An Improved Distance Heuristic Function for Directed Software Model Checking

Neha Rungta and Eric G. Mercer

Software Model Checking Lab

Computer Science Department

Brigham Young University

Provo, UT - 84602

Motivation

- Use of embedded systems has become ubiquitous
- Growing complexity challenges ad-hoc testing methods
- Vector simulation finds bugs in the early design phase
- Code coverage techniques are not feasible
- Low-level scheduling decisions create concurrency errors
- Motivates a need for a formal approach to find these errors

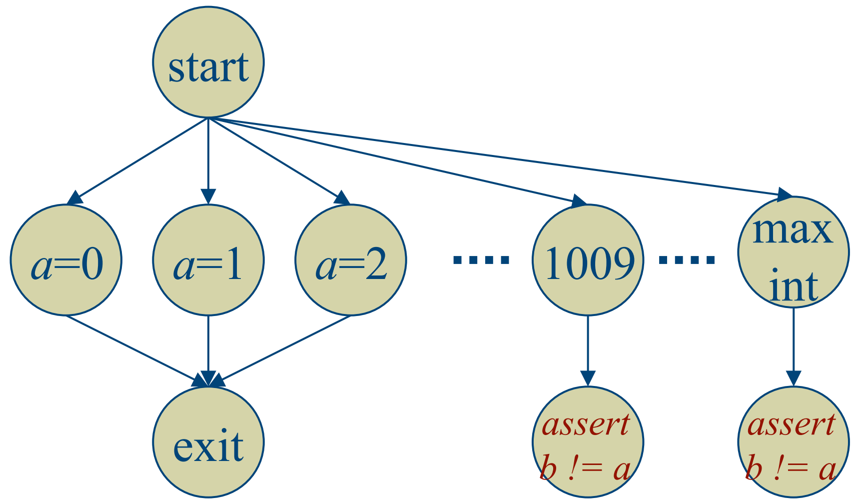
Software Model Checking

```
int b = 1009
void simple(){
   int a;
   read a;
   if (a > 1000)
      assert(b != a)
}
```

- It builds a model for a given software system

Simple program

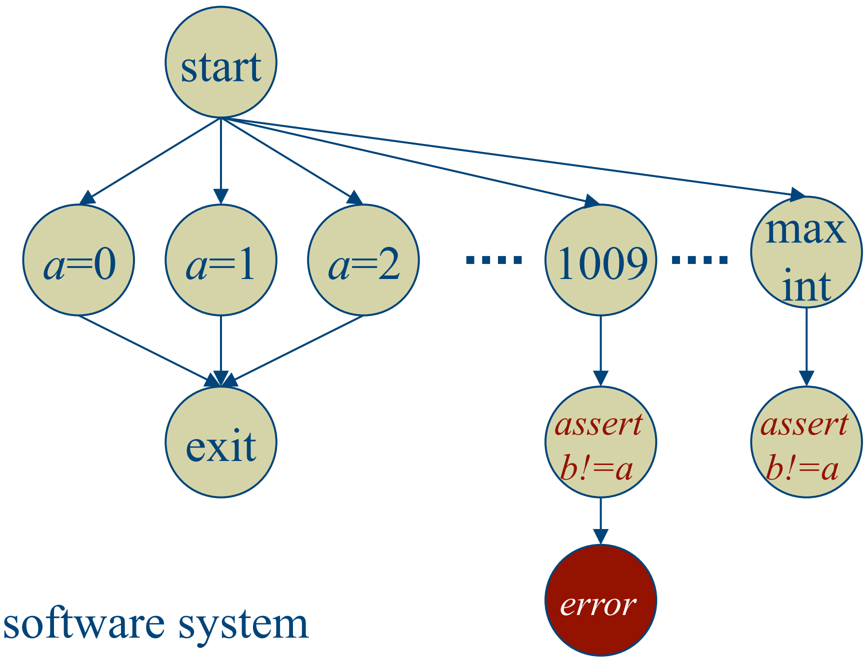
```
int b = 1009
void simple(){
   int a;
   read a;
   if (a > 1000)
      assert(b != a)
}
```



- It builds a model for a given software system
- The transition graph represents all the behaviors of the system

Simple program

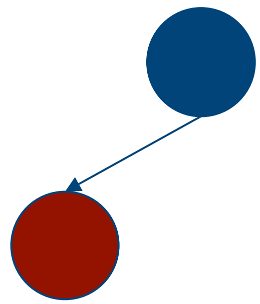
```
int b = 1009
void simple(){
    int a;
    read a;
    if (a > 1000)
        assert(b != a)
}
```



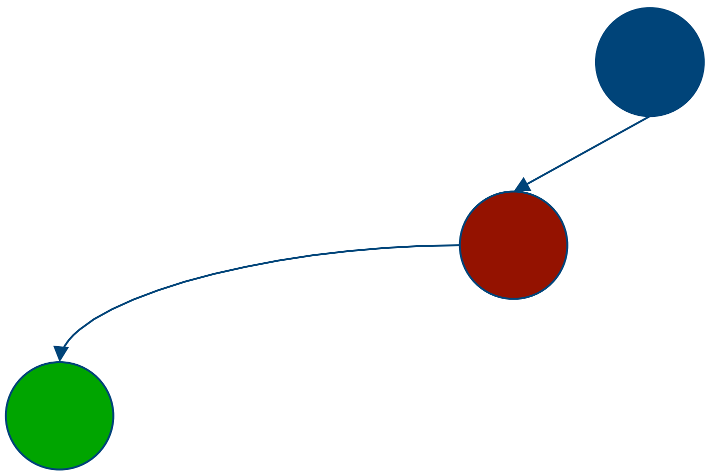
- It builds a model for a given software system
- The transition graph represents all the behaviors of the system
- The property being verified is whether there exists a path to the error

