MILCOM 2016

SECURE COMMUNICATIONS AT THE SPEED OF CYBER

Graph Compactification for Efficient Program Comprehension and Analysis

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

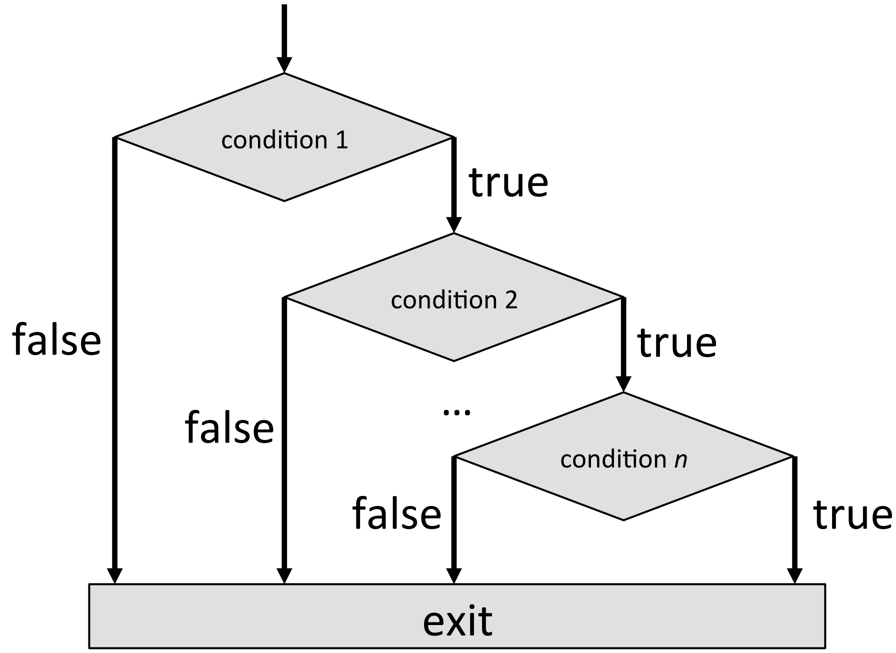
Ben Holland, Iowa State University

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

BALTIMORE, MD • NOVEMBER 1–3, 2016

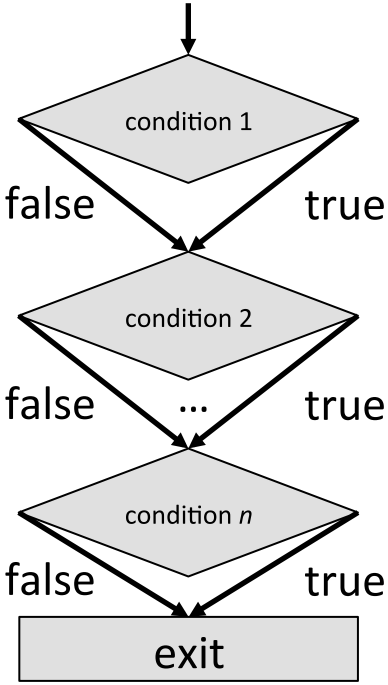
Counting Paths

- How many paths are possible for *n* nested conditions?
  - Answer: *n*+1 paths



Counting Paths

- How many paths are possible for *n* non-nested conditions?
  - Answer: 2^n paths

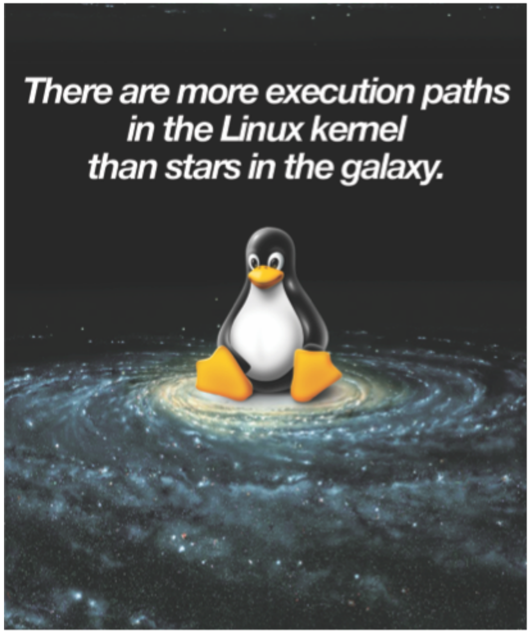


Counting Paths

- How many paths are *feasible* if c1 == c2?
  – i.e. How many paths could produce valid runtime execution traces?
  – More or less?

