centric Constraints: Exclude functions that imply an unnecessarily complicated design.

An example of design-centric constraint

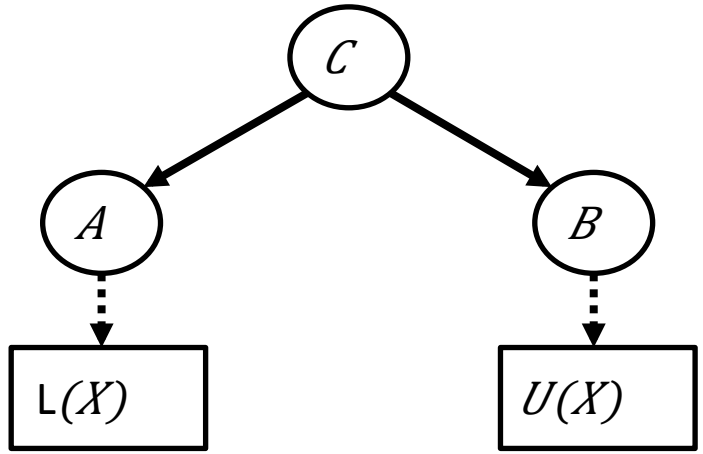
A constraint for the MP problems – If a function `f` reaches getbuf and freebuf, the developer can implement the matching logic at `f` itself. The design is unnecessarily complicated if that logic is moved up to a function `g` that calls `f`.

calls contains

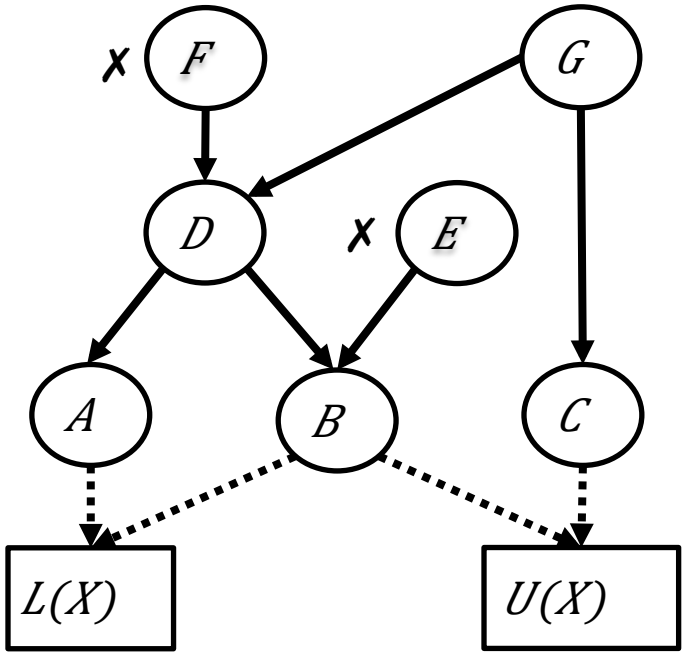
What functions should not be there (✗)?



(a)



(b)



(c)

Matching Pair Graph

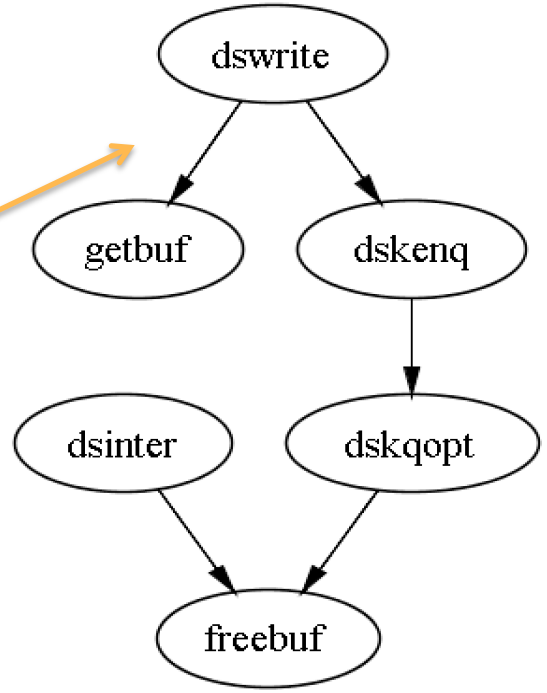
- A visual graph model to discover functions relevant to each MP instance.
- Founded on the knowledge of the fundamental *propagation mechanisms* and the *propagation boundaries.*
- *Directed graph* with functions as the nodes and calls as the edges.
- *Roots* are asynchronous functions.
- *Leaves* are MP functions (e.g. **`getbuf`** and **`freebuf`**).

MPG can serve as verification-critical evidence

We illustrate how MPG provides important clues to verify an MP instance.

```
dswrite(devptr, buff, block)
     struct   devsw    *devptr;
     char *buff;
     DBADDR   block;
{
     struct   dreq *drptr;
     char ps;

     disable(ps);
     drptr = (struct dreq *) getbuf(dskrbp);
     drptr->drbuff = buff;
     drptr->drdba = block;
     drptr->drpid = currpid;
     drptr->drop = DWRITE;
     dskenq(drptr, devptr->dvioblk);
     restore(ps);
     return(OK);
}
```

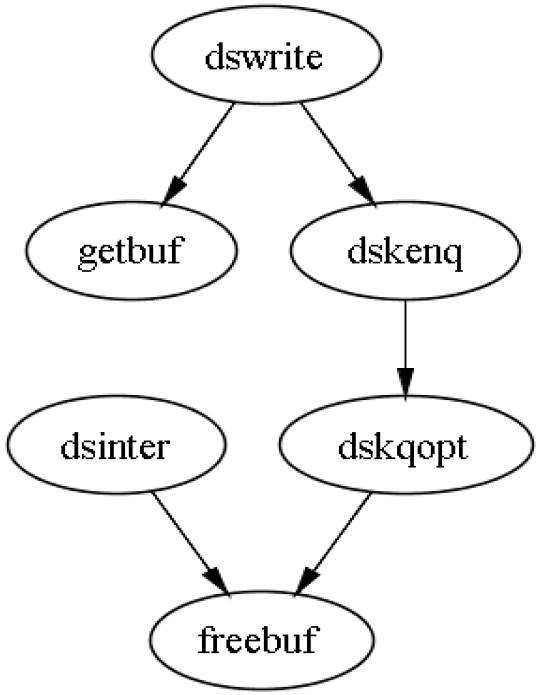
The call chain from **`dswrite`** to **`dskqopt`** indicates that the pointer to the allocated memory (**`drptr`**) gets passed to **`dskqopt`** which releases the memory.



Interesting Question: What does the presence of **`dsinter`** indicate?

MPG as Verification - Critical Evidence

Question: What does the presence of **`dsinter`** indicate?



Note that:

1. The function **`dsinter`** deallocates memory.
2. **`dsinter`** does not allocate memory.
3. **`dswrite`** and **`dsinter`** operate asynchronously.

⇓ *hypothesis*

On some paths the *pointer to the allocated memory* (**`drptr`**) is shared with **`dsinter`** and it deallocates the memory asynchronously.

Enables correct reasoning for a case involving *asynchronous deallocation* by an interrupt routine - a hard case for automation.

A closer look dsinter – the interrupt-driven routine

```
INTPROC     dsinter(dsptr)
        struct      dsblk *dsptr;
{
        struct      dtc   *dtptr;
        struct      dreq  *drptr;

        dtptr = dsptr->dcsr;
        drptr = dsptr->dreqlst;
        if (drptr == DRNULL) {
X           panic("Disk interrupt when disk not busy");
            return;
        }
X       if (dtptr->dt_csr & DTERROR)
            drptr->drstat = SYSERR;
        else
            drptr->drstat = OK;
X       if ( (dsptr->dreqlst=drptr->drnext) != DRNULL)
            dskstrt(dsptr);
        switch (drptr->drop) {

            case DREAD:
X           case DSYNC:
                ready(drptr->drpid, RESCHYES);
                return;

            case DWRITE:
                freebuf(drptr->drbuff);
                /* fall through */
            case DSEEK:
                freebuf(drptr);
        }
}
```

How many *relevant* execution paths ?

Principle: A branch *relevant* only if it has interesting events

Three interesting events marked

Several branches eliminated from consideration, because no interesting events on those branches

Left with only 2 paths out of 17 potential paths – less work !

```
INTPROC     dsinter(dsptr)
      struct      dsblk *dsptr;
{
      struct      dtc   *dtptr;
      struct      dreq  *drptr;

      dtptr = dsptr->dcsr;
      drptr = dsptr->dreqlst;
      if (drptr == DRNULL) {
            panic("Disk interrupt when disk not busy");
            return;
      }
      if (dtptr->dt_csr & DTERROR)
            drptr->drstat = SYSERR;
      else
            drptr->drstat = OK;
      if ( (dsptr->dreqlst=drptr->drnext) != DRNULL)
            dskstrt(dsptr);
      switch (drptr->drop) {

            case DREAD:
            case DSYNC:
                  ready(drptr->drpid, RESCHYES);
                  return;

            case DWRITE:
                  freebuf(drptr->drbuff);
                  /* fall through */
            case DSEEK:
                  freebuf(drptr);
      }
}
```

An escape happened but through parameter to dskstrt

Principle: Function calls not interesting if the function is not in MPG(X)

dskopt Seven execution paths shown as rows

Input	Conditions: dreq.drop ==						Event Description
Dreq devsw.dvioblk.dreqlst	DYNCH	DSEEK	C1	DWRITE & C2	DREAD & C3	DWRITE & C3	
	T	-	-	-	-	-	Error
	F	T	-	-	-	-	freebuf, sig = dreq
	F	F	T	-	-	-	*devsw.dvioblk.dreqlst ← dreq* freebuf, sig = dreq
	F	F	F	T	-	-	*devsw.dvioblk.dreqlst ← dreq* freebuf, sig = char freebuf, sig = dreq
	F	F	F	F	T	-	
	F	F	F	F	F	T	*devsw.dvioblk.dreqlst ← dreq*
	F	F	F	F	F	F	*devsw.dvioblk.dreqlst ← dreq*

dsinter

Input	Conditions							Event Description
dsblk	`drptr == DRNULL`	D1	D2	`drptr->drop ==`				
				DREAD	DSYNC	DWRITE	DSEEK	

	F	-	-	F	F	T		*dreq ← dsblk.dreqlst*
								`freebuf, sig = char*`
								`freebuf, sig = dreq`