Exhaustive Search - DFS

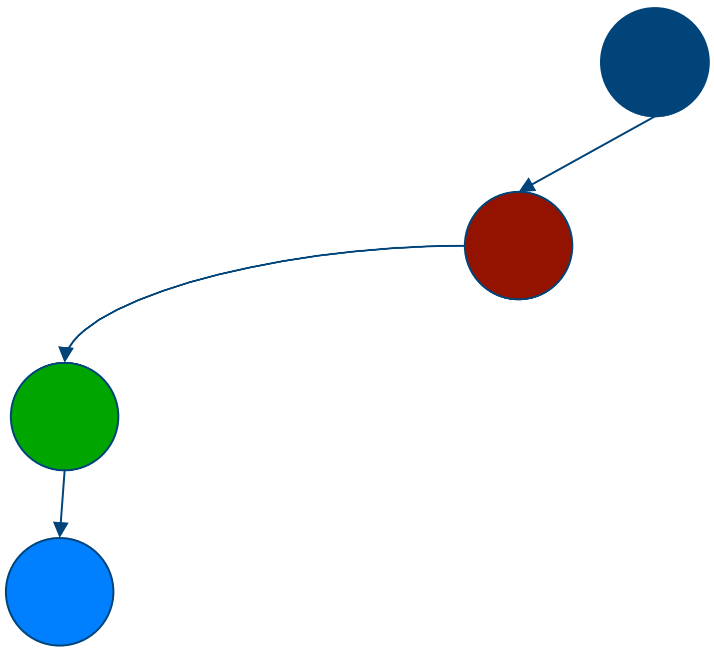
Exhaustive Search - DFS



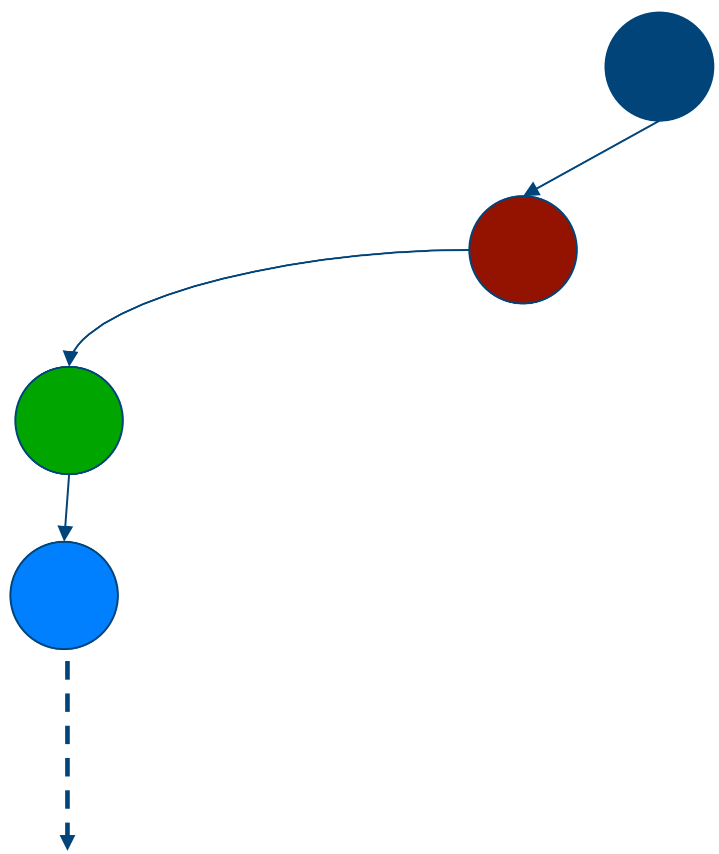
Exhaustive Search - DFS



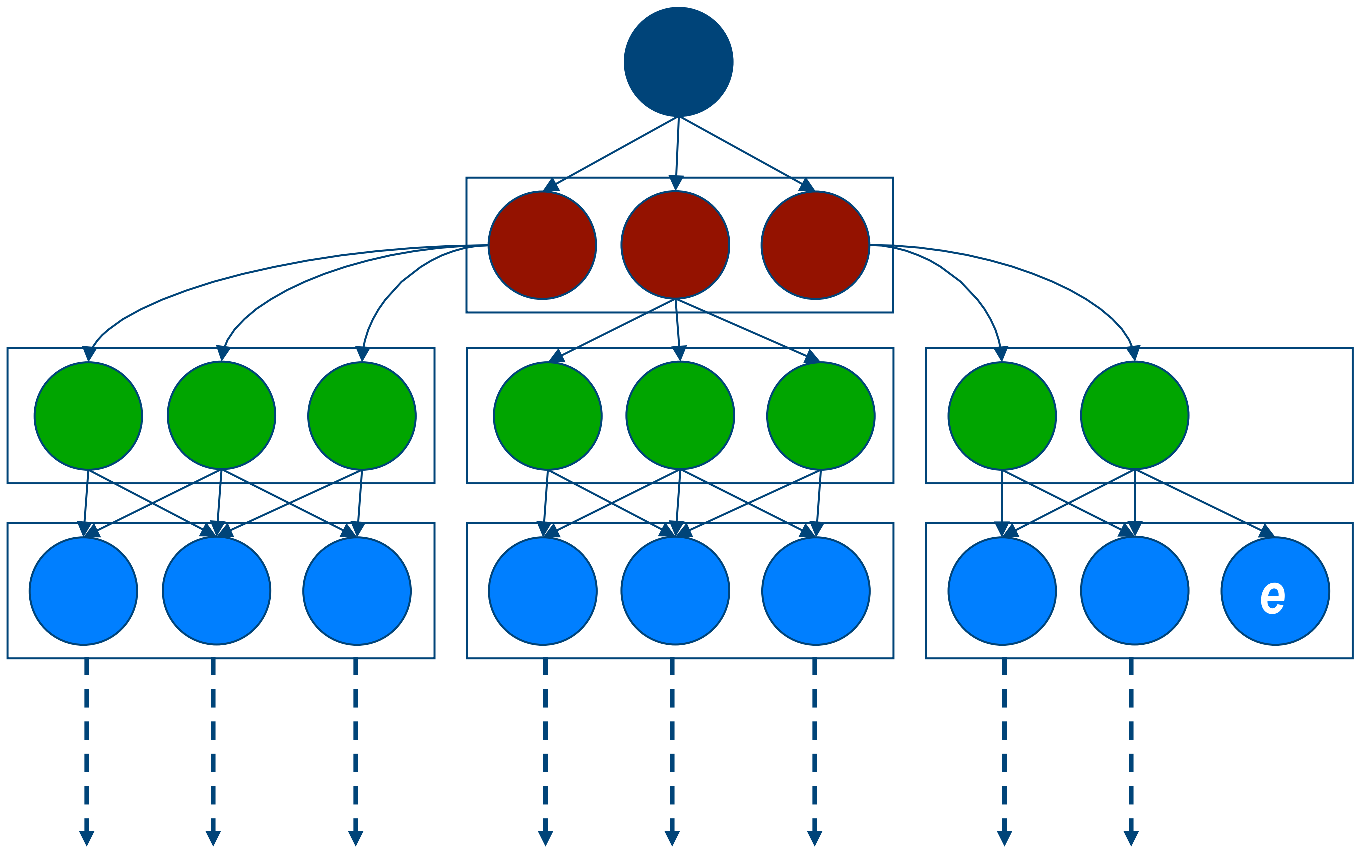
Exhaustive Search - DFS



Exhaustive Search - DFS

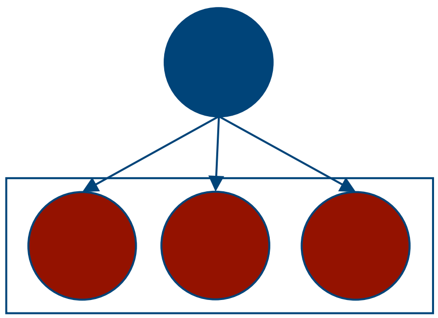


Exhaustive Search - DFS

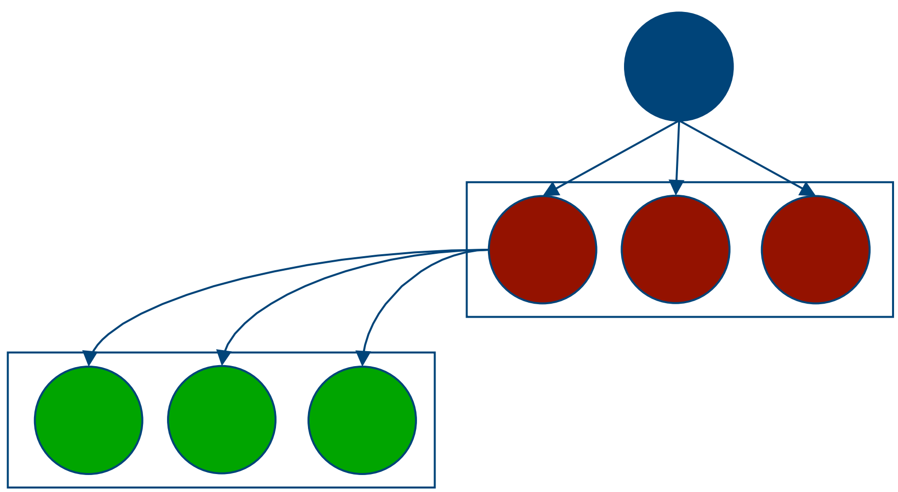


Exhaustive Search - BFS

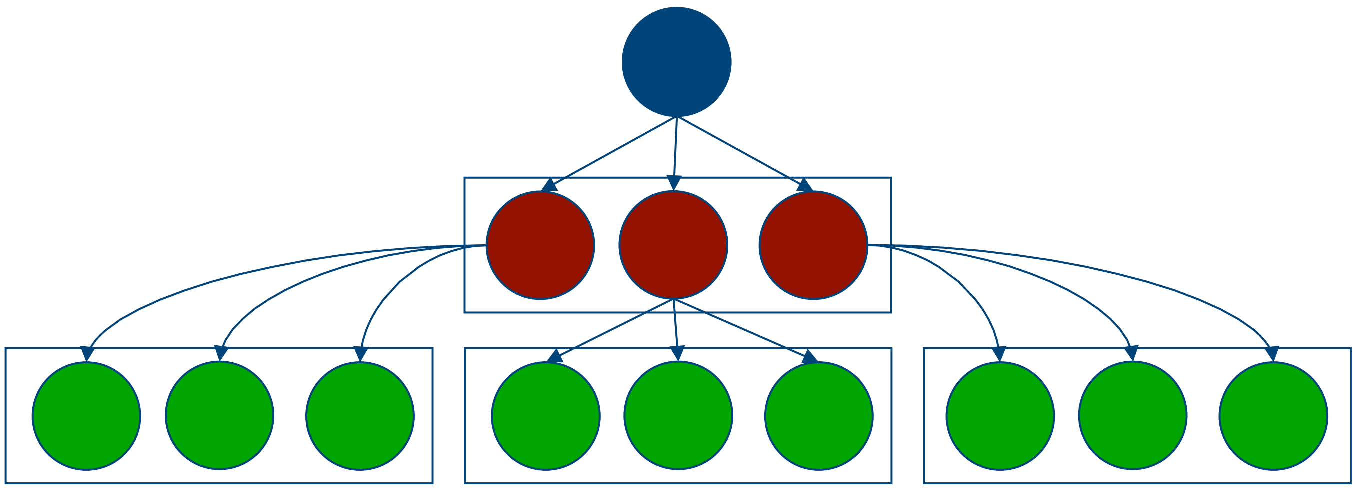
Exhaustive Search - BFS



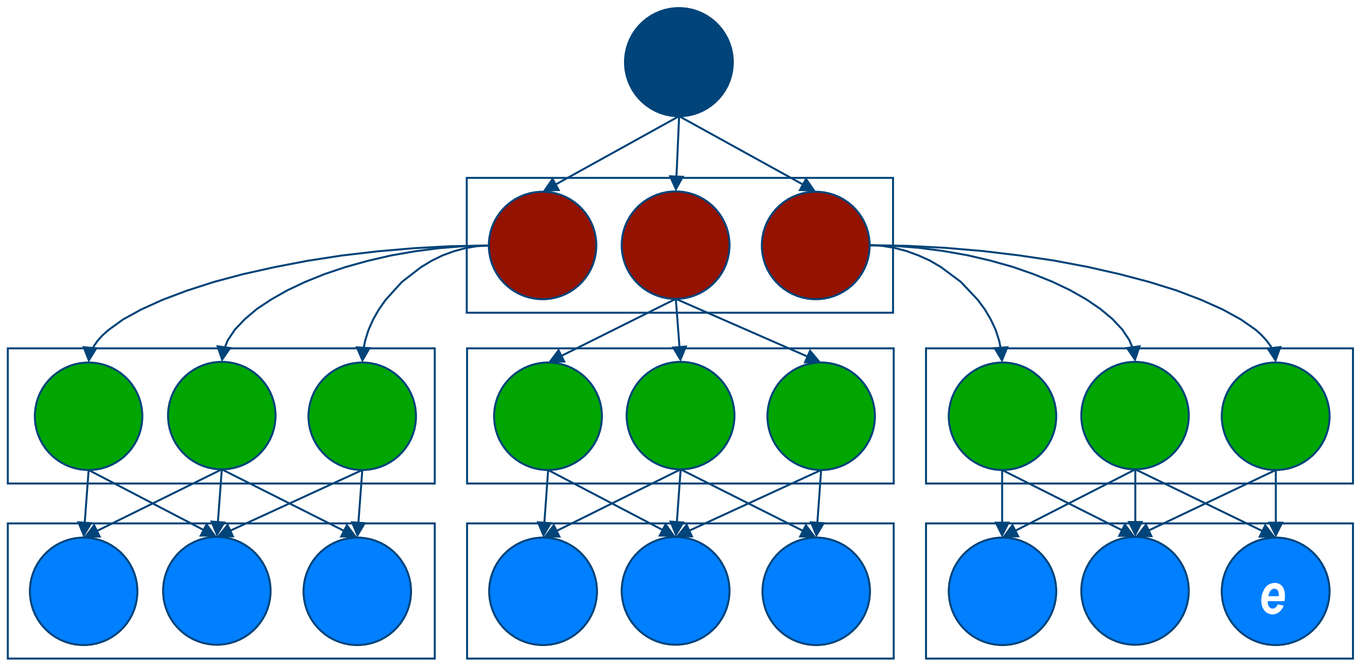
Exhaustive Search - BFS