Counting Paths

- In the worst case all conditions are non-nested and all paths are feasible.
  - Number of paths to consider in software is exponential!
  - In reality the number of feasible paths is much smaller.

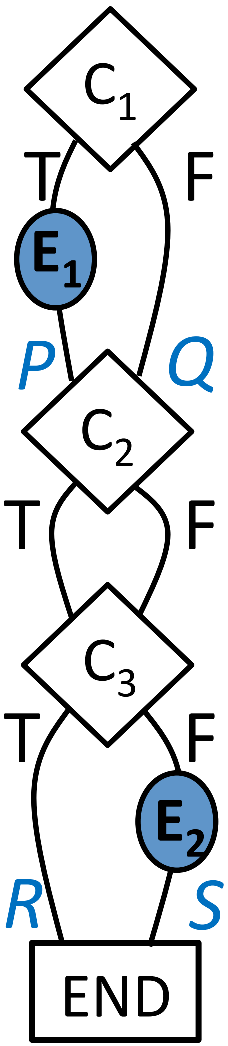


Intuition: Efficient Path-Sensitive Analysis

- 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?

MILCOM 2016

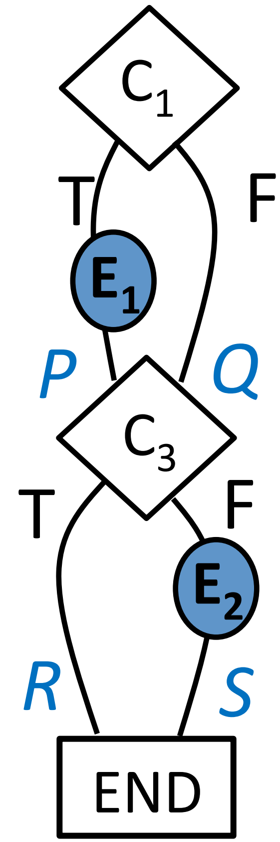
Irrelevant Branch Conditions



C_2 irrelevant to path-sensitive analysis w.r.t. E_1 and E_2

Remove the irrelevant branch conditions to avoid unnecessary path explosion & simplify the path feasibility check.