	F	-	-	F	F	F	T	*dreq ← dsblk.dreqlst* `freebuf, sig = dreq`

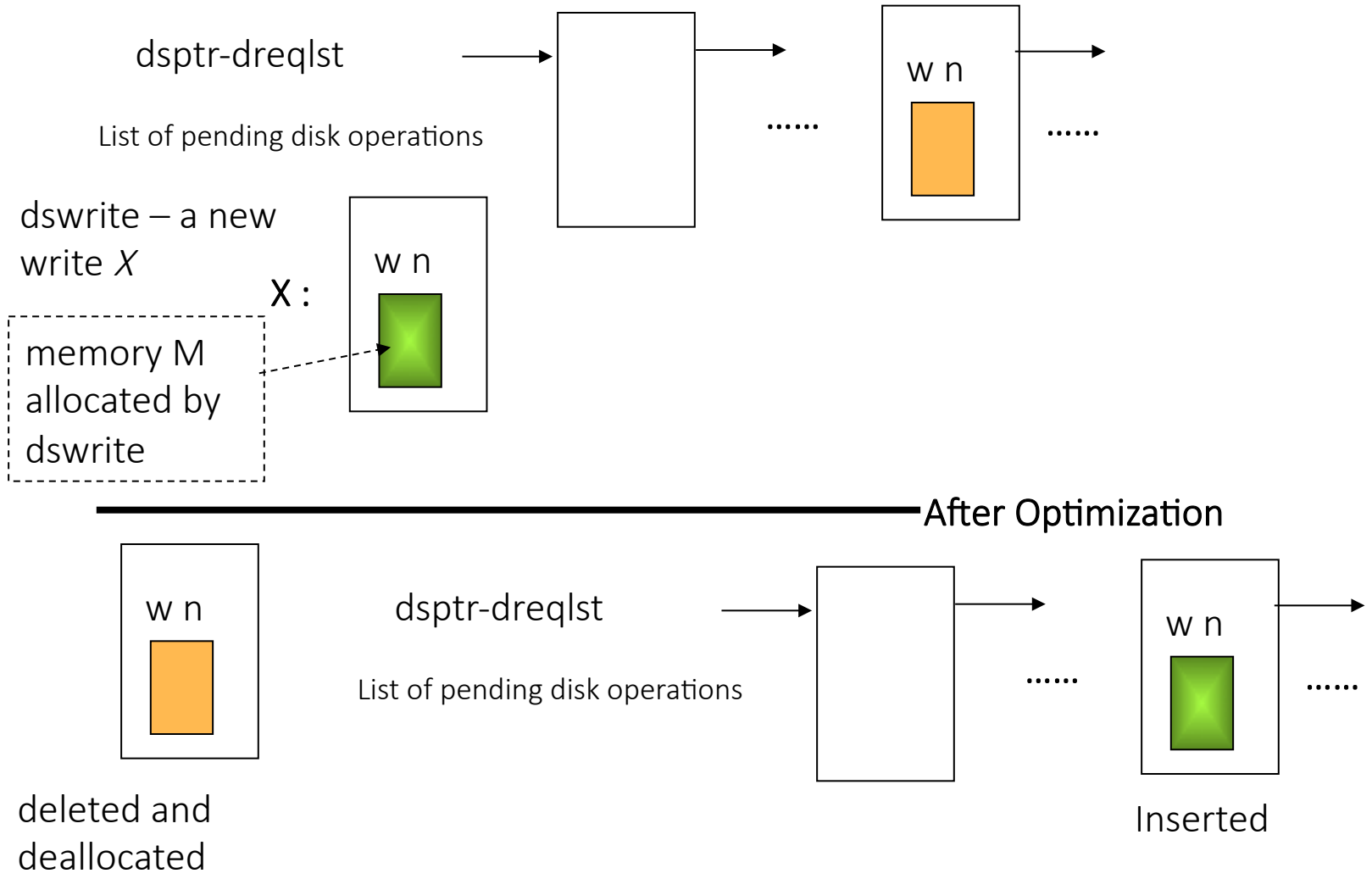
A logical explanation of the disk driver

- This part of the XINU code is very well written – it is complicated code because the underlying logic requires it to be.
- We will go through a relevant part of the underlying logic to appreciate the code.
- Note that the code had to be written very carefully – several lines of code are critical, miss one of them and it will create a memory leak.
- For example, miss the line `q->drnext = drptr;` in path 2-F and it will be a serious memory leak that will crash the system very quickly.

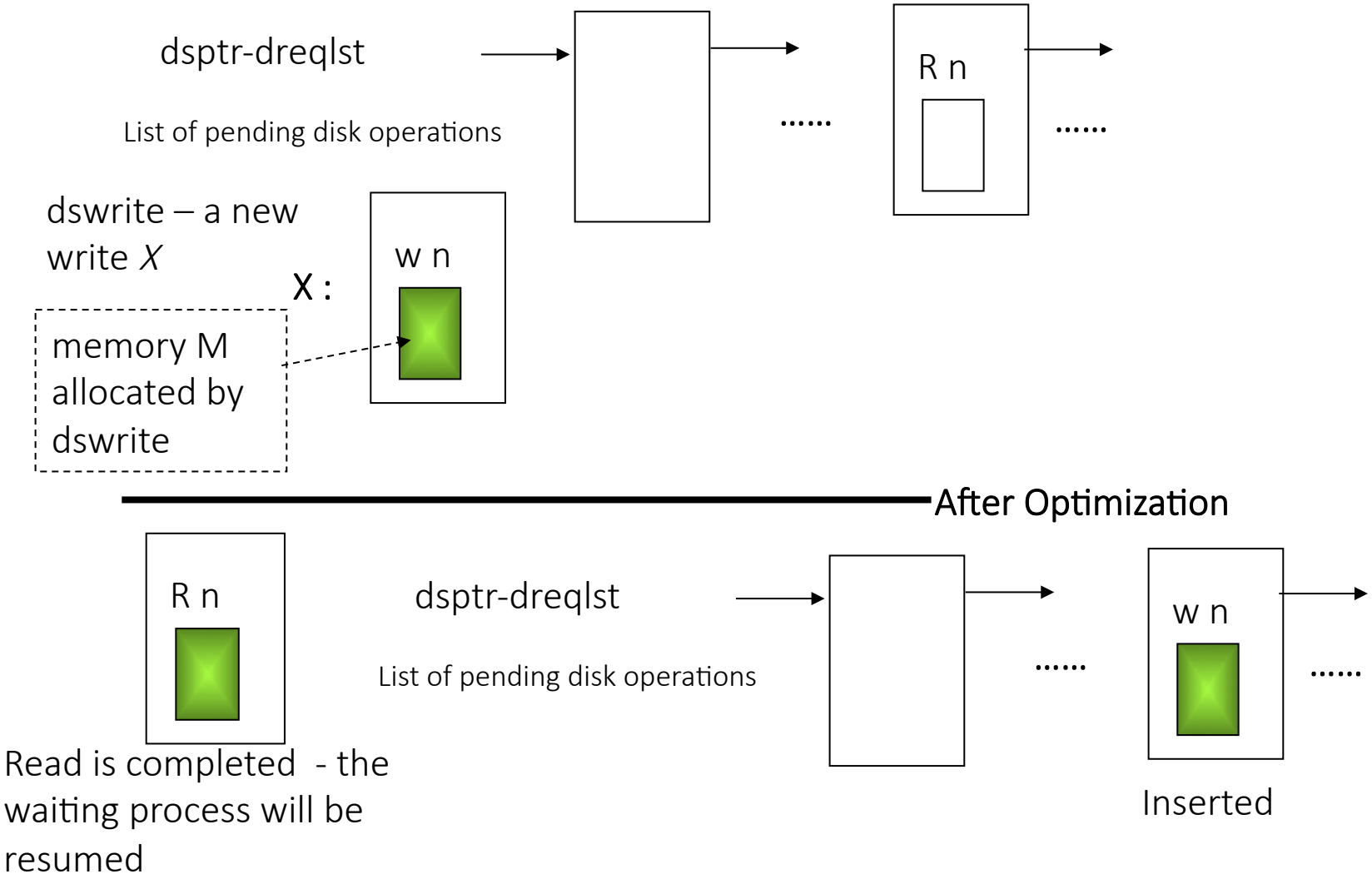
The underlying optimization

- A *write X* (or another disk operation) is issued for disk block *n*
- Disk operations, since they are slow, get queued in a linked list *L* of pending operations to be performed.
- Optimize if there is already a pending operation for the same block:
  - Path D (/* *dup write* */): A previous pending write to block n is deleted from L and the X is inserted in L.
  - Path F (/* sat. old read */): A previous pending *read* to block *n* is completed (and deleted from *L*) in memory using *X* the *X* is inserted in *L*.

