



Working
Together



1908

2002

Scalable Program Comprehension for Analyzing Complex Defects

ICPC 08 Presentation

S. C. Kothari

Iowa State University & EnSoft Corp.

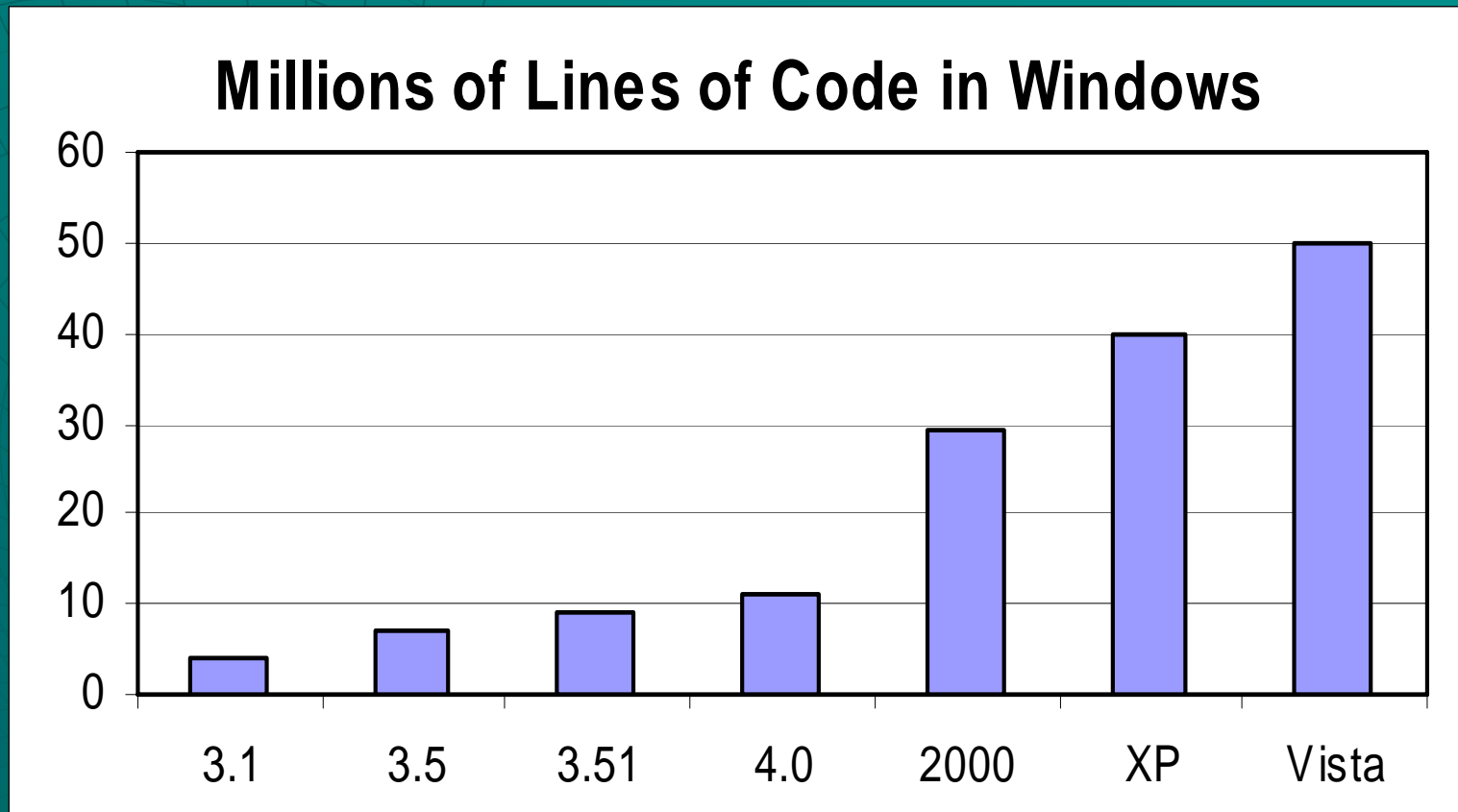
Email: kothari@iastate.edu

Operating System/360

- ◆ The not-unexpected passing away of OS/360 in its 21st release – August 2, 1972.
- ◆ Obituary:

The offspring first saw the light of day in December 1965 and the birth announcement recorded a weight of 64K. It rapidly became apparent that OS, in spite of its unusual size, was more than normally subject to childhood diseases. For a long period, this weak and sickly baby hovered close to death despite almost continuous transformations and major transplants of several vital organs. Many experts are of the opinion that the huge weight of OS at birth contributed greatly to its early ill health. OS is survived by two lineal descendants, OS/VS1 and OS/VS2. It will be mourned by its many friends and particularly by the over 10,000 system programmers throughout the world who owe their jobs to its existence.

Windows Operating System



A Classroom Experience

- ◆ In an operating system course project, a student spends:
 - *40* hours in identifying and understanding the relevant parts of code.
 - *2* hours in making the actual code changes to incorporate the specified functionality.
 - *10* hours in testing and debugging the code.

Software Reliability – Huge Problem



Infamous Ariane 5 disaster, caused by a bug in the rocket's control software.

Problem Solving with PC Tools

- ◆ Problem: A clear definition including the variations.
- ◆ Solution:
 - Estimating the work and the cost
 - Ease of applicability
 - Scalability to large software
 - Differentiating factors

Matching Pair (MP) Defects

- ◆ Defect - if certain *program artifacts* are not in matching pairs.
- ◆ A wide array of MP defects: non-matching parentheses, memory leaks, synchronization problems etc.
- ◆ Different levels of comprehension complexity.

Levels of Complexity

- ◆ Four Levels:
 - Level I involves knowledge of syntax
 - Level II involves knowledge of control flow
 - Level III involves knowledge of control flow and data flow
 - Level IV involves knowledge of control flow, data flow, and control transfer.

MP-1 Defects

- ◆ Syntactic program artifacts such as parentheses.
- ◆ Matching Constraint:
 - *Local*: matching must be within a program statement or a block.
 - *LIFO*: Different types of artifacts must individually match according to the *LIFO* property.