Exhaustive Search - BFS

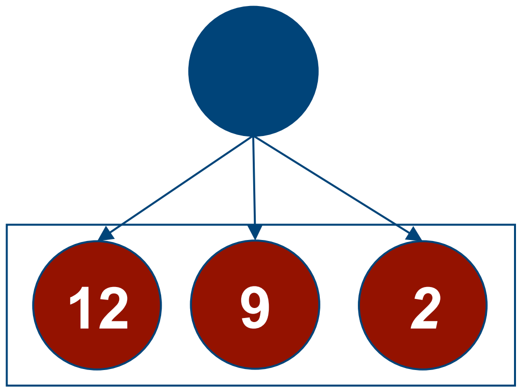


Exhaustive Search - BFS

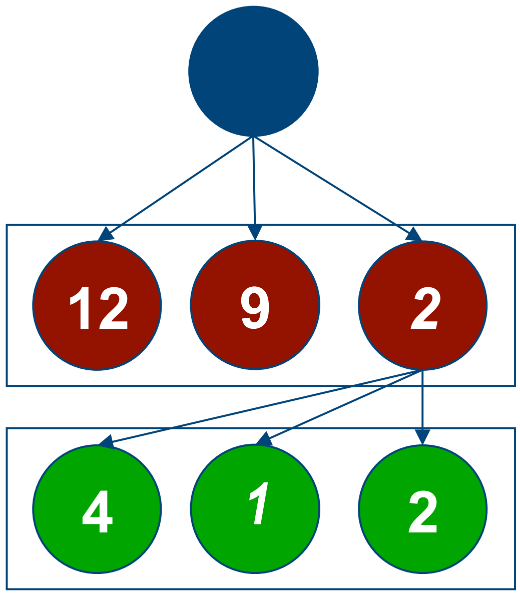


Guided Best-first Search

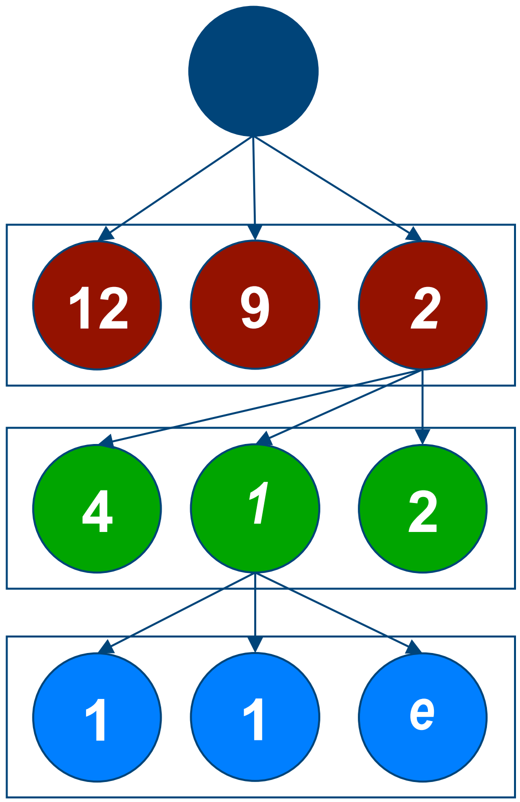
Guided Best-first Search



Guided Best-first Search



Guided Best-first Search



Related Work on Heuristic estimates

- *Edelkamp, Lafuente, and Leue*
- Minimum number of changes in program values

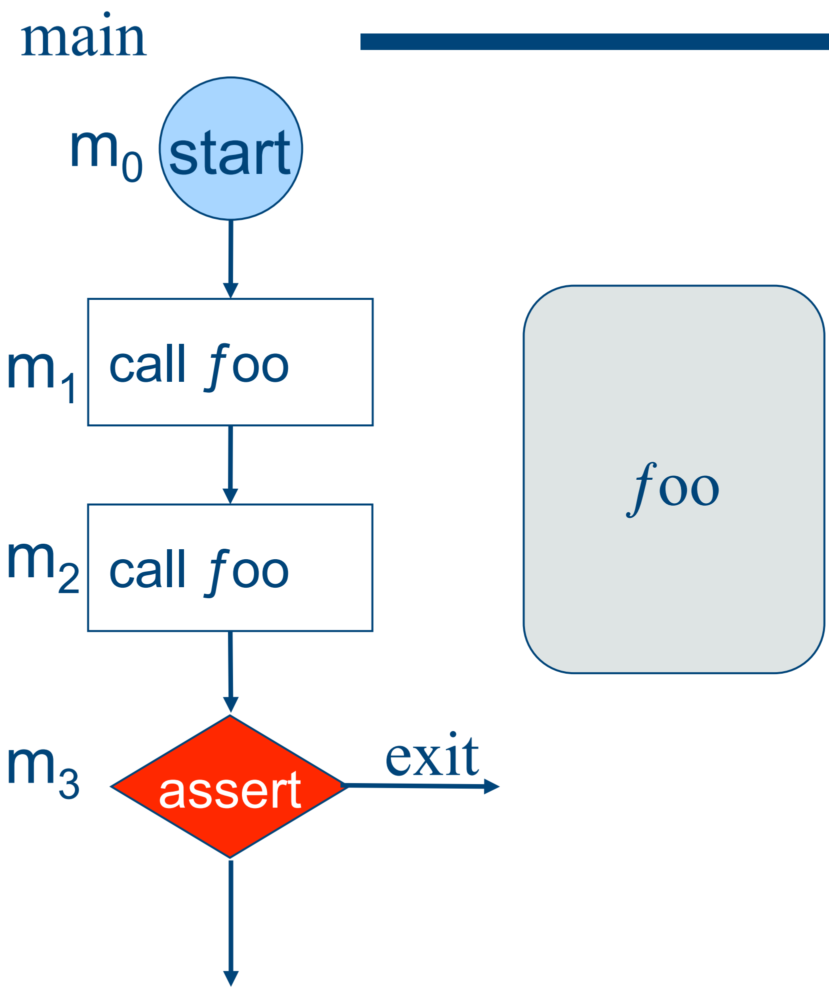
- *Seppi, Jones, and Lamborn*
- Use Bayesian reasoning