Duplicate *write* optimization



Duplicate *write*



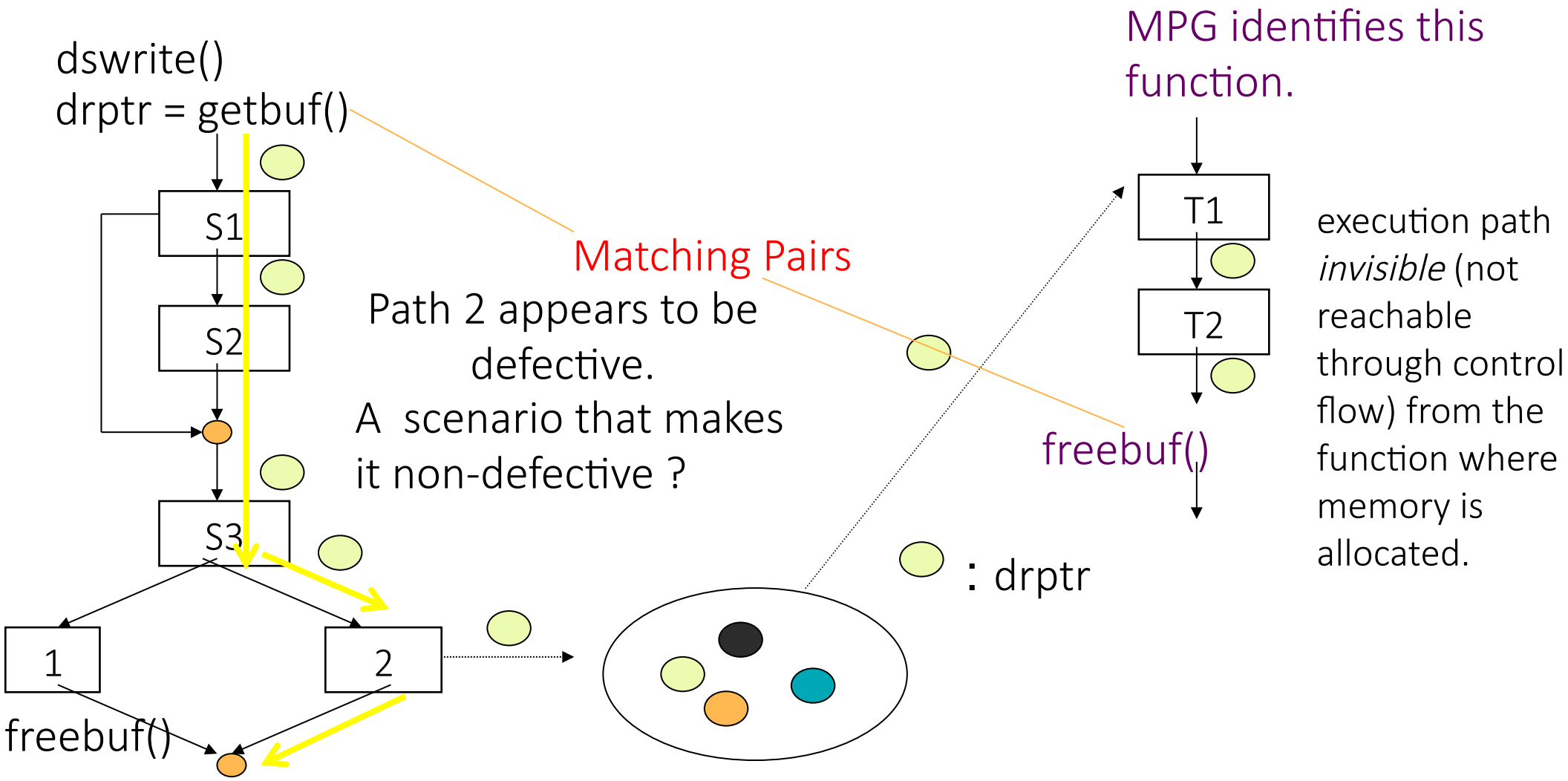
About the paths in dskenq

- Path 1: List *L* of pending disk operations is empty
- Path 2: *L* already has a pending operation for the same block
- Path 3: Insertion in the *middle* of the list – thus *L* is kept sorted to minimize time spent on moving the disk head
- Path 4: Insertion at the *end* of the list

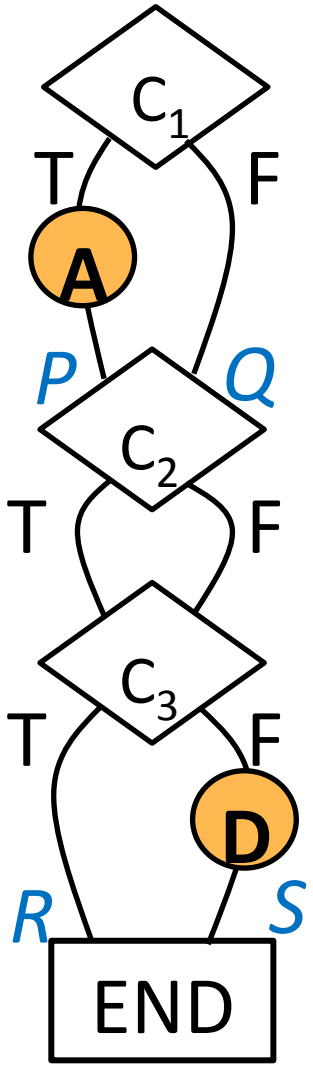
The producer-consumer pattern

- The device driver - follows a widely used coding pattern, called the *producer-consumer problem*.
  - Producer *P* – a thread that generates requests, *inserts* them in a list *L*. *P allocates* memory for the request.
  - Consumer C – another thread that consumes requests, deletes them from the list *L*. P *deallocates* memory for the request.
- A classic example of the *level-4 MP* problem
- Also, a *synchronization problem* because of the need for mutually exclusive access to the shared list *L*.
  - trivially solved in XINU by disabling interrupts

Hardness due to asynchronous paths



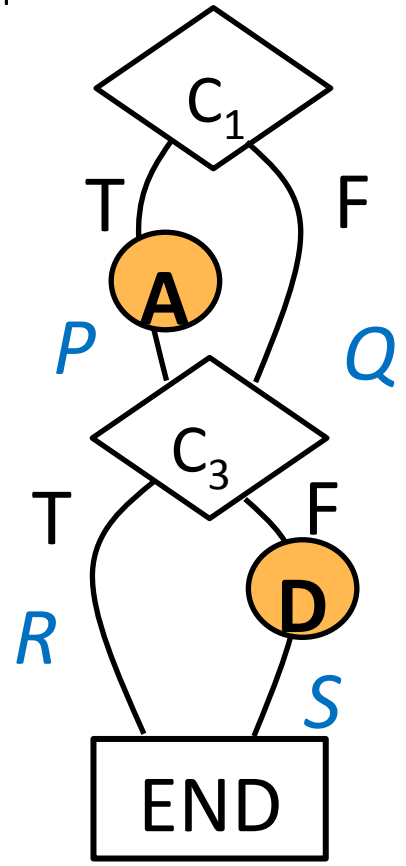
Irrelevant branch nodes



C_2 Irrelevant to path-sensitive analysis

Remove the Irrelevant branch nodes to avoid unnecessary path explosion & also simplify the path feasibility check.

paths reduced from 8 to 4

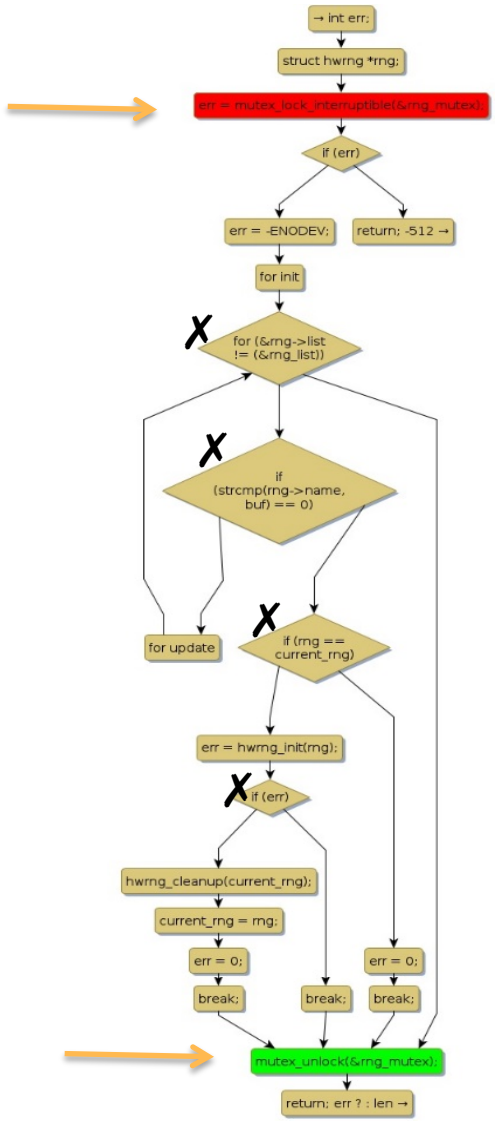


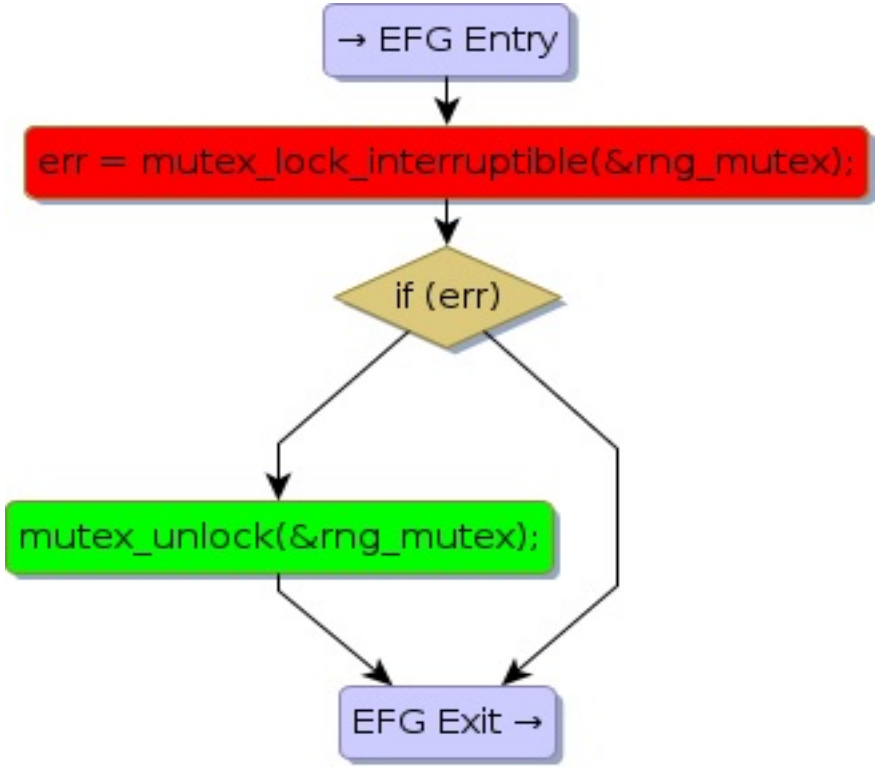
conditions for feasibility check reduced from 3 to 2

Efficient path-sensitive analysis

- A large number of paths could be partitioned into a small number of groups.
- All Paths in a group are equivalent – have the same execution behavior w.r.t. the property to be verified.
- Efficient computation by examining only one path from each group.
- Challenge: How can the groups be formed without examining each path at least once?

A Linux example to illustrate efficient path-sensitive analysis





PCG for efficient path-sensitive analysis

- Projected Control Graph (PCGG) is derived from the Control Flow Graph (CFG).
- Relevant events: CFG nodes relevant to the property (e.g. **`getbuf`**, **`freebuf`**, nodes of type **`p`** = **`drptr`** (pointer to the allocated memory)).
- Behavior: An event trace along a CFG path
- PCG has one path per group of CFG paths with identical behaviors.
- PCG construction: o(K) where K = N + E, N: # nodes in CFG, E: # edges in CFG.

Hardness captured by MPG and PCG

- MPG captures interprocedural complexity:
  - the functions to be analyzed.
  - the data flow between the relevant functions.
  - the control flow between the relevant functions.
- EFG captures intra-procedural Complexity:
  - the paths with distinct behaviors.
  - the relevant conditions for performing path feasibility check.
- MPG & EFG together serve as:
  - The evidence to verify each MP instance.
  - They reveal the hardness challenges of each MP instance and thus can be the basis for defining hardness metrics.