MP-2 Defects

- ◆ Matching involves control flow.
- ◆ Matching must happen on all *feasible execution paths*.
- ◆ Example: matching pairs of functions to *disable* and *enable* interrupts.

Feasible vs. non-feasible paths

```
Read (X);  
A = B = C =5;  
If (X > 5)  
    enable();  
If (X == 0)  
    A = B + C;  
If (X > 5)  
    disable();  
Print (A,B,C);
```

Note that enable() and disable() match on all feasible control paths. There are infeasible control paths on which they do not match.

MP-3 Defects

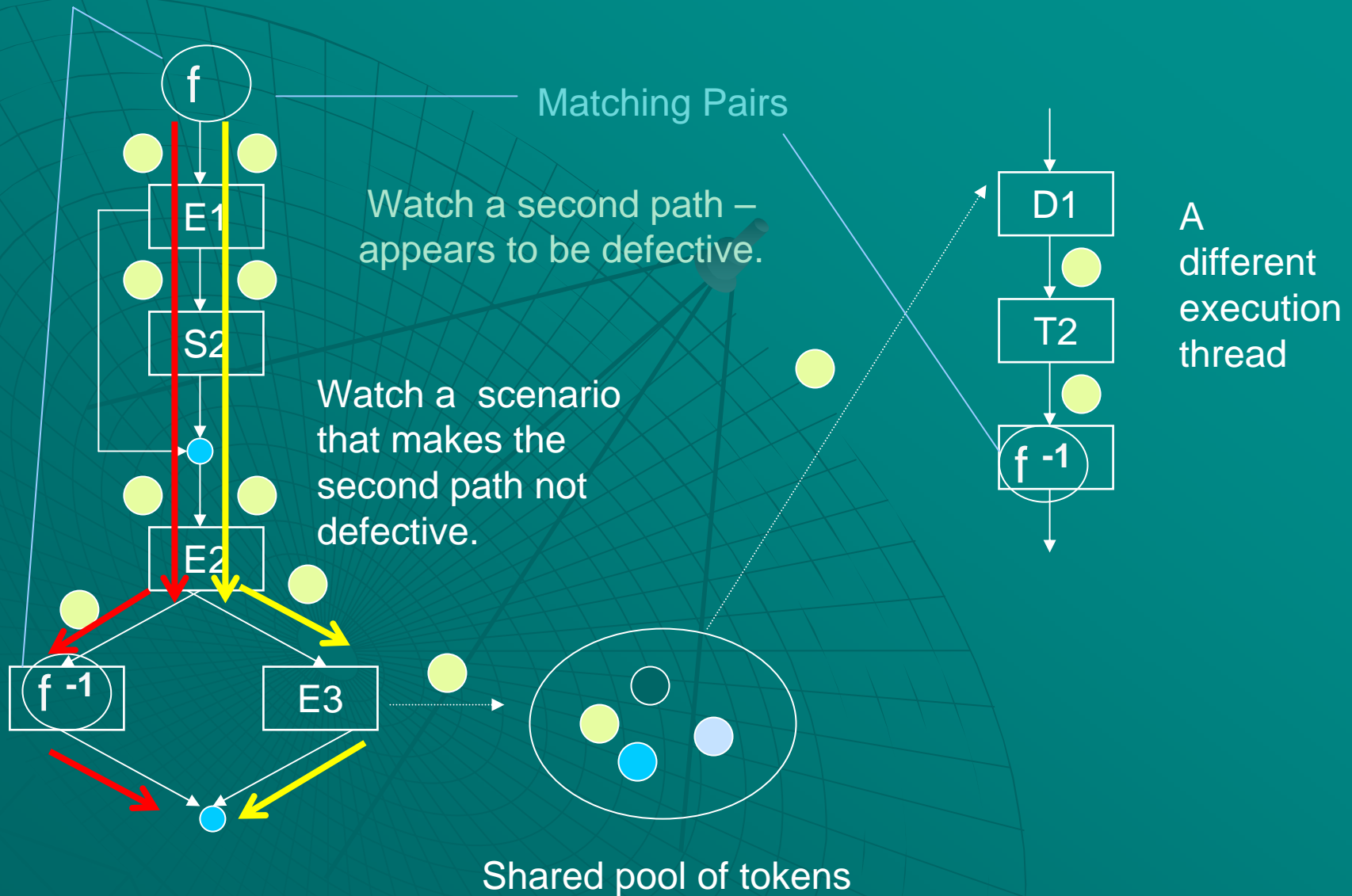
- ◆ Matching Constraint:
 - Involves *control flow* and *data flow*.
 - Matching must happen on all *feasible execution paths*.
 - Matching involves data elements.
- ◆ Example: memory leaks
 - Allocation not matched by deallocation.
 - Matching requires allocation and deallocation to have pointers to the same memory location.

MP-4 Defects

- ◆ Similar to MP-3, but complicated by concurrent and interrupt processing.
- ◆ Implication - the feasible execution paths may not be directly linked by control flow.
- ◆ Example: memory leaks
 - Allocation may be matched by deallocation across a different thread or deallocation done by an interrupt service routine.

Watch one non-defective path.

Matching Pairs



Knowledge-Centric Software (KCS) tools technology

... Work at EnSoft and ISU

IOWA STATE UNIVERSITY

ISU Search Search Engines

[E-Mail/Phones](#) | [Calendar](#) | [Maps](#) | [Contact Us](#)

- [Students](#)
- [Alumni/Parents](#)
- [Visitors](#)
- [Faculty/Staff](#)
- [Business/Industry](#)
- [Employers/Job Seekers](#)

- [About Iowa State](#)
- [Admissions](#)
- [Athletics](#)
- [Colleges/Departments](#)
- [Diversity](#)
- [Giving to Iowa State](#)
- [Information Technology](#)
- [Library](#)
- [Outreach/Extension](#)

Sign-Ons

(For Students, Faculty, Staff)

- [AccessPlus](#)
- [WebCT](#)
- [WebMail](#)



Video: Adventure of a lifetime

Explore the student experience at Iowa State University.