- *Visser and Groce*
- Structural properties of thread interdependencies

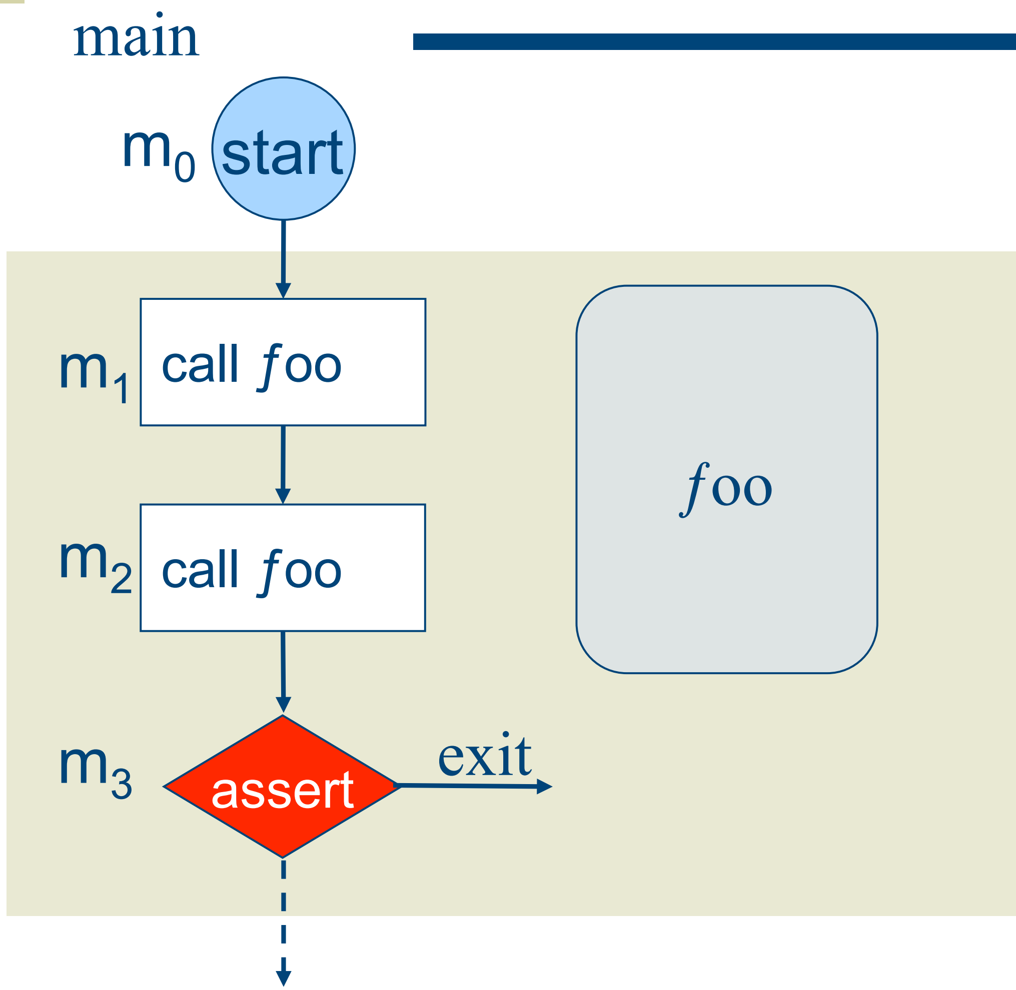
- *Edelkamp and Mehler*
- Minimal number of transitions (FSM distance)