Using Atlas to create matching pair graph

- MPG:
  - Functions and their call relationships *relevant* for verifying an MP instance.
  - *Relevant*: functions that contain the matching events and also the functions propagating the pointers to objects for matching the events.
  - Based on fundamental principles of how functions communicate.
- The lab covers simple experiments to evolve ideas

Project Explorer

- csv.s
- ctxsw.c
- ctxsw.s
- devdump.c
- dfalloc.c
- dfdsrch.c
- dgalloc.c
- dgclose.c
- dgcntl.c
- dgdump.c
- dginit.c
- dgmcntl.c
- dgmopen.c
- dgparse.c
- dgread.c
- dgwrite.c
- dot2ip.c
- dscntl.c
- dsinit.c
- dsinter.c
- dskbcpy.c
- dskenq.c
- dskqopt.c
- dskstrt.c
- dsksync.c
- dsopen.c
- dsread.c
- dsseek.c
- dswrite.c
- dudir.c
- dumkfs.c
- ethinit.c
- ethinter.c
- ethread.c
- ethrstrt.c
- ethstrt.c
- ethwrite.c
- ethwstrt.c
- freebuf.c
- freemem.c
- getaddr.c
- getbuf.c
- getc.c
- getitem.c
- getmem.c

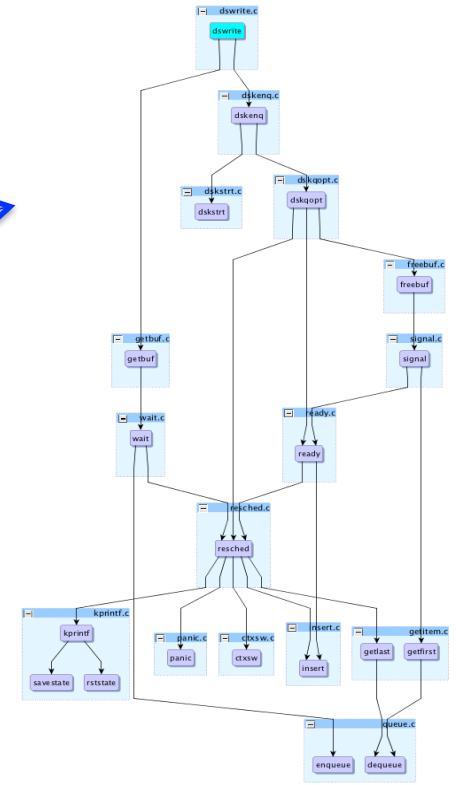
dswrite.c | Graph 2 | Graph 3 | Graph 4 | Graph 5 | Graph 6

```
/* dswrite.c - dswrite */

#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <disk.h>

 *  dswrite  --  write a block (system buffer) onto a disk device
dswrite(devptr, buff, block)
    struct  devsw   *devptr;
    char    *buff;
    DBADDR  block;
{
    struct  dreq    *drptr;
    char    ps;

    disable(ps);
    drptr = (struct dreq *) getbuf(dskrbp);
    drptr->drbuff = buff;
    drptr->drdba = block;
    drptr->drpid = currpid;
    drptr->drop = DWRITE;
    dskenq(drptr, devptr->dvioblk);
    restore(ps);
    return(OK);
}
```

Forward call graph is neither sufficient nor necessary.

Let us try some ideas for the MPG (the graph of relevant functions with call relationships for verifying the allocation instance in **dswrite**.

Atlas Connection View | Atlas Smart View



Script: Forward Call

Problems | Tasks | Console | Properties | Atlas Shell (AtlasToolbox

Atlas Shell (Project: AtlasToolbox)

```
var z = graph(rootx,y)

z: com.ensoftcorp.atlas.core.query.Q = <Atlas query expression>

show(z)
```

Evaluate: <type an expression or enter ":help" for more information>

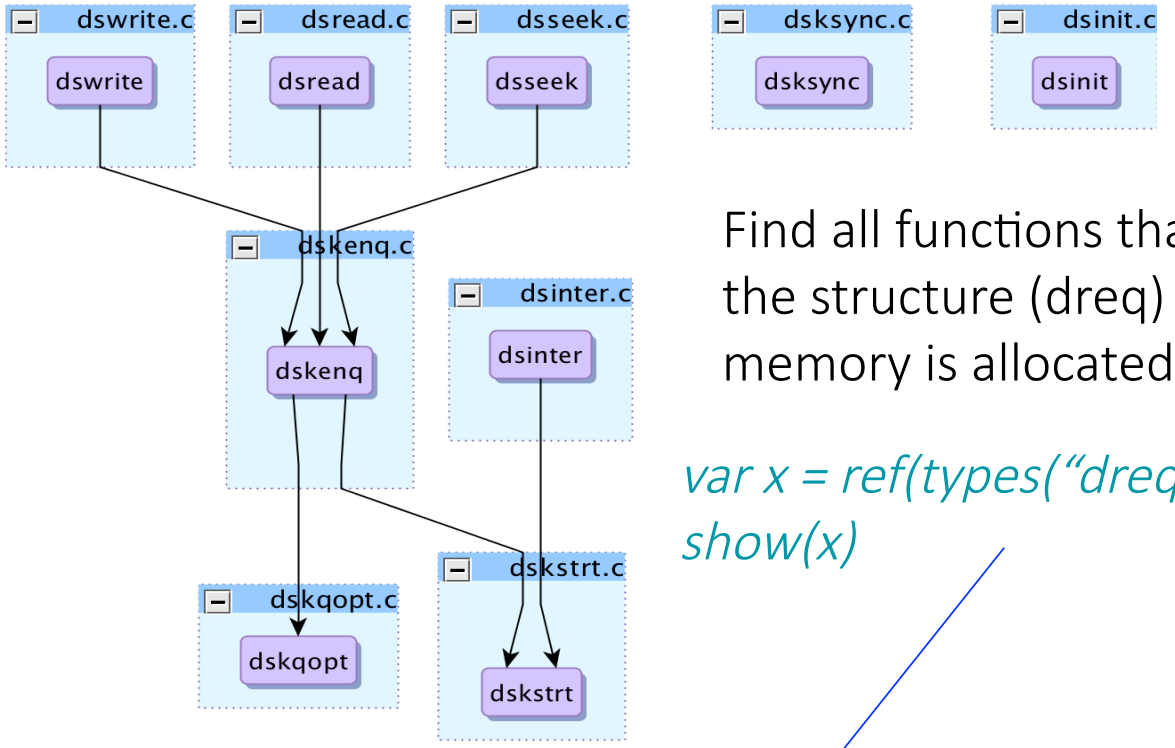
Element Detail View

- dswrite
  - XCSG.ModelElement.id = [Xinu/Xinu/sys/dswrite.c#dswrite(struct devsw*, char*, DBADDR)]
  - XCSG.ModelElement.name = dswrite
  - XCSG.ModelElement.sourceCorrespondence = [322+7+/Xinu/sys/dswrite.c]
  - ##index
  - external
  - isDef
  - XCSG.Function
  - XCSG.Language.C

Project Explorer

- csv.s
- ctxsw.c
- ctxsw.s
- devdump.c
- dfalloc.c
- dfdsrch.c
- dgalloc.c
- dgclose.c
- dgcntl.c
- dgdump.c
- dginit.c
- dgmcntl.c
- dgmopen.c
- dgparse.c
- dgread.c
- dgwrite.c
- dot2ip.c
- dscntl.c
- dsinit.c
- dsinter.c
- dskbcpy.c
- dskenq.c
- dskqopt.c
- dskstrt.c
- dsksync.c
- dsopen.c
- dsread.c
- dsseek.c
- dswrite.c
- dudir.c
- dumkfs.c
- ethinit.c
- ethinter.c
- ethread.c
- ethrstrt.c
- ethstrt.c
- ethwrite.c
- ethwstrt.c
- freebuf.c
- freemem.c
- getaddr.c
- getbuf.c
- getc.c
- getitem.c
- getmem.c

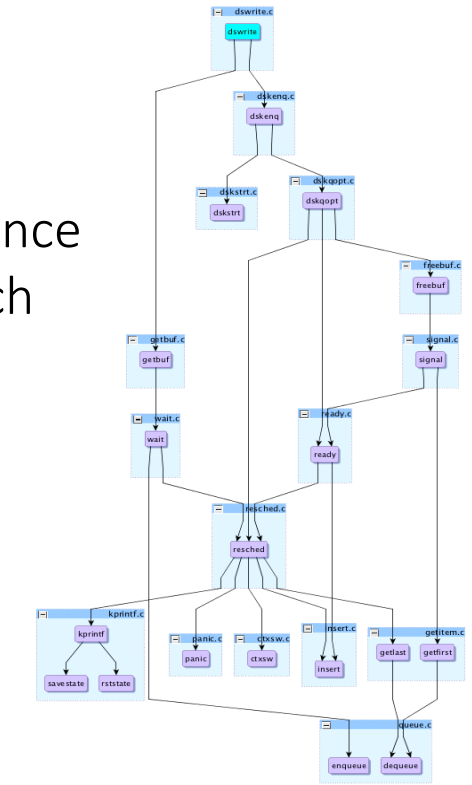
Graph 32 | Graph 33 | Graph 34 | Graph 35 | Graph 36 | 58



Find all functions that reference the structure (dreq) for which memory is allocated

var x = ref(types("dreq"))
show(x)

Atlas Connection View | Atlas Smart View



Script: Forward Call

Problems | Tasks | Console | Properties | Atlas Shell (AtlasToolbox)

Atlas Shell (Project: AtlasToolbox)

```
var x = ref(types("dreq"))

x: com.ensoftcorp.atlas.core.query.Q = <Atlas query expression>

show(x)
```

Evaluate: <type an expression or enter "help" for more information>

Element Detail View

- dswrite
  - XCSG.ModelElement.id = [Xinu/Xinu/sys/dswrite.c#dswrite(struct devsw*, char*, DBADDR)]
  - XCSG.ModelElement.name = dswrite
  - XCSG.ModelElement.sourceCorrespondence = [322+7+/Xinu/sys/dswrite.c]
  - ##index
  - external
  - isDef
  - XCSG.Function
  - XCSG.Language.C

Quick Access | Java | SCK | Resource

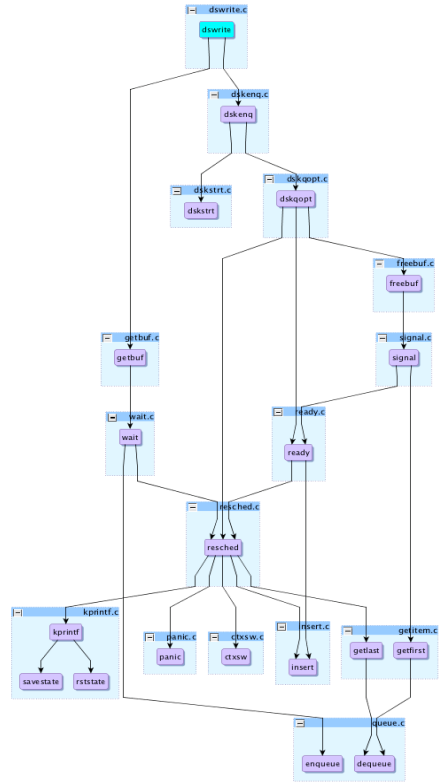
Project Explorer

- csv.s
- ctxsw.c
- ctxsw.s
- devdump.c
- dfalloc.c
- dfdsrch.c
- dgalloc.c
- dgclose.c
- dgcntl.c
- dgdump.c
- dginit.c
- dgmcntl.c
- dgmopen.c
- dgparse.c
- dgread.c
- dgwrite.c
- dot2ip.c
- dscntl.c
- dsinit.c
- dsinter.c
- dskbcpy.c
- dskenq.c
- dskqopt.c
- dskstrt.c
- dsksync.c
- dsopen.c
- dsread.c
- dsseek.c
- dswrite.c
- dudir.c
- dumkfs.c
- ethinit.c
- ethinter.c
- ethread.c
- ethrstrt.c
- ethstrt.c
- ethwrite.c
- ethwstrt.c
- freebuf.c
- freemem.c
- getaddr.c
- getbuf.c
- getc.c
- getitem.c
- getmem.c