News & Events

[ISU Army ROTC brigade named top unit in western U.S.](#)

[Celebration kicks off program to advance women scientists](#)

[Students raise nearly \\$161,000 for children's hospital](#)

[President's report, "Where Breakthroughs Happen" \(PDF\)](#)

► [Latest news, updates.](#)

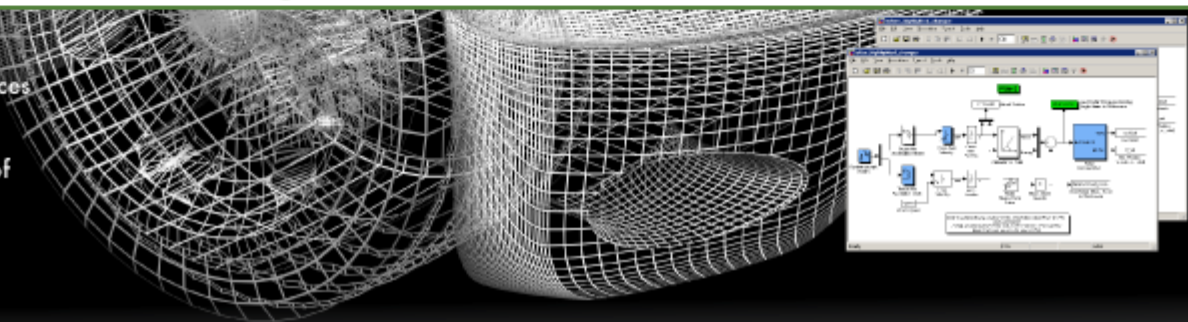
Putting biorenewables to work

ISU scientists spin soy into plastic, crops into fuel, even manure into oil.

SimDiff 2.0

Automatically find differences between Simulink models.

Try it out! Request a trial of SimDiff.



New in 2.0: Stateflow and TargetLink Support.

Rubric

Rubric checks C/C++ code for compliance to your coding standards. Rubric identifies violations of conventions such as:

- Names of variables must begin with a lower case letter.
- Switch clauses must always end with a

SimDiff

SimDiff compares two Simulink model files for additions, deletions, and changes to:

- blocks
- lines
- configuration parameters
- model properties
- annotations

Total Insight

COBOL developers spend substantial time finding program, file, SQL, and data interactions. COBOL Total Insight's easy-to-use interface instantly reveals interactions in multi-million line codes.

COBOL Total Insight has

Products

- Rubric
- SimDiff
- Total Insight

By Platform

- Simulink: SimDiff
- C/C++: Rubric
- COBOL: Total Insight

Tools to amplify human intelligence

Fredrick Brooks:

"... 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."

Query-Model-Refine (QMR) Technique

- ◆ A natural way to *amplify human intelligence* by assisting in:
 - Retrieval of information by analyzing software.
 - Generation of visual models from the retrieved information.
 - Refinement of the models to manage complexity.

A Demonstration

- ◆ Defect analysis using the QMR technique to program comprehension.
- ◆ We will show how to analyze the Linux operating systems for MP defects.

A Defect Analysis Problem

- ◆ Problem: analyze the Linux 2.6 kernel for MP defects w.r.t. `mutex_lock` and `mutex_unlock` functions.

IA Approach

- Define a comprehension strategy to solve the problem.
- Use the tool to execute the strategy.
- Implication: To design useful tools, prior understanding of problems and solution strategies is important.

A Defect Analysis Solution

- ◆ Design a sequence of solution steps.
- ◆ Define the query, model, or the refinement to be done at each step.
- ◆ Quantify the work in an early stage of the solution process.
- ◆ Argue that all the possible cases are handled.

Step 1

- ◆ Divide the problem in two cases:
 - Case 1: the `lock` and `unlock` are called within the same function.
 - Case 2: the `lock` and `unlock` are not called within the same function.
- ◆ We will follow up case 2.
- ◆ Execute queries to obtain a list of functions that call `lock` but *not* `unlock`.