- *Rungta and Mercer*
- Use partial context information to improve FSM distance

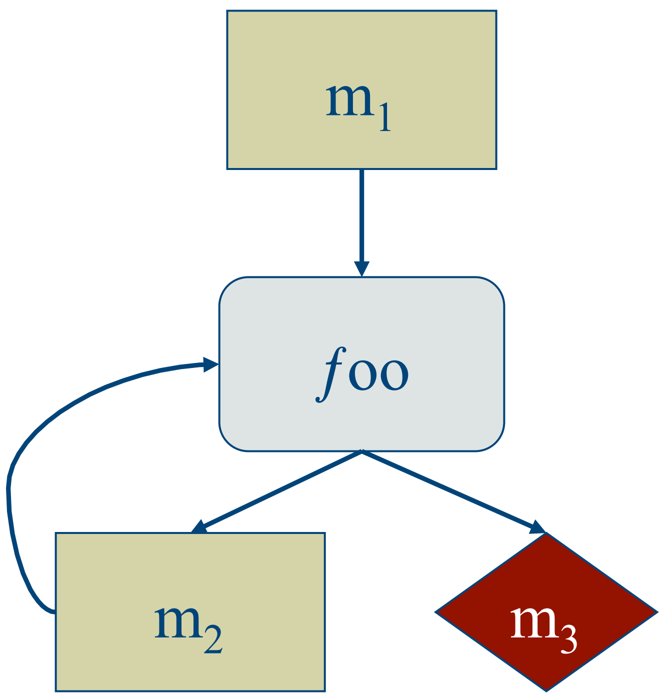
FSM Distance Heuristic

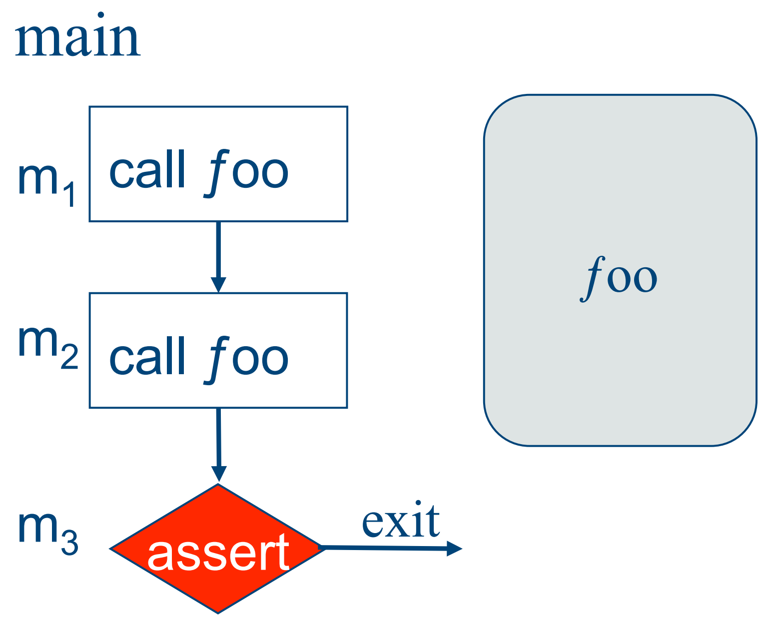


FSM Distance Heuristic

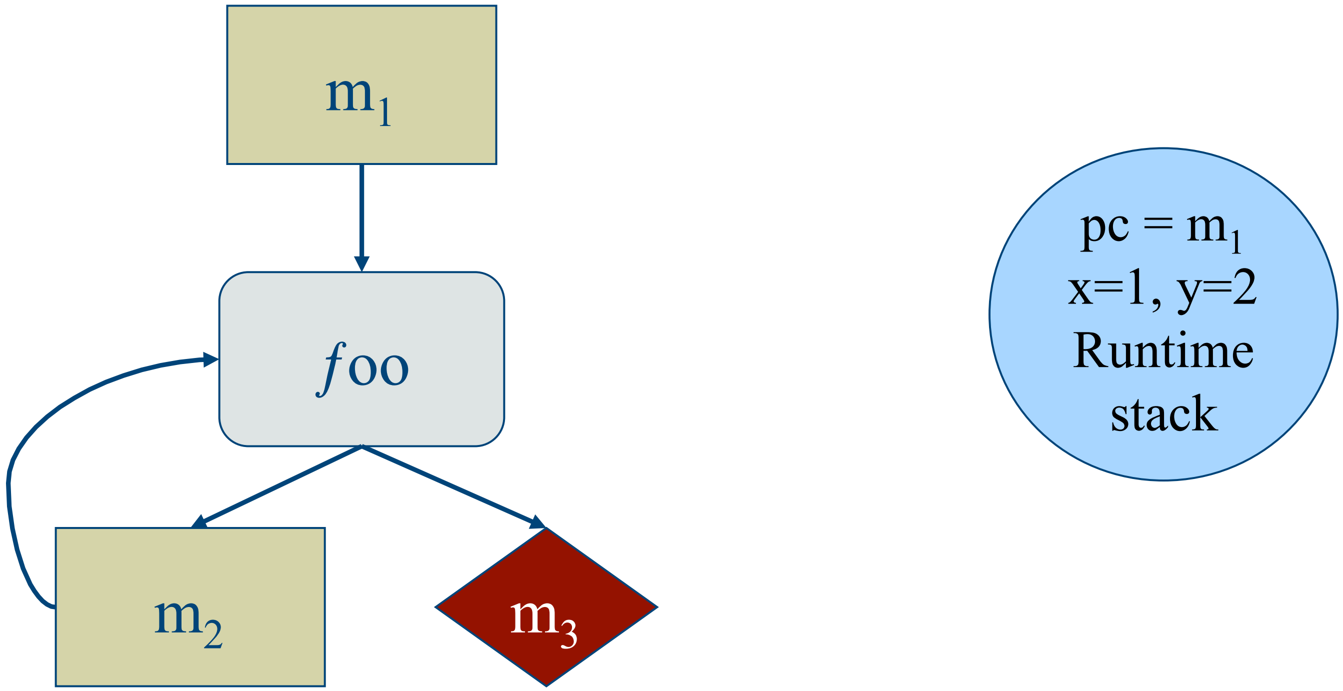


FSM Distance Heuristic

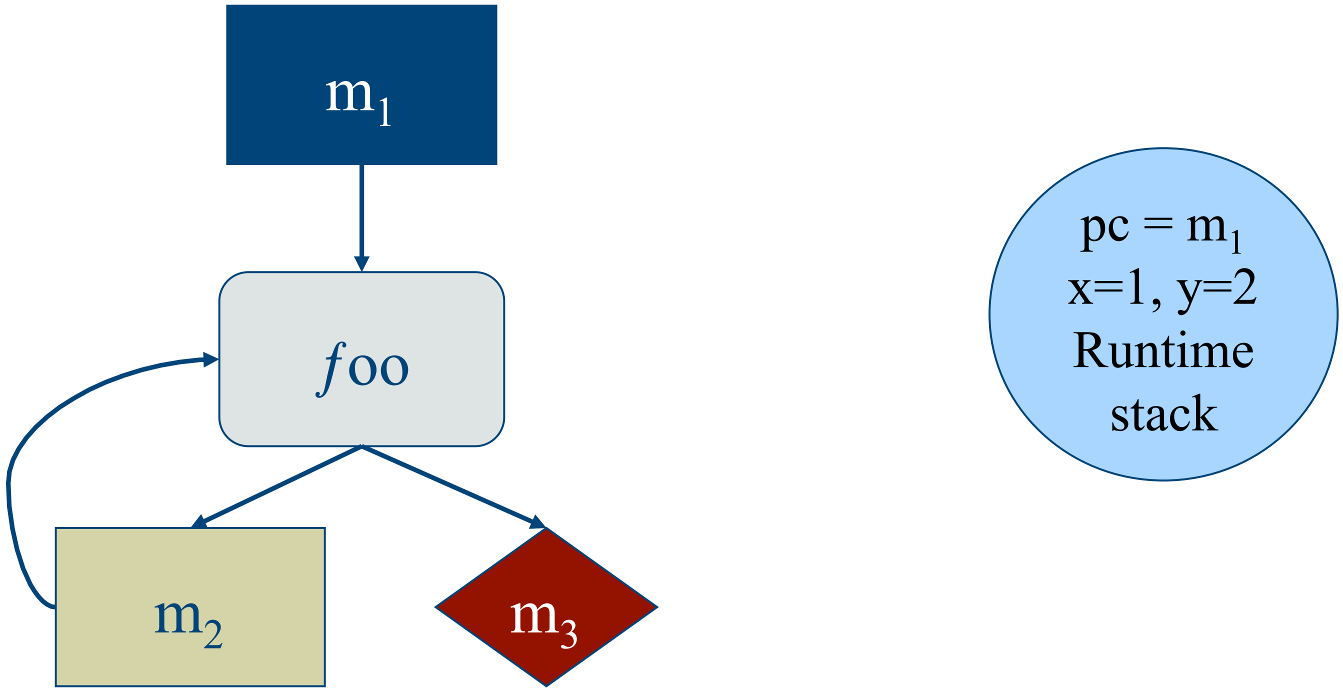