Graph 33 | Graph 34 | Graph 35 | Graph 36 | Graph 37 | 59

var rootx = x.roots()
show(rootx)

Atlas Connection View | Atlas Smart View



Script: Forward Call

Problems | Tasks | Console | Properties | Atlas Shell (AtlasToolbox

Atlas Shell (Project: AtlasToolbox)

```
var rootx = x.roots()

rootx: com.ensoftcorp.atlas.core.query.Q = <Atlas query expression>

show(rootx)
```

Evaluate: <type an expression or enter ":help" for more information>

Element Detail View

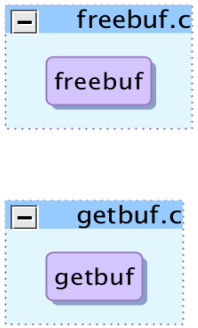
- dswrite
  - XCSG.ModelElement.id = [Xinu/Xinu/sys/dswrite.c#dswrite(struct devsw*, char*, DBADDR)]
  - XCSG.ModelElement.name = dswrite
  - XCSG.ModelElement.sourceCorrespondence = [322+7+/Xinu/sys/dswrite.c]
  - ##index
  - external
  - isDef
  - XCSG.Function
  - XCSG.Language.C

Quick Access | Java | SCK | Resource

Project Explorer

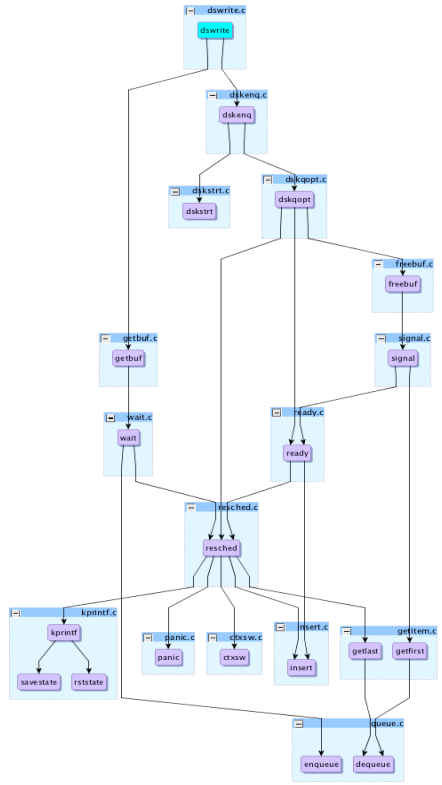
- csv.s
- ctxsw.c
- ctxsw.s
- devdump.c
- dfalloc.c
- dfdsrch.c
- dgalloc.c
- dgclose.c
- dgcntl.c
- dgdump.c
- dginit.c
- dgmcntl.c
- dgmopen.c
- dgparse.c
- dgread.c
- dgwrite.c
- dot2ip.c
- dscntl.c
- dsinit.c
- dsinter.c
- dskbcpy.c
- dskenq.c
- dskqopt.c
- dskstrt.c
- dsksync.c
- dsopen.c
- dsread.c
- dsseek.c
- dswrite.c
- dudir.c
- dumkfs.c
- ethinit.c
- ethinter.c
- ethread.c
- ethrstrt.c
- ethstrt.c
- ethwrite.c
- ethwstrt.c
- freebuf.c
- freemem.c
- getaddr.c
- getbuf.c
- getc.c
- getitem.c
- getmem.c

Graph 34 | Graph 35 | Graph 36 | Graph 37 | Graph 38 | 60



var leaves = functions("getbuf","freebuf")
show(leaves)

Atlas Connection View | Atlas Smart View



Script: Forward Call

Problems | Tasks | Console | Properties | Atlas Shell (AtlasToolbox

Atlas Shell (Project: AtlasToolbox)

```
var leaves = functions("getbuf","freebuf")

leaves: com.ensoftcorp.atlas.core.query.Q = <Atlas query expression>

show(leaves)
```

Evaluate: <type an expression or enter ":help" for more information>

Element Detail View

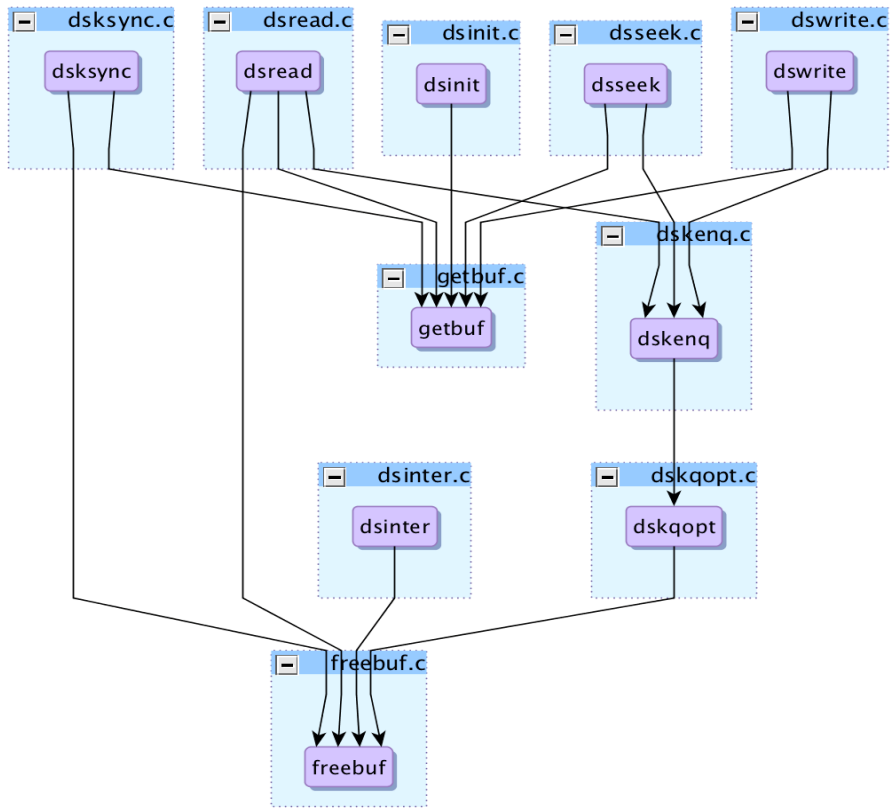
- dswrite
  - XCSG.ModelElement.id = [Xinu/Xinu/sys/dswrite.c#dswrite(struct devsw*, char*, DBADDR)]
  - XCSG.ModelElement.name = dswrite
  - XCSG.ModelElement.sourceCorrespondence = [322+7+/Xinu/sys/dswrite.c]
  - ##index
  - external
  - isDef
  - XCSG.Function
  - XCSG.Language.C

Quick Access | Java | SCK | Resource

Project Explorer

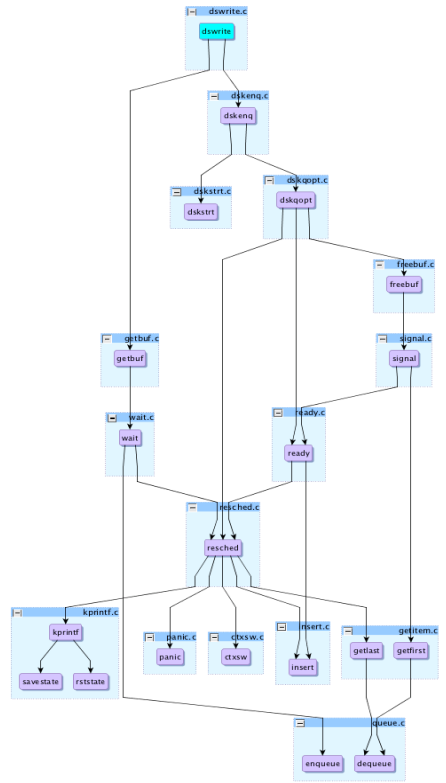
- csv.s
- ctxsw.c
- ctxsw.s
- devdump.c
- dfalloc.c
- dfdsrch.c
- dgalloc.c
- dgclose.c
- dgcntl.c
- dgdump.c
- dginit.c
- dgmcntl.c
- dgmopen.c
- dgparse.c
- dgread.c
- dgwrite.c
- dot2ip.c
- dscntl.c
- dsinit.c
- dsinter.c
- dskbcpy.c
- dskenq.c
- dskqopt.c
- dskstrt.c
- dsksync.c
- dsopen.c
- dsread.c
- dsseek.c
- dswrite.c
- dudir.c
- dumkfs.c
- ethinit.c
- ethinter.c
- ethread.c
- ethrstrt.c
- ethstrt.c
- ethwrite.c
- ethwstrt.c
- freebuf.c
- freemem.c
- getaddr.c
- getbuf.c
- getc.c
- getitem.c
- getmem.c

Graph 35 | Graph 36 | Graph 37 | Graph 38 | Graph 39 | 61



show(graph(rootx,leaves))

Atlas Connection View | Atlas Smart View



Script: Forward Call

Problems | Tasks | Console | Properties | Atlas Shell (AtlasToolbox

Atlas Shell (Project: AtlasToolbox)