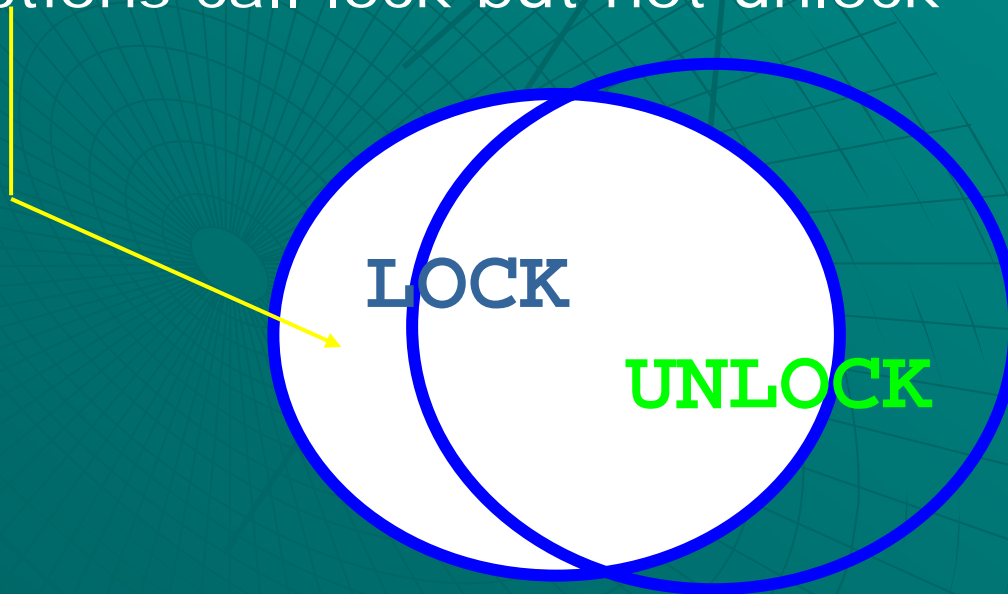
Result 1

401 functions that call lock

436 functions that call unlock

51 functions call unlock but not lock

16 functions call lock but not unlock



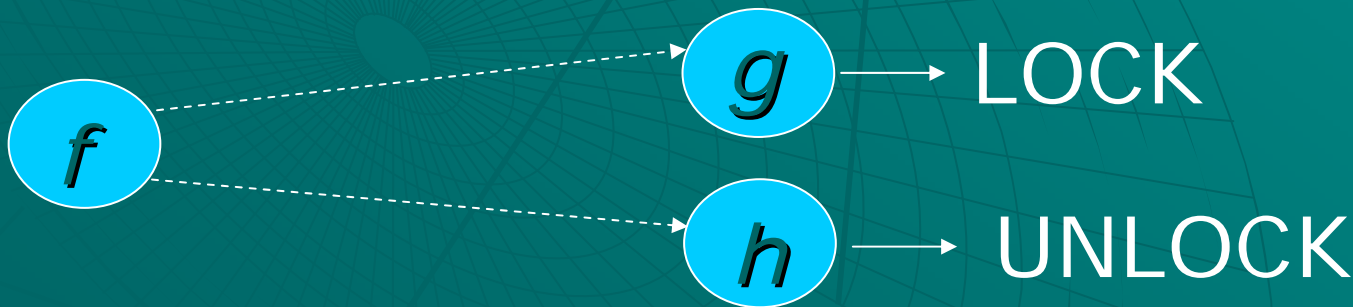
A function calls lock but not unlock

```
static void *diskstats_start(struct seq_file *part, loff_t *pos)
{
    loff_t k = *pos;
    struct list_head *p;

    mutex_lock(&block_subsys_lock);
    list_for_each(p, &block_subsys.kset.list)
        if (!k--)
            return list_entry(p, struct gendisk, kobj.entry);
    return NULL;
}
```

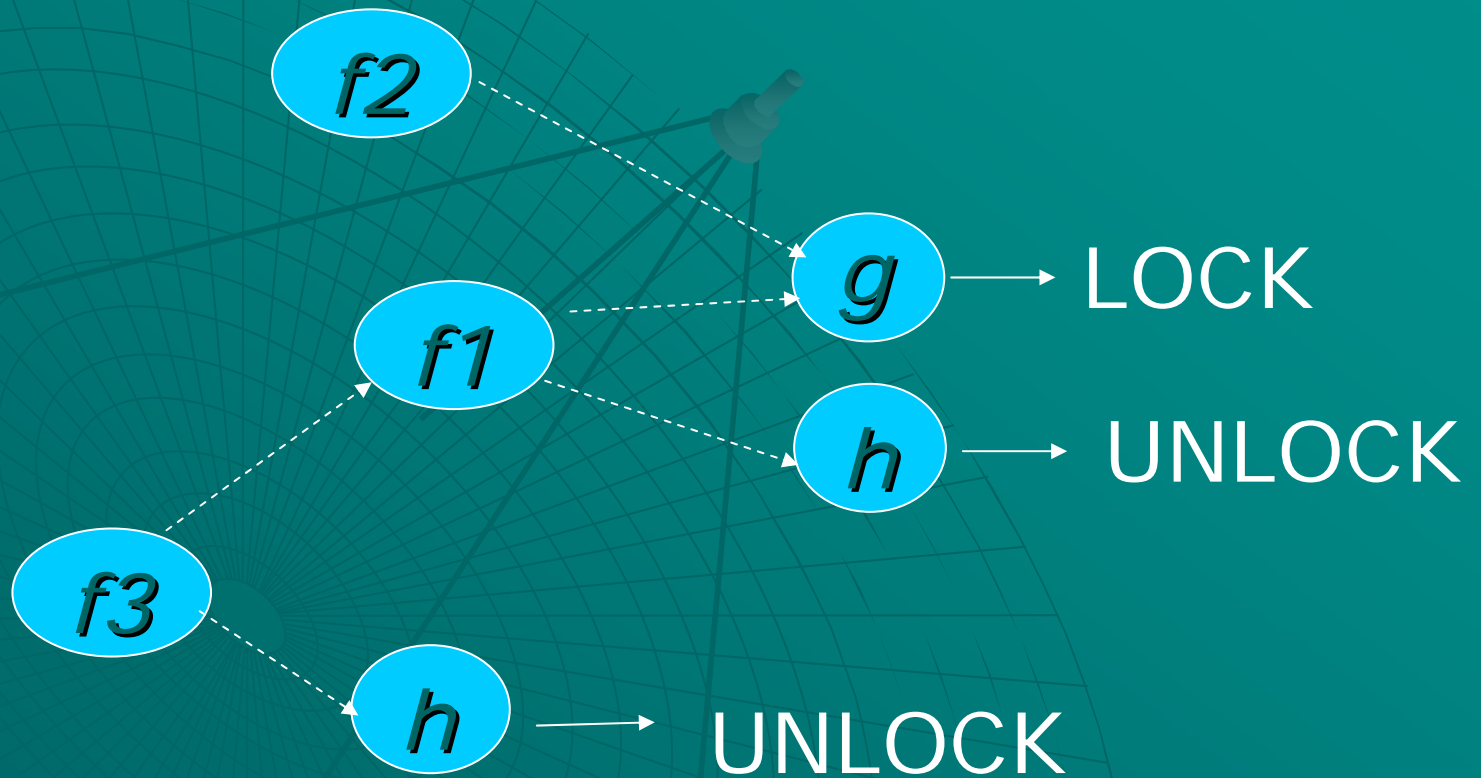
Checking a Possibility

- ◆ Analyze these 16 functions that call lock but not unlock.
- ◆ For such a function g check the possibility: an ancestor f of g calls unlock (directly or indirectly).