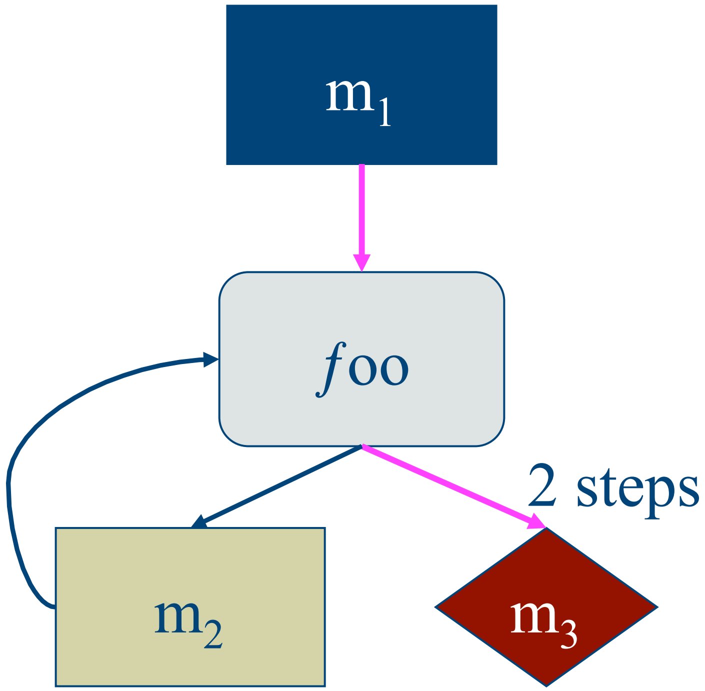
FSM Distance Heuristic



FSM Distance Heuristic

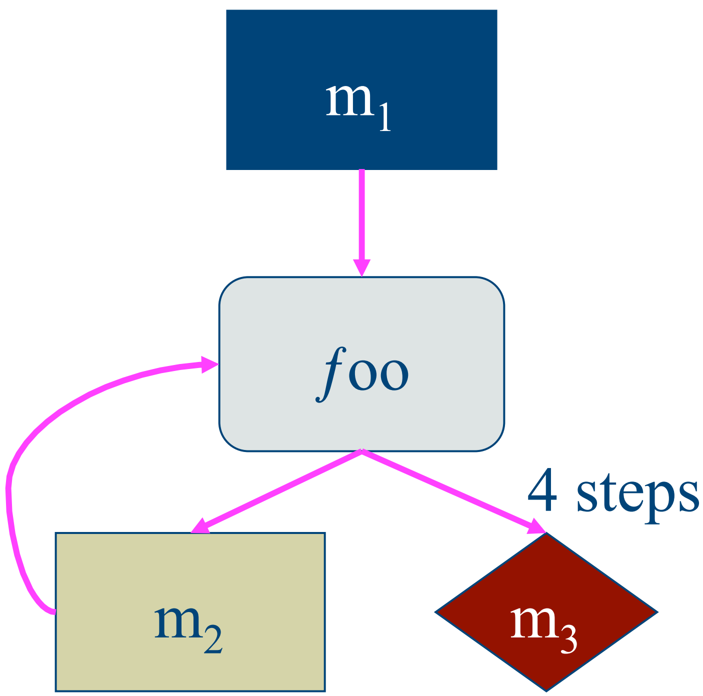


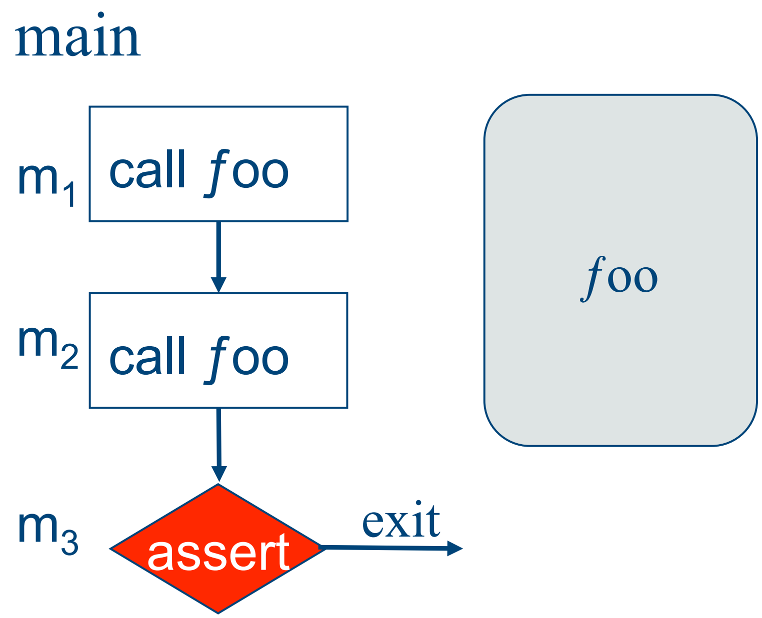
FSM Distance Heuristic



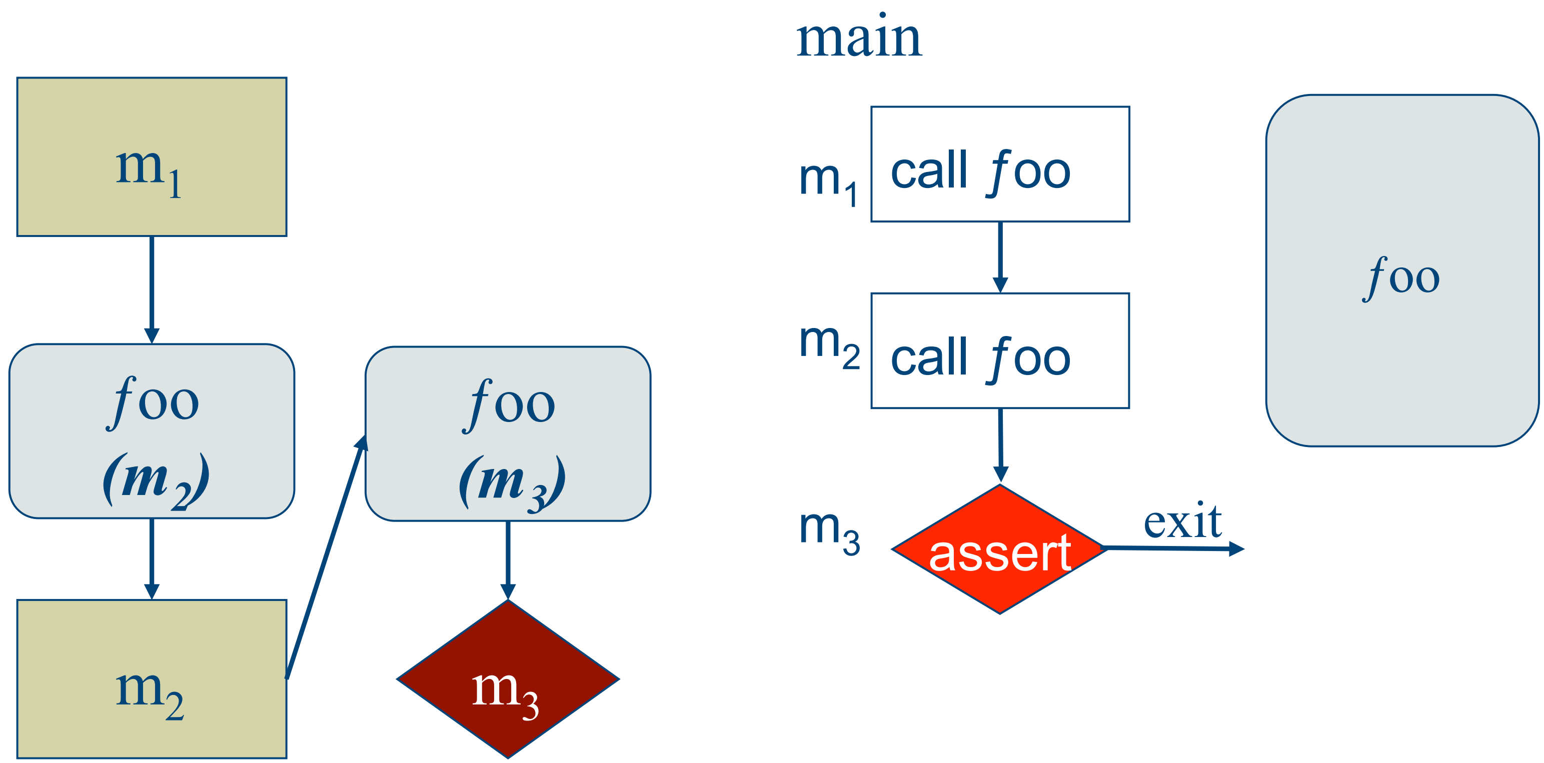
pc = m_1
x=1, y=2
Runtime
stack

Lack of Context