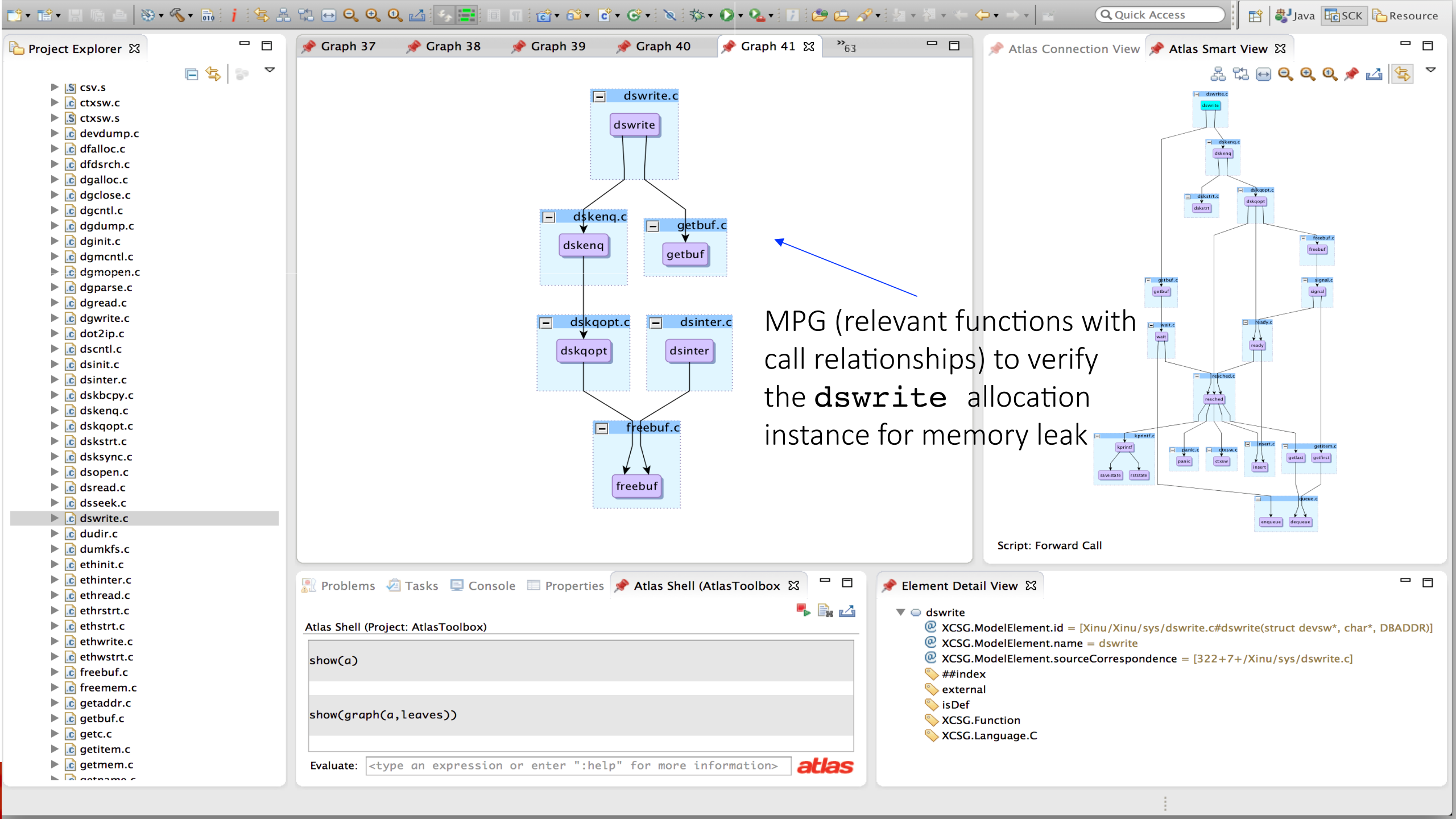
```
show(leaves)

show(graph(rootx,leaves))
```

Evaluate: <type an expression or enter ":help" for more information>

atlas

Element Detail View

- dswrite
  - XCSG.ModelElement.id = [Xinu/Xinu/sys/dswrite.c#dswrite(struct devsw*, char*, DBADDR)]
  - XCSG.ModelElement.name = dswrite
  - XCSG.ModelElement.sourceCorrespondence = [322+7+/Xinu/sys/dswrite.c]
  - ##index
  - external
  - isDef
  - XCSG.Function
  - XCSG.Language.C


Project Explorer
csv.s
ctxsw.c
ctxsw.s
devdump.c
dfalloc.c
dfdsrch.c
dgalloc.c
dgclose.c
dgcntl.c
dgdump.c
dginit.c
dgmcntl.c
dgmopen.c
dgparse.c
dgread.c
dgwrite.c
dot2ip.c
dscntl.c
dsinit.c
dsinter.c
dskbcpy.c
dskenq.c
dskqopt.c
dskstrt.c
dsksync.c
dsopen.c
dsread.c
dsseek.c
dswrite.c
dudir.c
dumkfs.c
ethinit.c
ethinter.c
ethread.c
ethrstrt.c
ethstrt.c
ethwrite.c
ethwstrt.c
freebuf.c
freemem.c
getaddr.c
getbuf.c
getc.c
getitem.c
getmem.c
Graph 37
Graph 38
Graph 39
Graph 40
Graph 41
dswrite.c
dswrite
dskenq.c
dskenq
getbuf.c
getbuf
dskqopt.c
dskqopt
dsinter.c
dsinter
freebuf.c
freebuf
MPG (relevant functions with call relationships) to verify the dswrite allocation instance for memory leak
Atlas Connection View
Atlas Smart View
Script: Forward Call
Problems
Tasks
Console
Properties
Atlas Shell (AtlasToolbox
Atlas Shell (Project: AtlasToolbox)
show(a)
show(graph(a,leaves))
Evaluate:
<type an expression or enter ":help" for more information>
atlas
Element Detail View
dswrite
XCSG.ModelElement.id = [Xinu/Xinu/sys/dswrite.c#dswrite(struct devsw*, char*, DBADDR)]
XCSG.ModelElement.name = dswrite
XCSG.ModelElement.sourceCorrespondence = [322+7+/Xinu/sys/dswrite.c]
##index
external
isDef
XCSG.Function
XCSG.Language.C

Project: Verify the Linux kernel for a safety property

- Goal
  - Bring out the challenges of real-world software.
  - Demonstrate how graph models can address the challenges.
- Planned Activities
  - Perform experiments to understand the challenges of multiplicative growth of execution paths and path feasibility.
  - Identify the fundamentals necessary to address the challenges.
  - Apply the principles to design graph models for effective detection and verification.
  - Practice critical thinking through experiments to design the graph models.
  - Discuss research projects that participants can take up as a follow-on.

Project details

- Purpose of the project: verify that the lock is followed unlock on all feasible paths for every locking instance in the Linux kernel.
- The details of the project are provided in a separate document.
- A link is given in the project document to a website that provides the graph models for all the locking instances in a recent version of the Linux kernel.

Overarching ideas for mission-critical verification

Intelligence Amplifying Automation (Frederick Brooks): "If indeed our objective is to build computer systems that solve very challenging problems, my thesis is that IA > AI, that is, that intelligence amplifying systems can, at any given level of available systems technology, beat AI systems. That is, a machine and a mind can beat a mind-imitating machine working by itself."

Automation Amenable to Human Scrutiny (De Millo, Perlis, and Lipton): "In mathematics, the aim is to increase the human confidence in the correctness of a theorem. [A] proof [does not] settle the matter, contrary to what its name suggests, a proof is only one step in the direction of confidence. Verification is nothing but a model of believability. It cannot be a model where proofs are accepted in blind faith. A proof should be amenable to human scrutiny. *A good proof is one that makes us wiser.*"

The role for automation in mission-critical software verification

- *Prove properties* of programs to find and verify vulnerabilities, where
  - find = identify the relevant program artifacts + hypothesize the vulnerability
  - verify = ascertain the hypothesized vulnerability
- *Produce evidence* to support human scrutiny of verification by the automation.
- *Produce evidence* to amplify human intelligence to complete the verification tasks where automation falls short.
- Examples: Automation such that,
  - it can prove loop termination and produce the termination pattern.
  - if it cannot prove the termination, but it still helps the human to reason about the termination by producing relevant loop artifacts and their characteristics.

Our pragmatic approach to advance automation

- *Advances* are designed to:
  - Increase the share of properties that can be proved automatically
  - Enhance the evidence quality for human scrutiny and for amplifying human intelligence
- *Experimentation Capabilities* to advance theory and automation:
  - Quick prototyping of new property verifiers
  - Exploration to discover new concepts to formulate properties
  - Developing new ideas for high-quality evidence

On experimentation: Gauss declared that his way of arriving at mathematical truths was through systematic experimentation — his notebooks attest to it. As mathematicians do, we need to perform experiments; experiments that motivate future development of theory and automation technology.

Our practical framework

- *Graph Paradigm*:
  - *eXtensible Common Software Graph* (XCSG) schema to capture language semantics – currently works across Java, Java byte code, C, C++
  - *Graph database* of pre-computed program artifacts and relationships
  - Q*uery language* to traverse and transform program graphs
  - *Analysis composition* through the medium of graphs
  - *Interactive visualization, source correspondence* and a query *interpreter*
  - *Plug-in architecture* to build powerful toolboxes that leverage these capabilities
- *Benefits*:
  - Enables quick prototyping of property verifiers
  - Provides a rich infrastructure for experimenting
  - Helps to develop language-independent graph models to enhance scalability and accuracy of analyses
  - Facilitates the leveraging of graph theory to develop new concepts

An overview of Linux verification with our tool

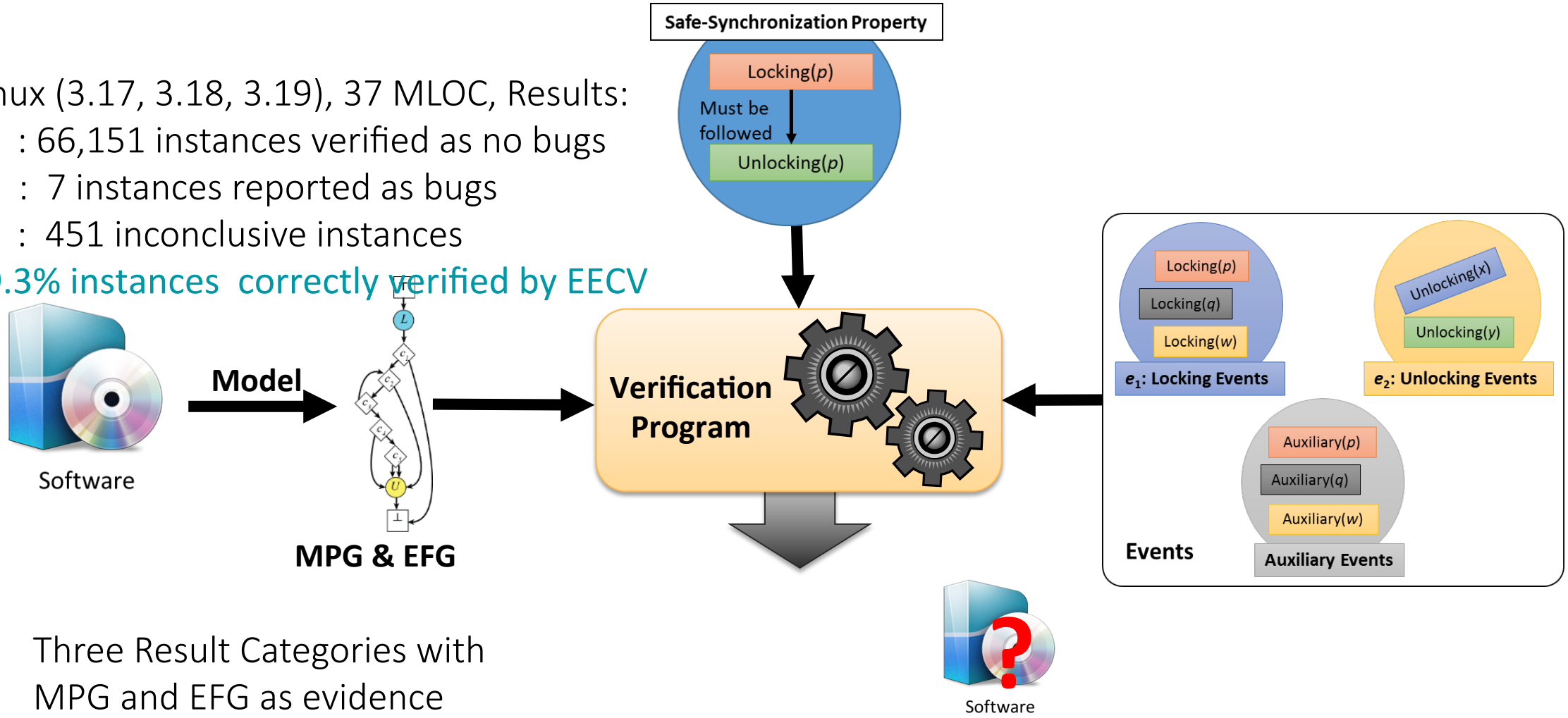
Linux (3.17, 3.18, 3.19), 37 MLOC, Results:

✓ : 66,151 instances verified as no bugs

X : 7 instances reported as bugs

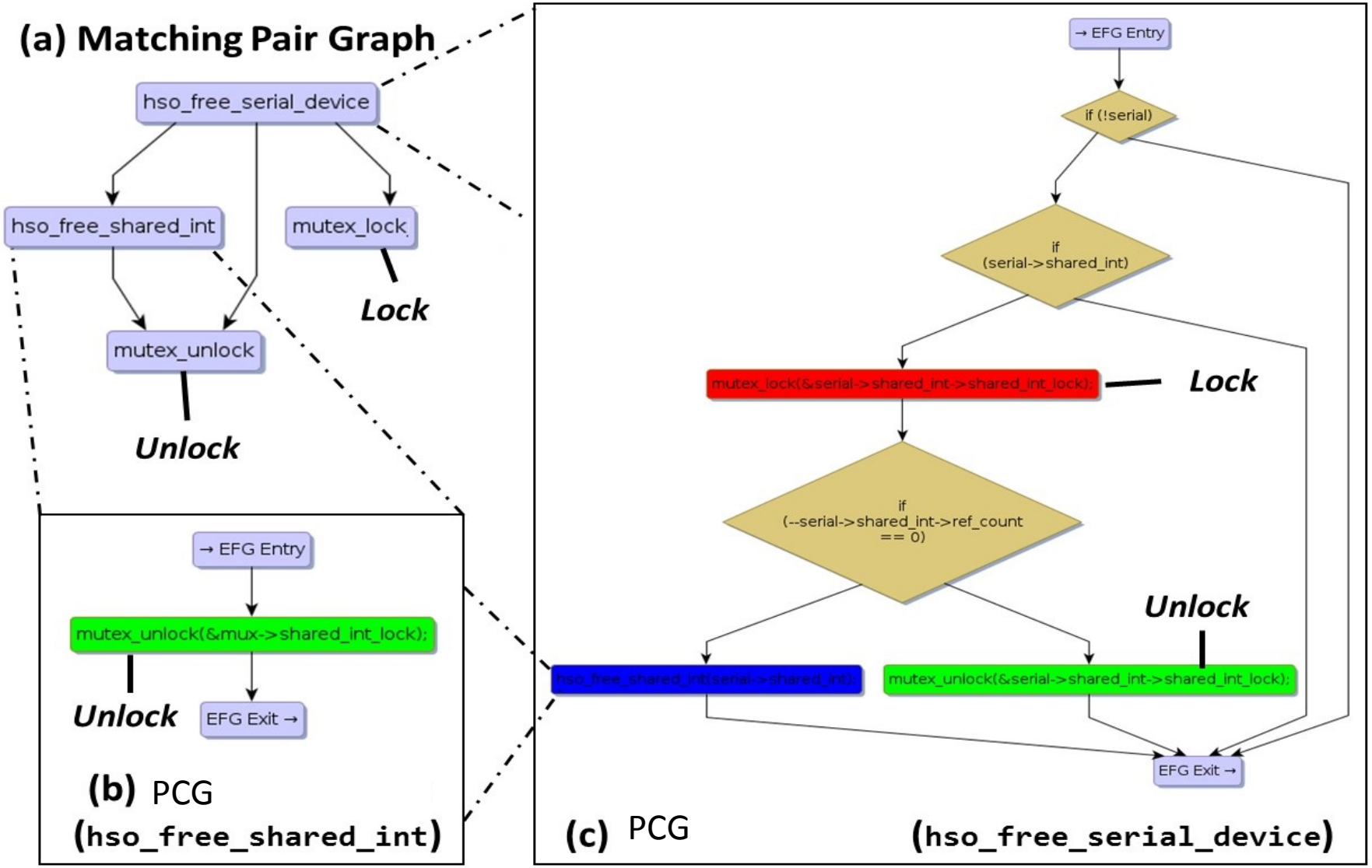
? : 451 inconclusive instances

99.3% instances correctly verified by EECV

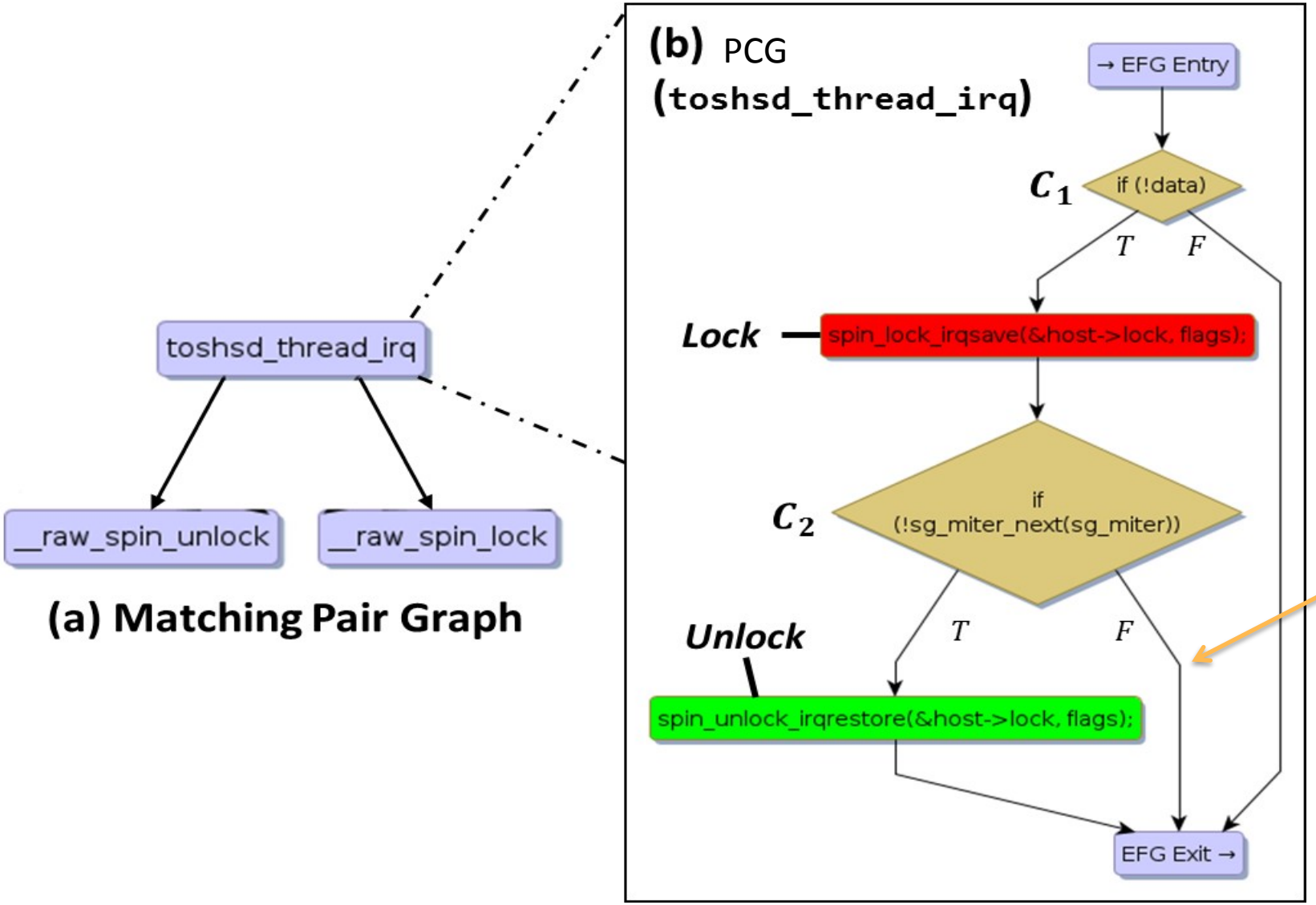


Three Result Categories with MPG and EFG as evidence

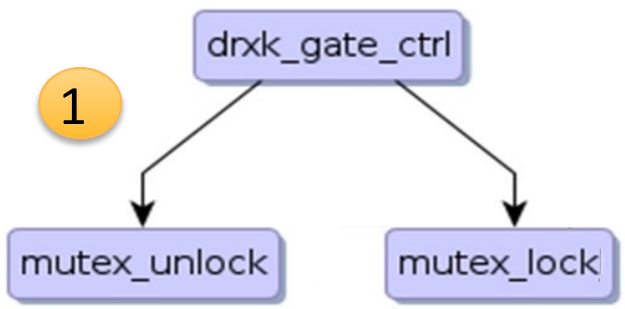
An illustration of an automatically verified instance



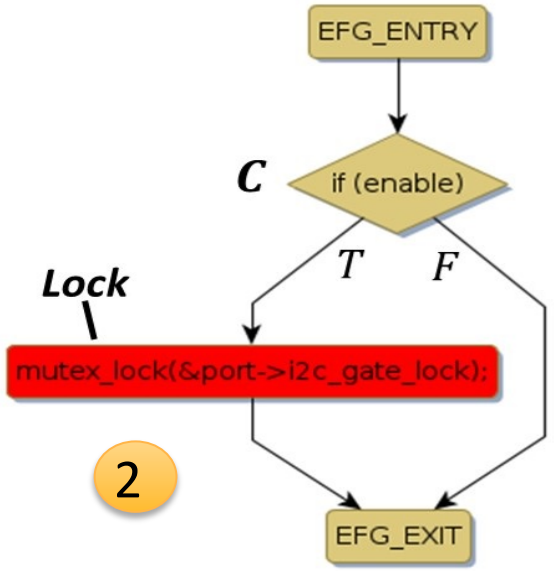
An illustration of automatically discovered bug