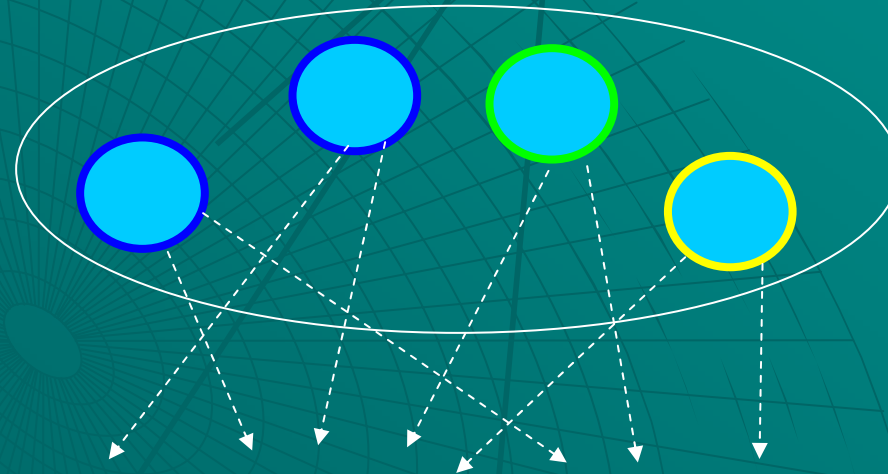
What should be the query to find
the ancestor?



Step 2

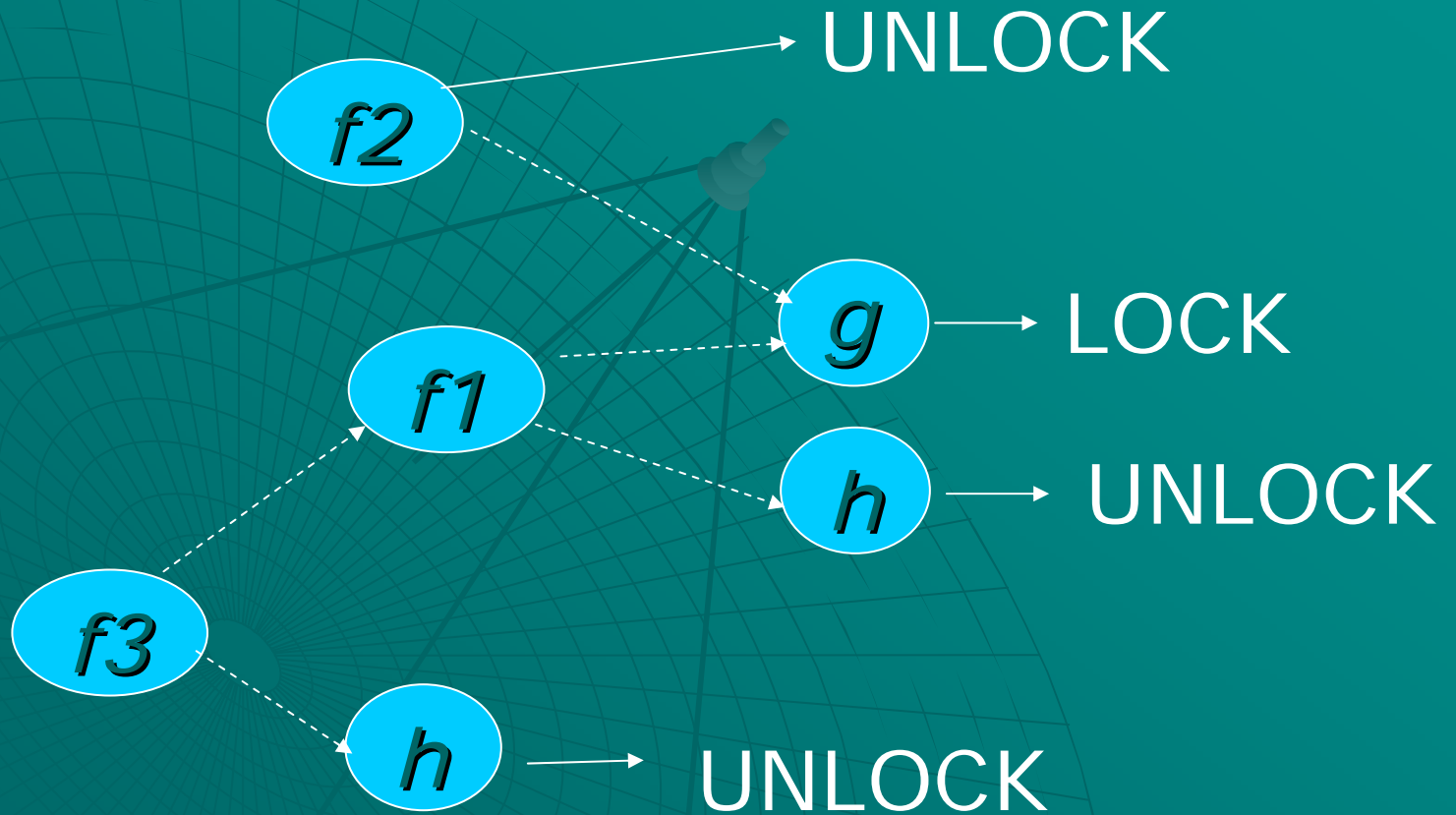
- ◆ Execute a query to find the *roots* of *reverse call graph* with the given 16 functions as leaves.