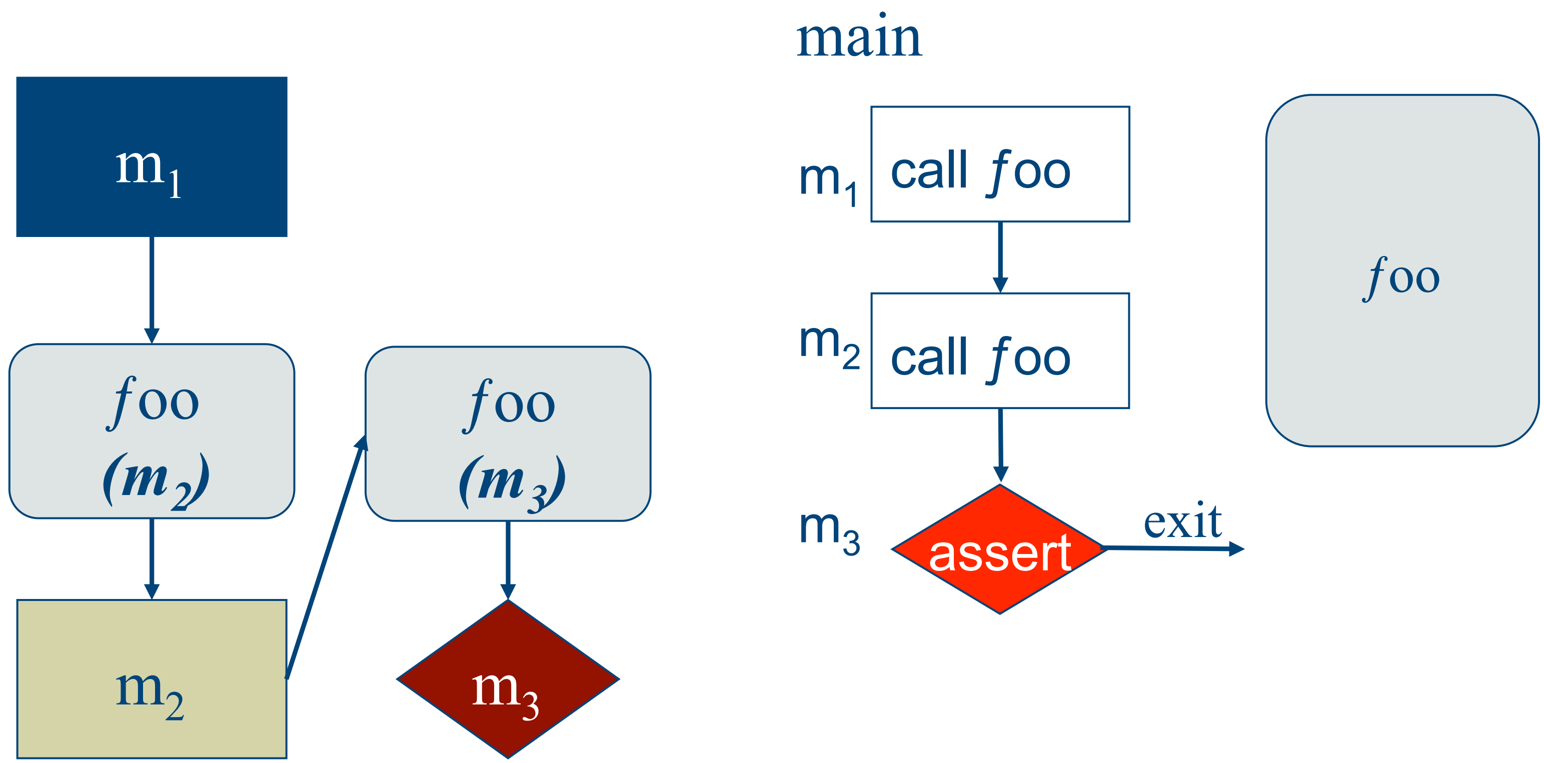




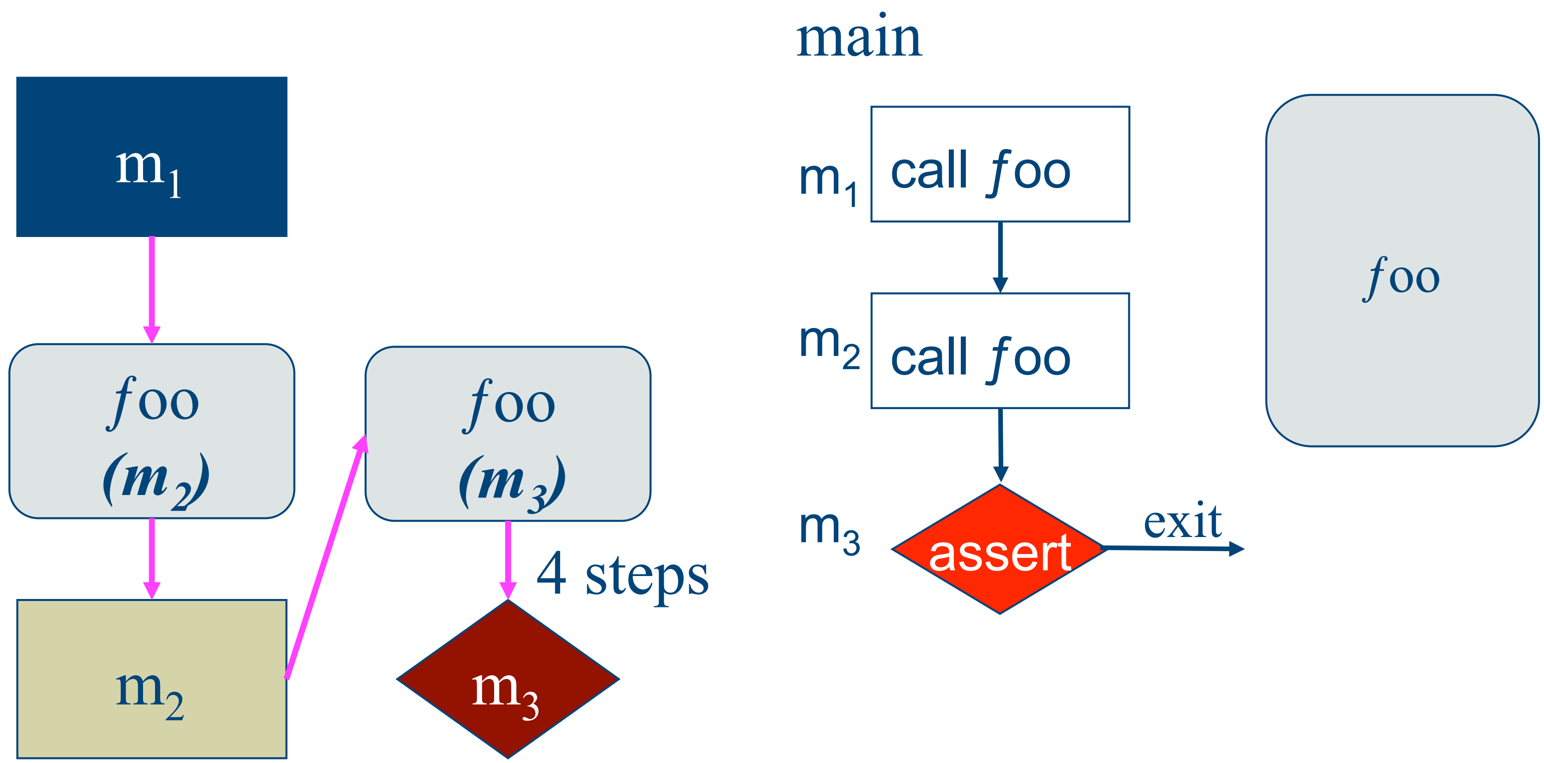
k-bounded Graph



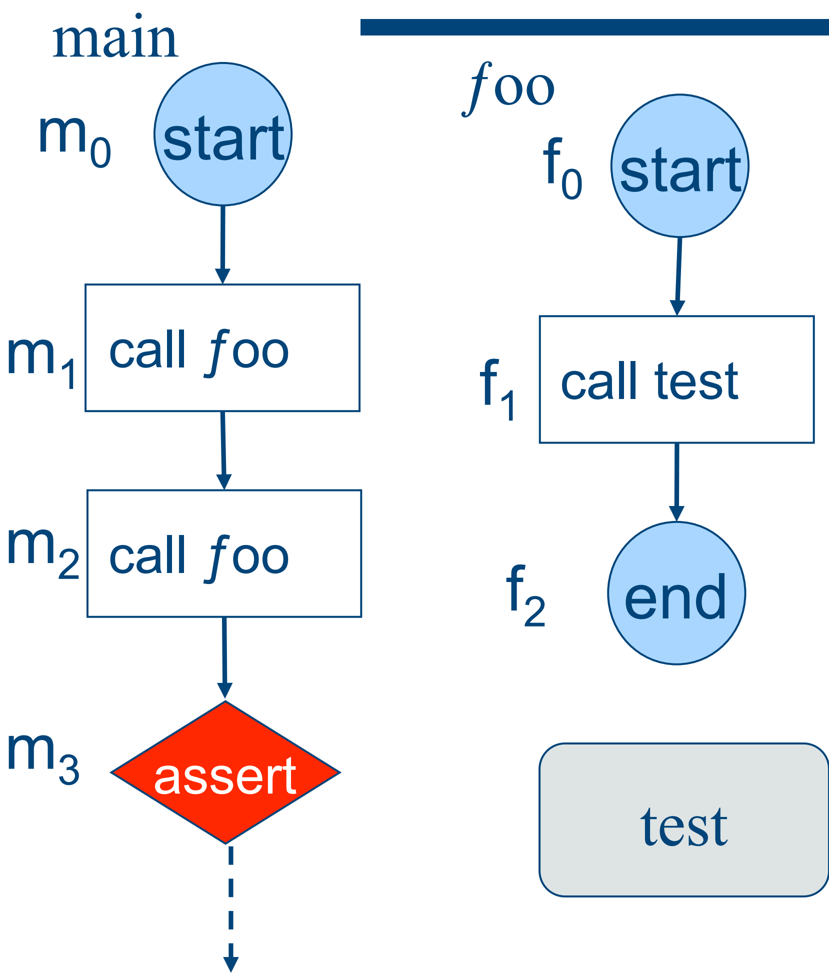
k-bounded Graph



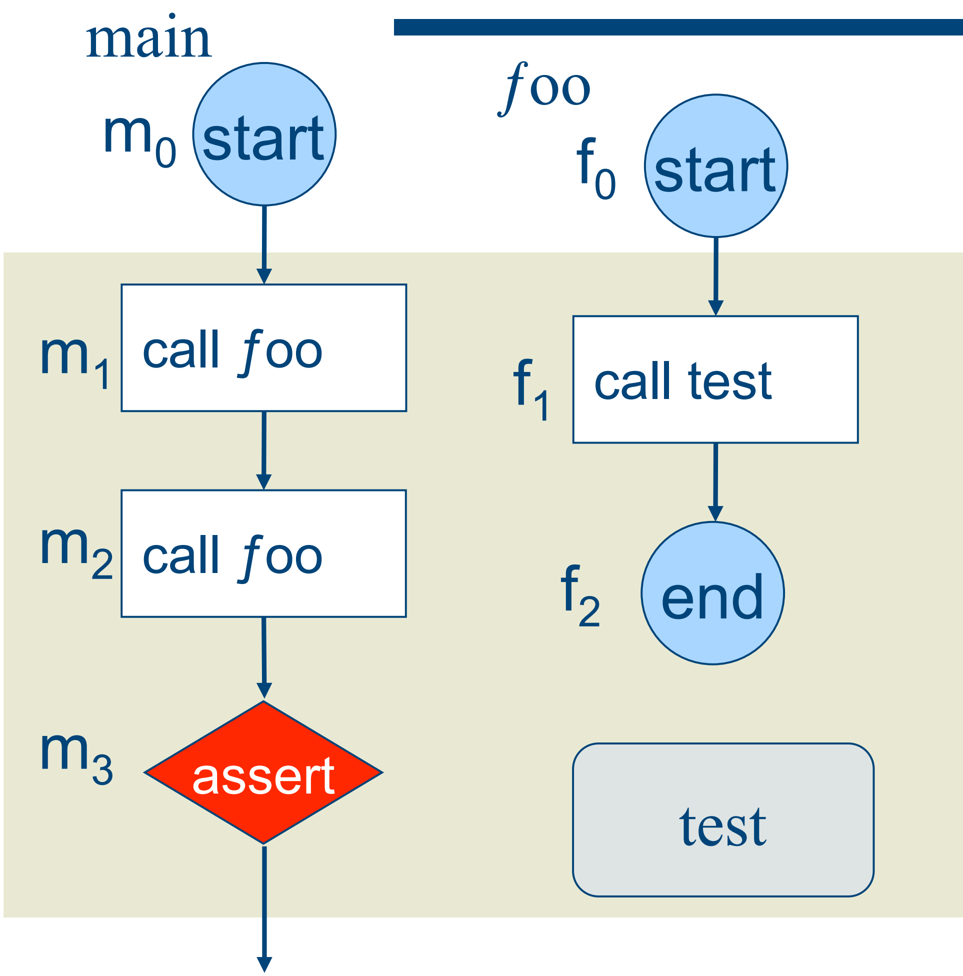
k-bounded Graph



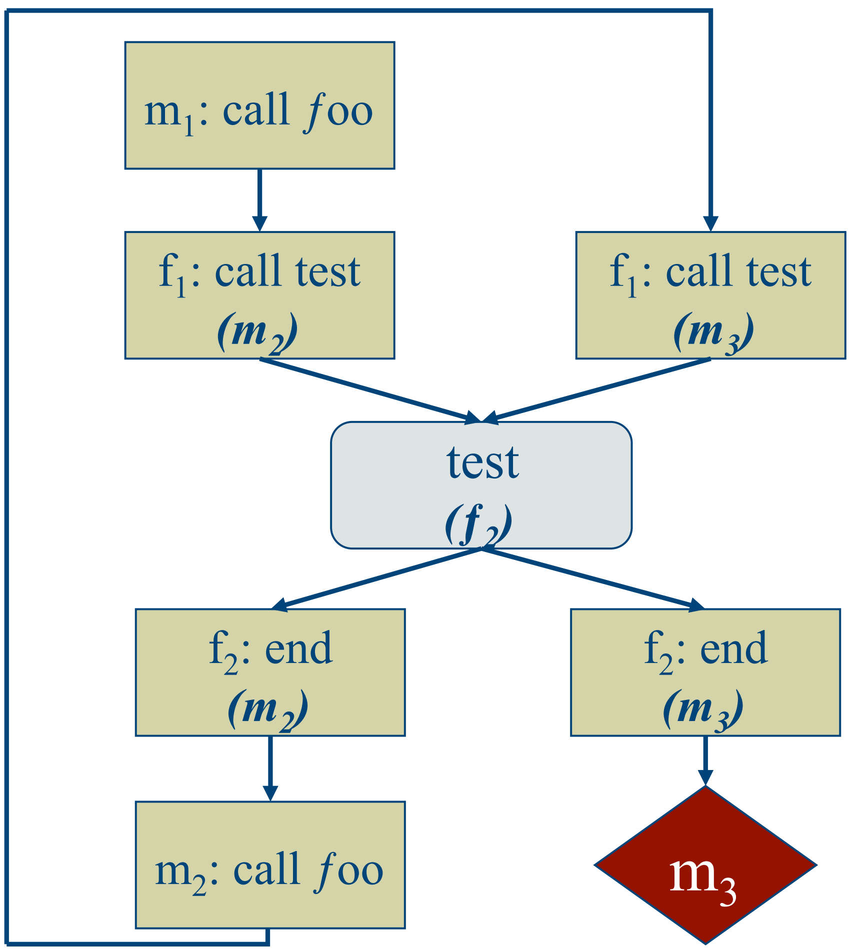
EFSM Distance Heuristic

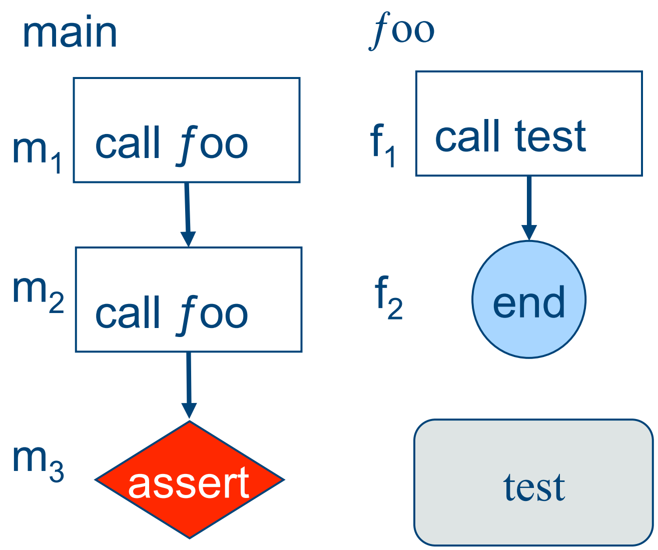