\# paths reduced from 8 to 4



\# conditions for feasibility check reduced from 3 to 2

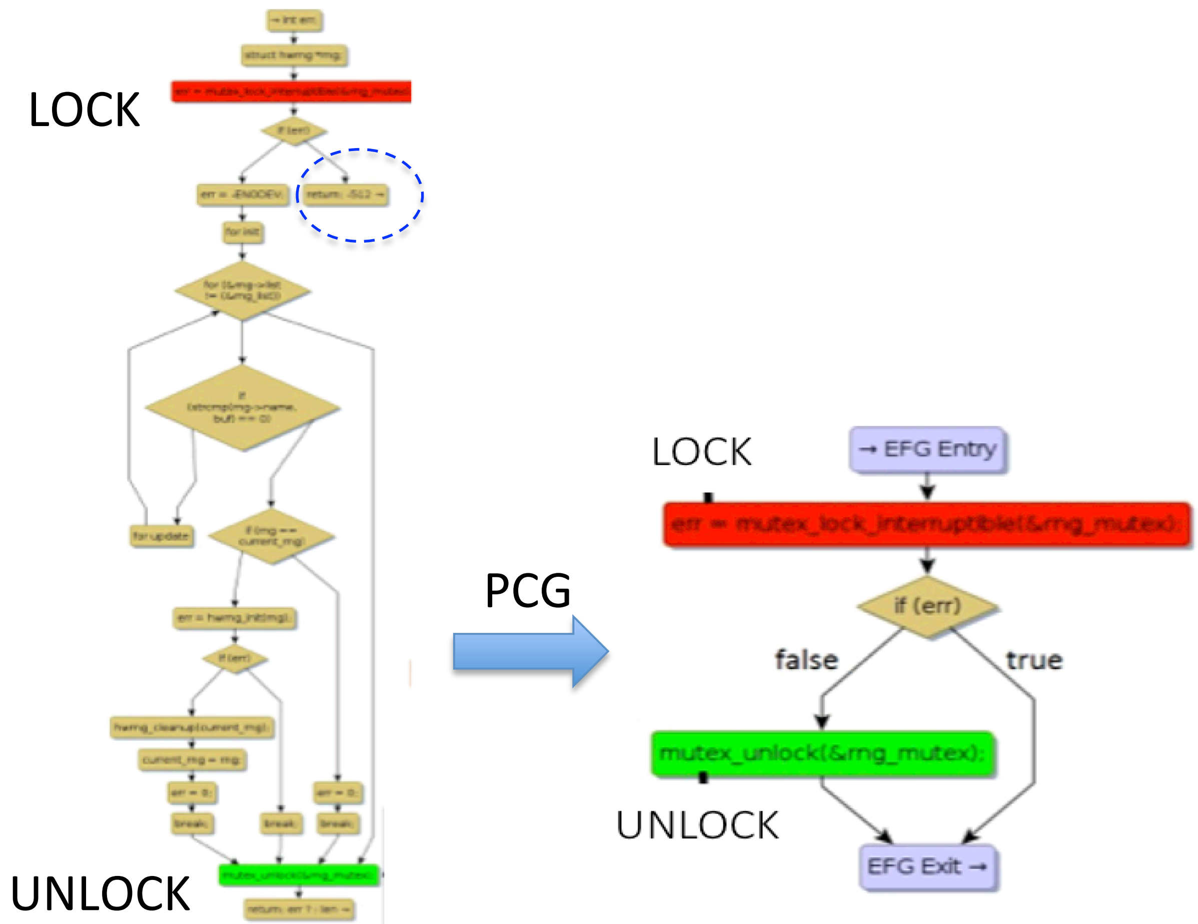
A Mathematical Formulation PCGs

- Basics:
  - A CFG has *operation* and *branch* nodes.
  - The *execution behavior* B(P) along a CFG path P, is a regular expression consisting of the operation nodes along P.
  - A subset of the operation nodes are relevant to a given problem.
  - **The *relevant execution behavior* RB(P) along a CFG path P, is a regular expression consisting of only the relevant operation nodes along P.**
- PCG is a transform of the CFG with the following attributes:
  - It retains all relevant operation nodes.
  - It retains a subset of the branch nodes.
  - It has exactly one path for each distinct ***relevant execution behavior.***

PCGs Minimize Computation

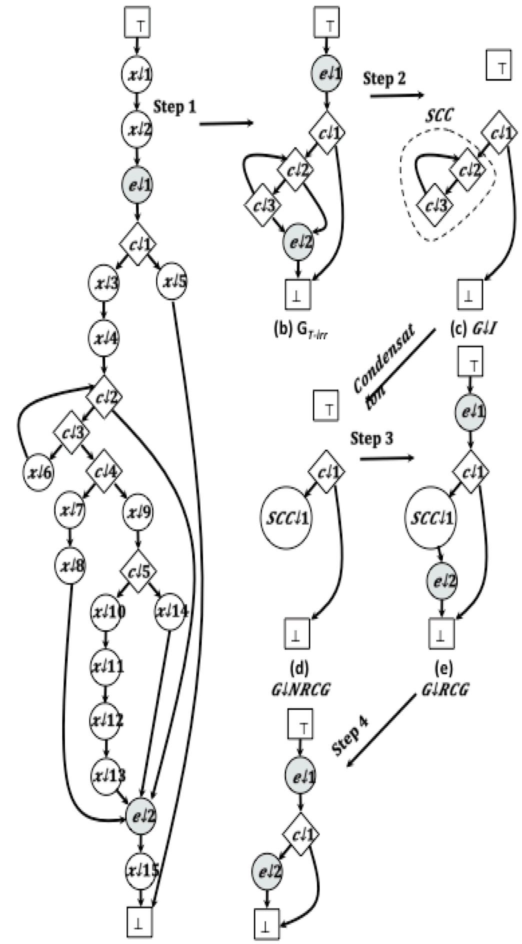
- The CFG has a large number of paths, including paths with loops, but the PCG has only two paths corresponding to distinct behaviors.
- The verifier can maintain the same accuracy but perform less computation by using the PCG instead of the CFG.
- The PCG serves as the evidence and simplifies human reasoning as well.

PCGs Minimize Computation



Transforms to derive the PCGs from a CFG

- The CFG to PCG transformation involves a sequence of three basic transformations and a condensation transformation.
- The relevant operation nodes govern the transform.
- Before the transform, the relevant operation nodes are identified w.r.t. a given problem.
- The transform uses a well-known graph algorithm published in the late 70's.
- It is a linear-time algorithm.



The Applicability of PCGs

PCG is a powerful abstraction for a software symmetry with many applications.

- Any application where a problem can be abstracted using the notion of relevant execution behaviors.
- We use it for verifying the Linux kernel for important safety properties.
- Our new generation of verifiers have achieved unprecedented accuracy and scalability using the PCG and other abstractions based on the software symmetries of the Linux kernel.
- In the DARPA STAC project, we use PCGs to reason about *side channel* (SC) and *algorithmic complexity* (AC) vulnerabilities.