183 roots



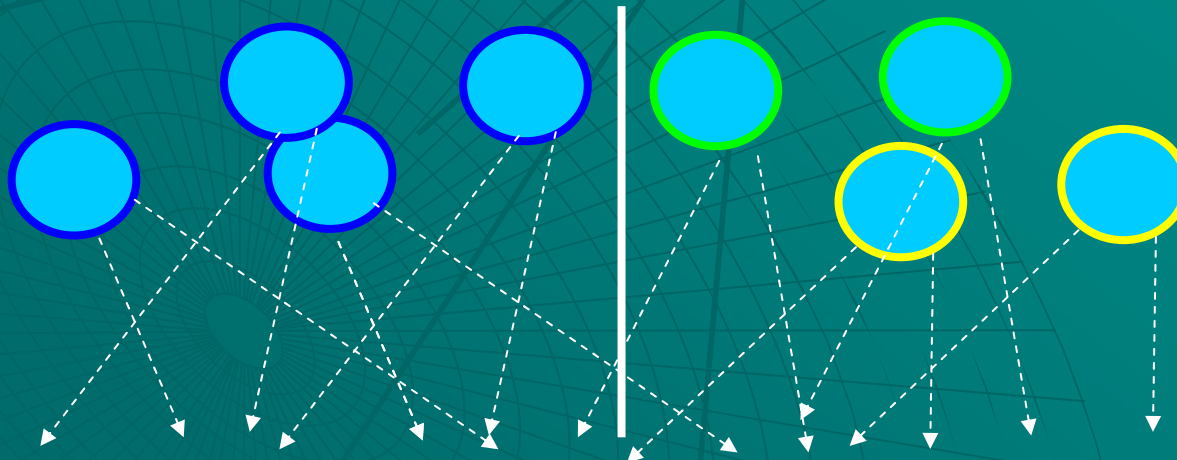
16 functions that call lock but not unlock

Separate threads or interrupt service routines



Step 3

- ◆ Partition the roots into groups of related functions.

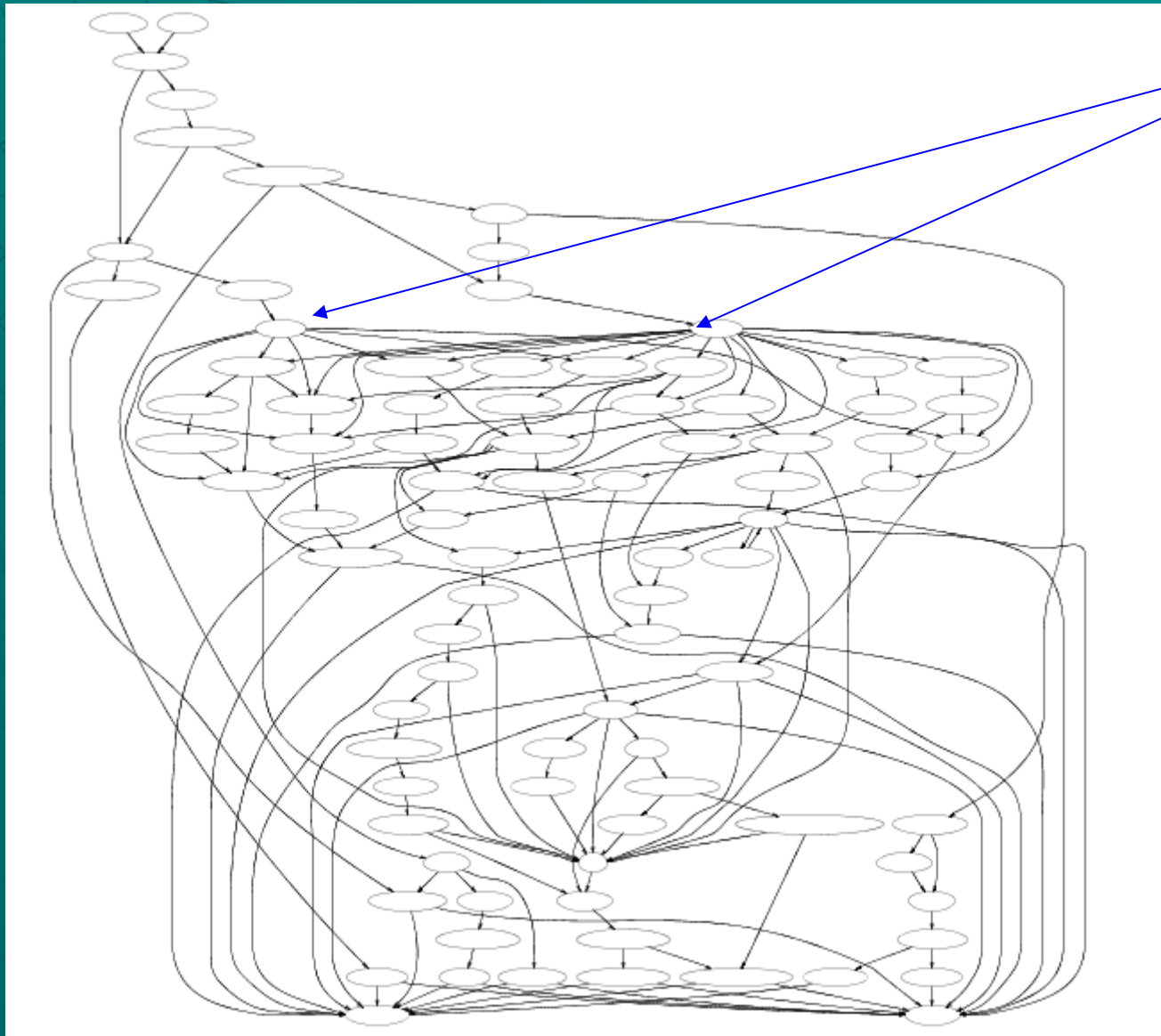


16 functions that call
lock but not unlock

Step 4

- ◆ Model: Call tree with a selected group of roots and lock and unlock as leaves.
- ◆ For demonstration, we will select:
 - `idecd_ioctl`
 - `idedisk_ioctl`
- ◆ Note that the ancestor *f* where the lock and unlock can be matched, if it exists, can be found through the above model.

Result



Represents two cases which can be analyzed separately

87 nodes

Step 5

- ◆ Refine the Model: Omit one case at a time.
- ◆ This refinement is achieved by a graph transformation provided by Atlas.