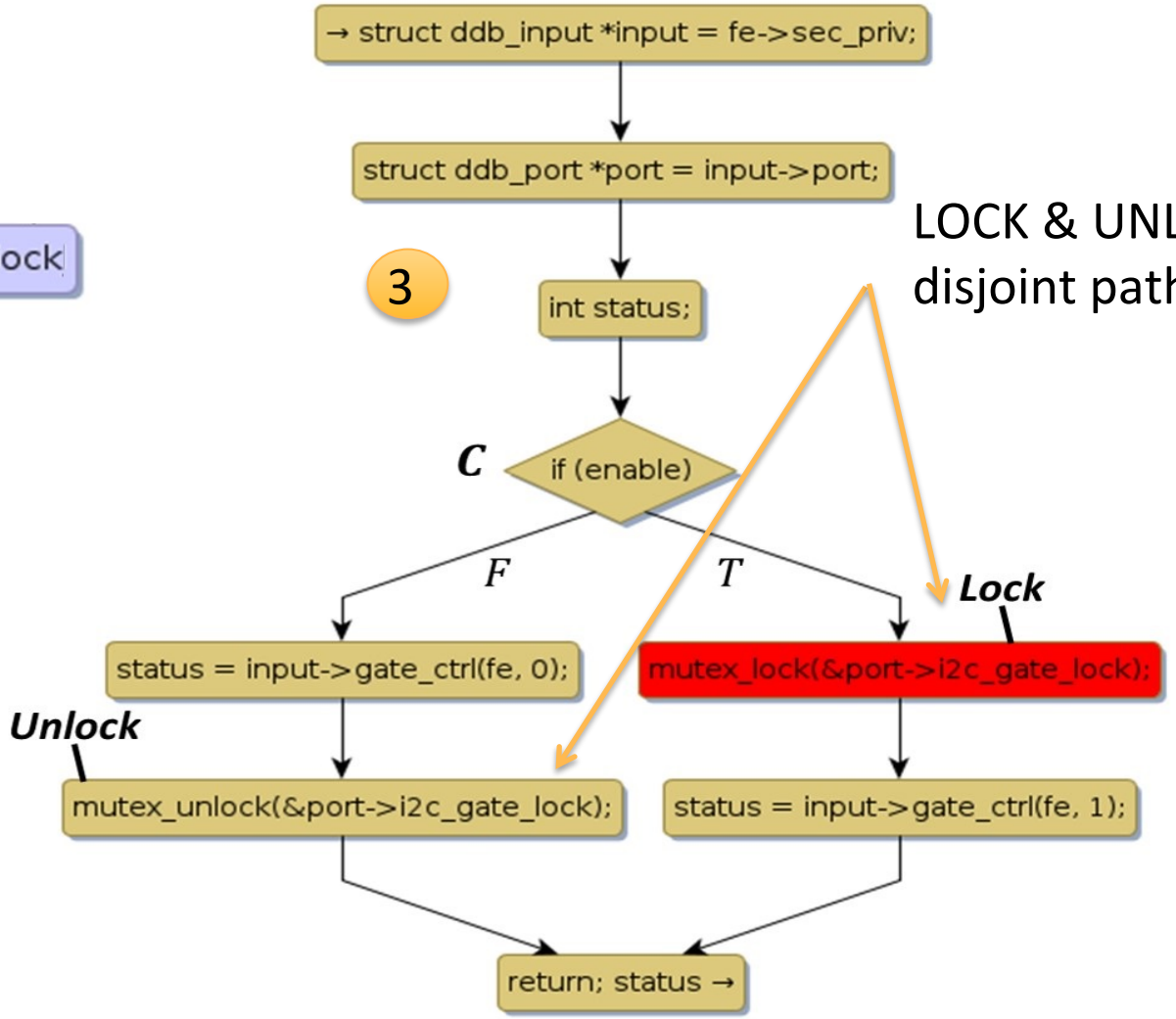
Evidence produced for the bug



(a) MPG



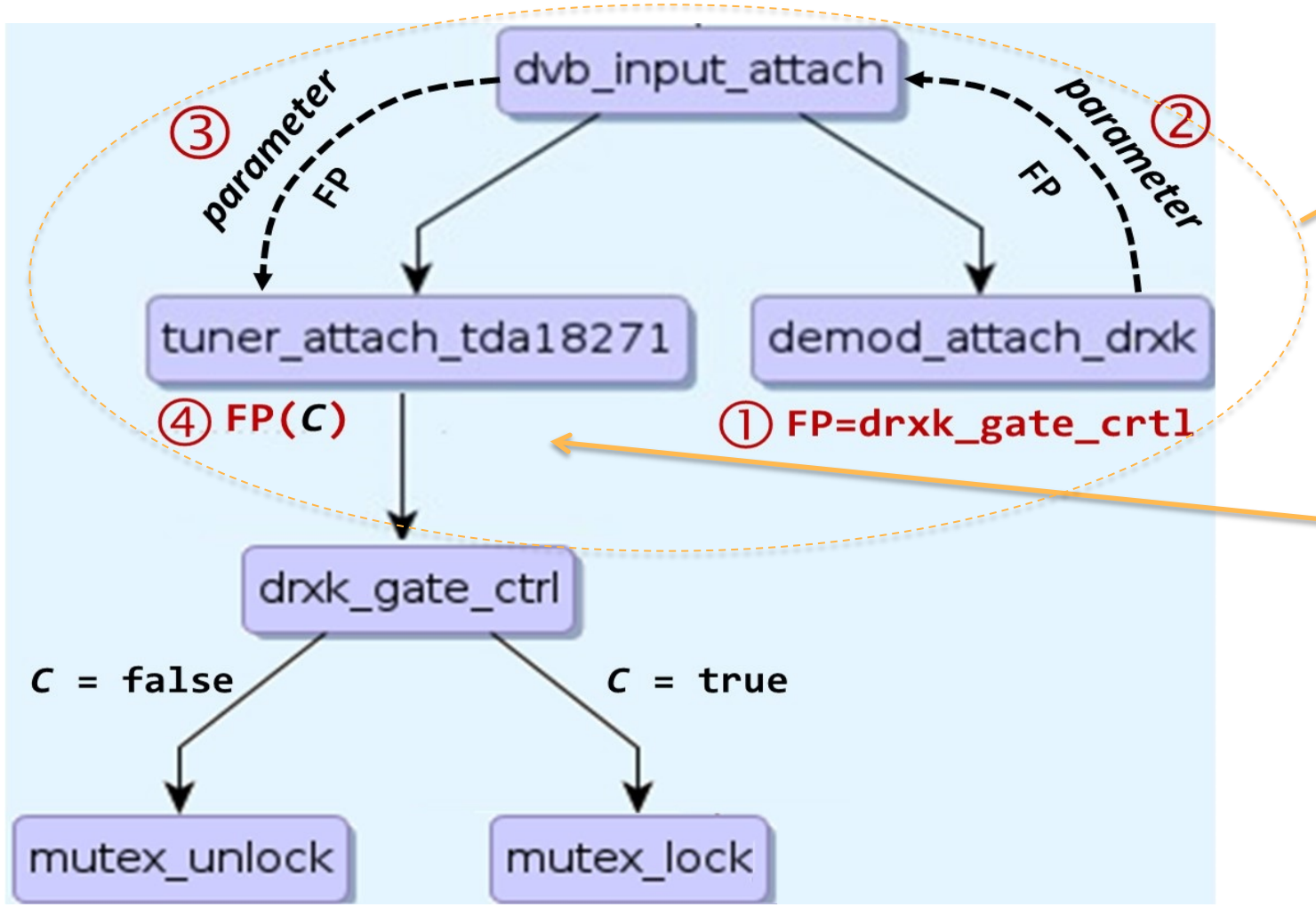
(b) PCG drxk_gate_ctrl



LOCK & UNLOCK on disjoint paths

(c) CFG for drxk_gate_ctrl

Automatic evidence + manual effort for highly complex bug



Manual Effort: A set of queries to find where the function pointer (FP) is set to the function being investigated, and to find the functions that make calls using FP

The function called twice using FP, first call acts like LOCK and second call acts like UNLOCK

Bug: The usage is correct on all but one path where the second call does not happen

Linux Lock/Unlock Stats

- Versions 3.17-rc1, 3.18-rc1, 3.19-rc1
  - 37 MLOC & 66,609 instances
  - 62,663 Intra-Procedural (MPG size =1), average MPG size 1.3, the maximum MPG size 40 for one MPG
  - 55,251 functions in the union of all MPGs
  - 4 nodes per EFG, the maximum EFG size 63 for one EFG

CFG to EFG Reduction – Top Ten

Linux version 3-19-rc-1

Function Name	Nodes		Edges		Branch Nodes	
	CFG	EFG	CFG	EFG	CFG	EFG
client_common_fill_super	1,101	15	1,179	28	249	13
kiblnd_create_conn	731	18	925	34	197	15
CopyBufferToControlPacket	392	20	559	39	180	18
kiblnd_cm_callback	662	38	831	56	170	15
kiblnd_passive_connect	622	22	784	44	164	20
dst_ca_ioctl	349	2	518	1	163	0
qib_make_ud_req	621	10	821	15	156	5
cfs_cpt_table_al	522	7	672	13	153	6
private_ioctl	569	16	732	24	148	8
vCommandTimer	490	47	623	75	143	28

Our automatic verification results

- Linux (3.17, 3.18, 3.19), 37 MLOC, LOCK-UNLOCK MPV
- SAP Verification Results:
  - Automatically verified 66,151 instances with no violation (Category-1)
  - Automatically verified 7 instances with no violation (Category-2)
  - 451 Instances of inconclusive automatic verification(Category-3)
- 99.3% instances automatically verified with no false positives or negatives

Inconclusive Verification – Sources of Difficulty

99.3% automatically verified with remaining 451 Instances of inconclusive automatic verification.

Kernel	Type	C3	Intra-procedural Feasibility	Inter-procedural Feasibility	Function Pointers	Signature Problems
3.17-rc1	spin	82	14	16	22	30
	mutex	73	10	31	28	4
3.18-rc1	spin	74	13	20	23	18
	mutex	92	13	35	39	5
3.19-rc1	spin	77	12	16	28	21
	mutex	53	6	22	21	4
All Kernels		**451**	**68**	**140**	**161**	**82**

Targets for the future tool improvements:

1. Automated path feasibility checking.
2. Automated resolution of function pointers to complete call graphs.
3. Improve data flow techniques to handle identified patterns of passing pointers.