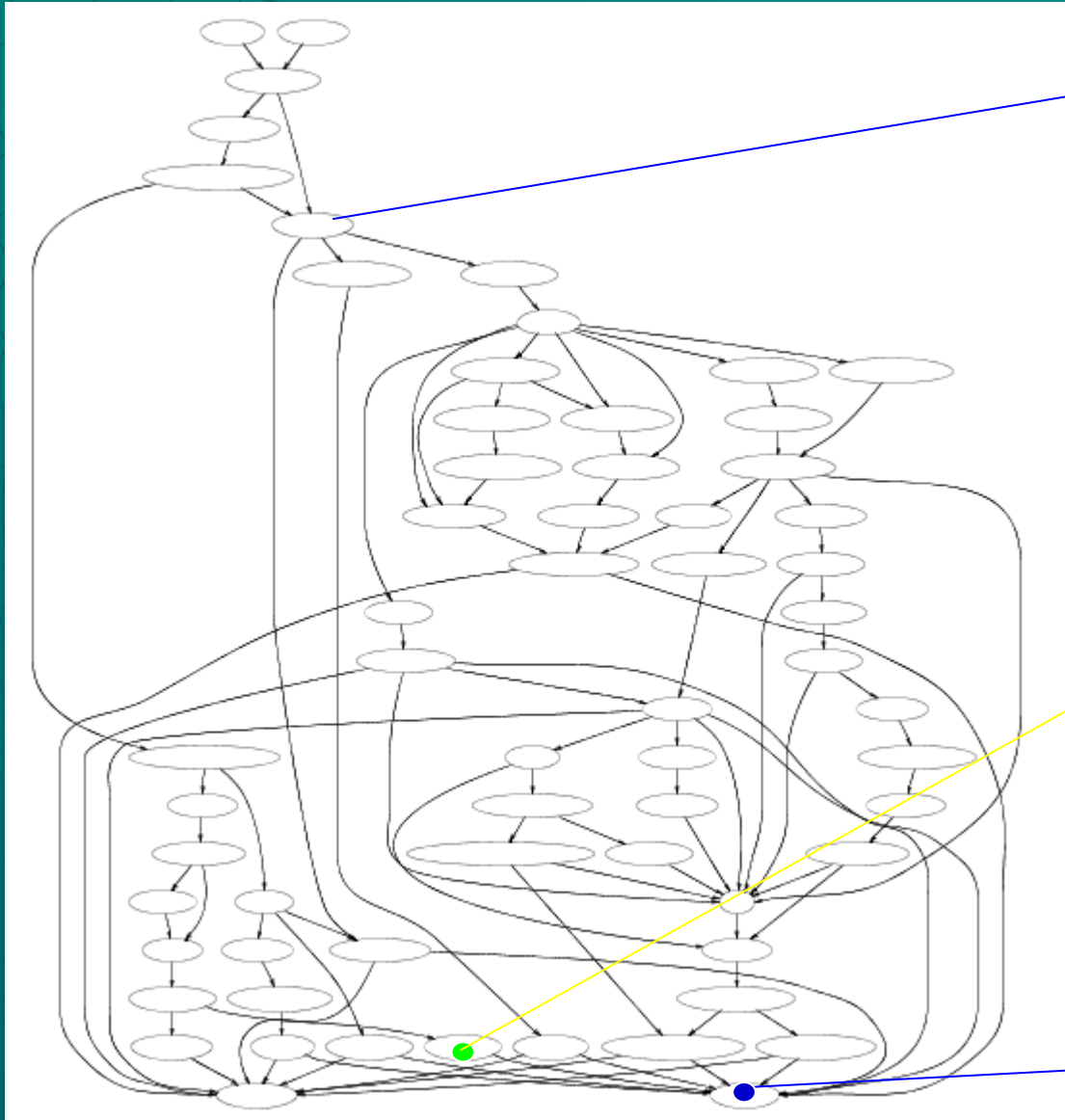
62 nodes

Result

This is not an ancestor of the function g

The function g that calls LOCK but not unlock.

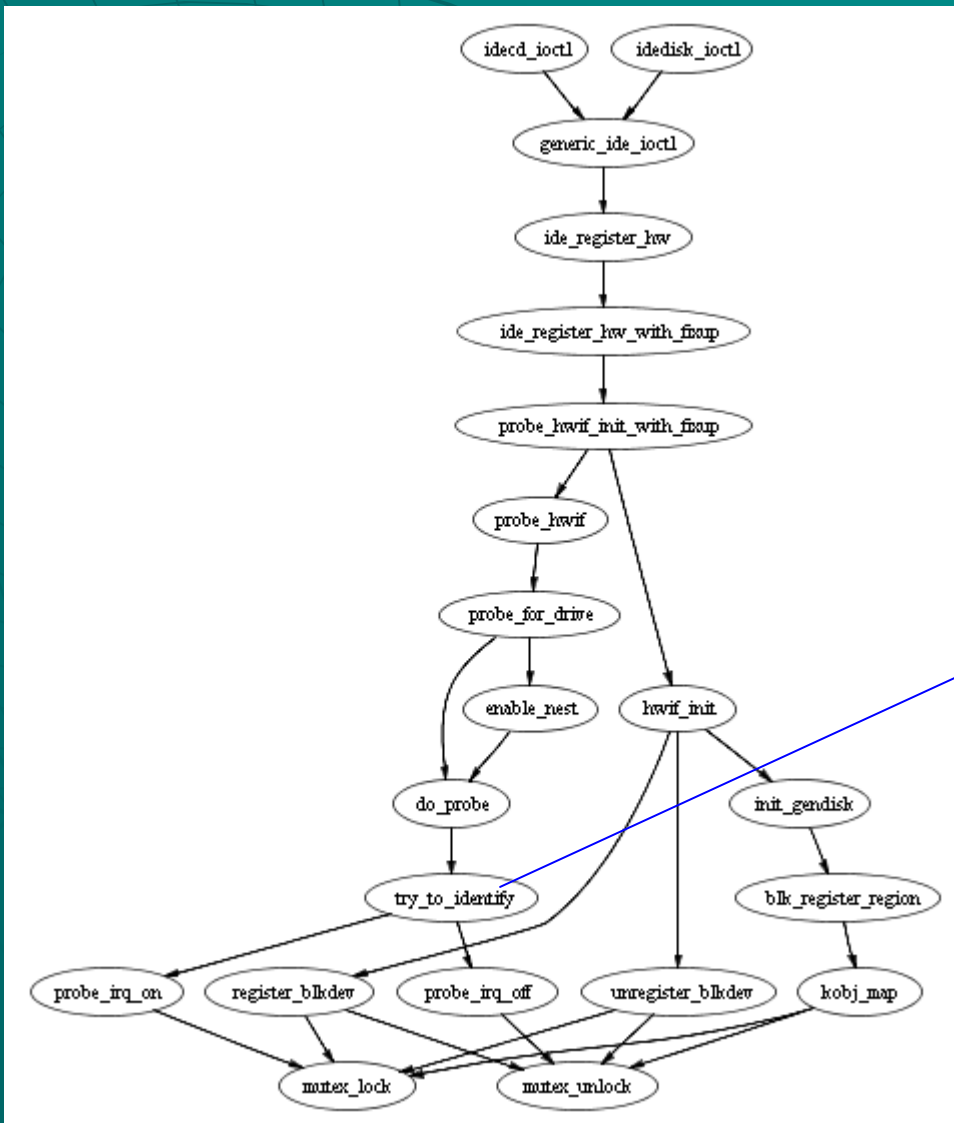
Lock



Step 6

- ◆ Further Refinement: Omit the part which is not an ancestor of g .
- ◆ This refinement is achieved by a graph transformation provided by Atlas.

Result

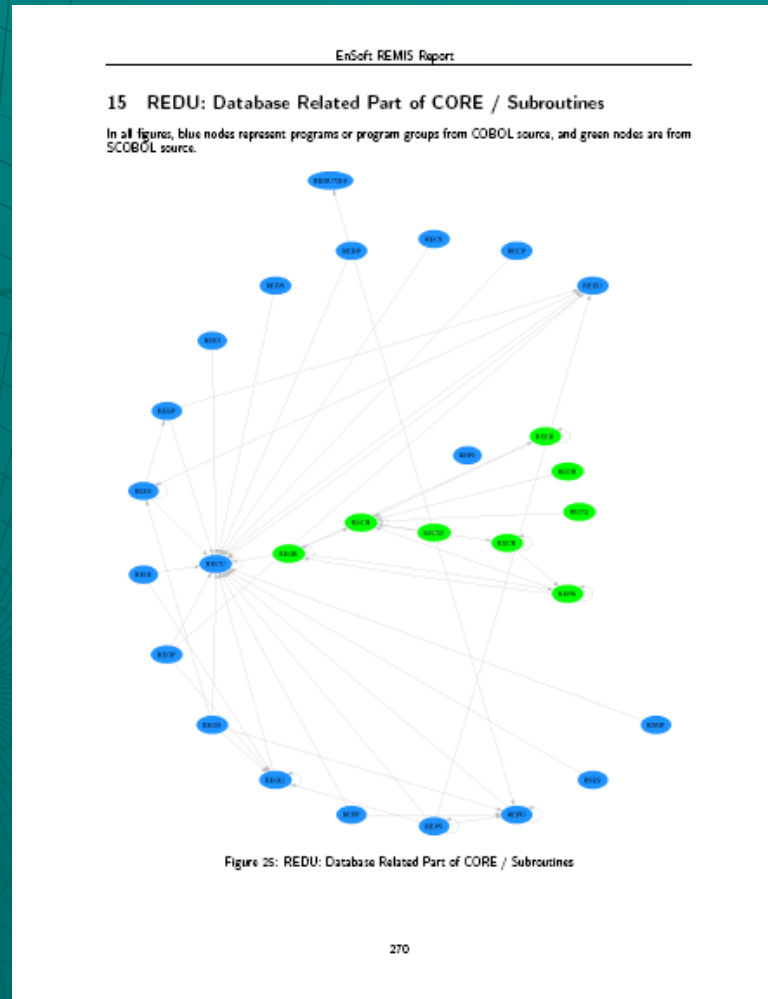


The ancestor where the lock and unlock call can be matched.

Questions ??

- ◆ Would like to discuss:
 - Program comprehension tools
 - Program comprehension problems and solution strategies
 - Query languages
 - Use of graph theory in program comprehension

Total Insight – A COBOL Tool



SimDiff – A Model Differencing Tool

The screenshot displays the SimDiff software interface, which is used for comparing two models. The interface is divided into several sections:

- Left Panel (SimDiff):** Contains configuration options for the models being compared. It includes fields for "First Model (Before)", "Second Model (After)", and "Output Folder". Below these are options for "Attributes Filter" and "Highlight" (Additions, Changes, Deletions). A "Blocks/Lines" section shows a list of 10 blocks, with the first two checked: "1. Cross-Axis Velocity" and "2. Radar Kalman Filter".
- Top Window (before_highlighted_changes *):** Shows a Simulink model diagram. The diagram includes blocks for "Random aircraft motion", "Cross-Axis Acceleration Model", "Long Velocity", "Cross-Axis Velocity", "Cross-Axis Position", "Cartesian to Polar", "Radar Measurement Noise", "Radar Kalman Filter", and "Measurement_noise". A red highlight is visible on the "Filter1" block.
- Bottom Window (after_highlighted_changes *):** Shows a similar Simulink model diagram. A green highlight is visible on the "Filter2" block. A new block, "Send Radar Range and Bearing Angle Meas. to Workspace", is added to the diagram. The diagram also includes blocks for "Random aircraft motion", "Cross-Axis Acceleration Model", "Long Velocity", "Cross-Axis Velocity", "Cross-Axis Position", "Cartesian to Polar", "Radar Measurement Noise", "Radar Kalman Filter", and "Measurement_noise".

References

- ◆ Fredrick Brooks, *The Computer Scientist as Toolsmith*,
 - <http://www.cs.unc.edu/~brooks/Toolsmith-CACM.pdf>
- ◆ Earliest paper on software complexity:
 - Rubey, R.J.; Hartwick, R.D.: Quantitative Measurement Program Quality. ACM, National Computer Conference pp. 671-677, 1968.
- ◆ Knowledge-Centric Software Research Laboratory at ISU
 - <http://dirac.ece.iastate.edu/sec/>
- ◆ EnSoft Corp.
 - <http://www.ensoftcorp.com/>