EFSM Distance Heuristic

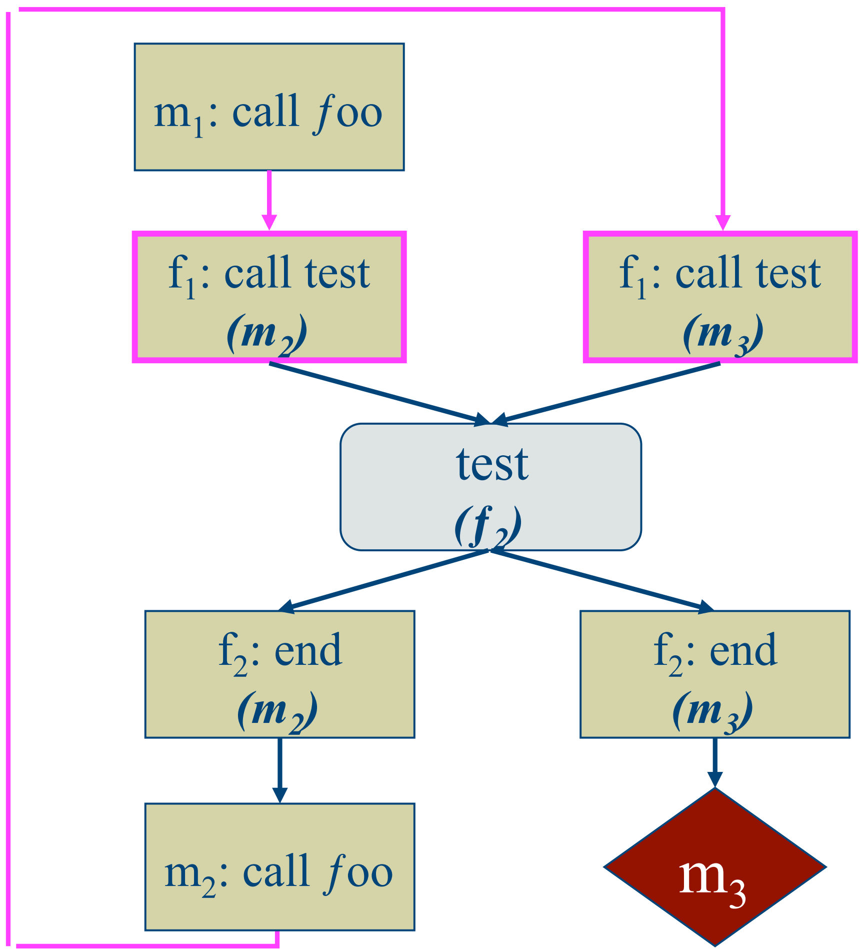


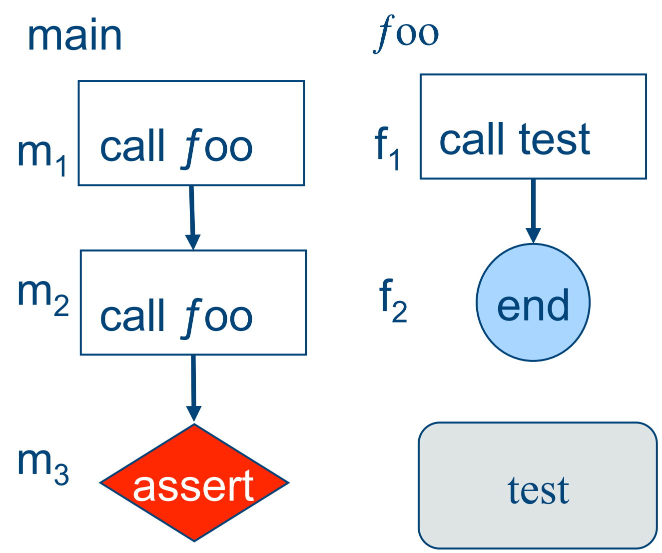
EFSM Distance Heuristic



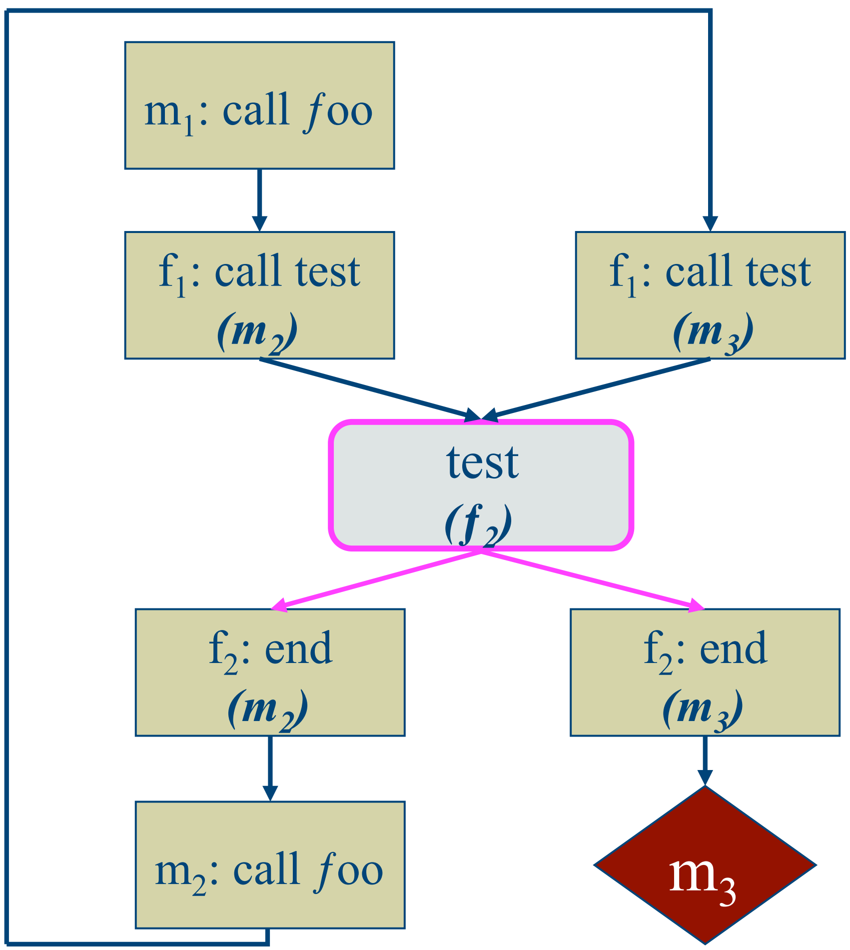


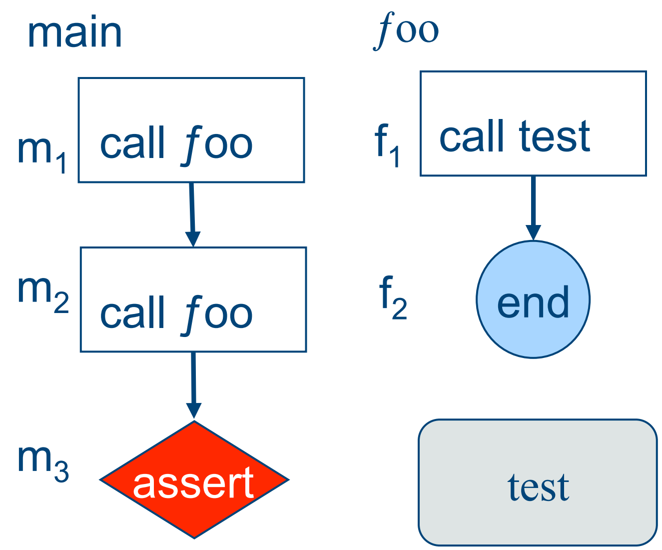
EFSM Distance Heuristic





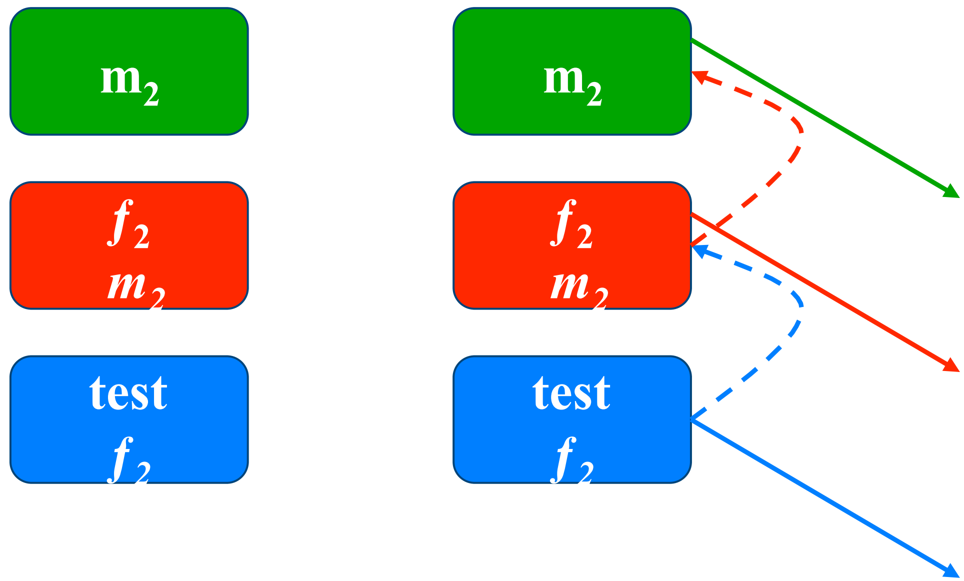
EFSM Distance Heuristic



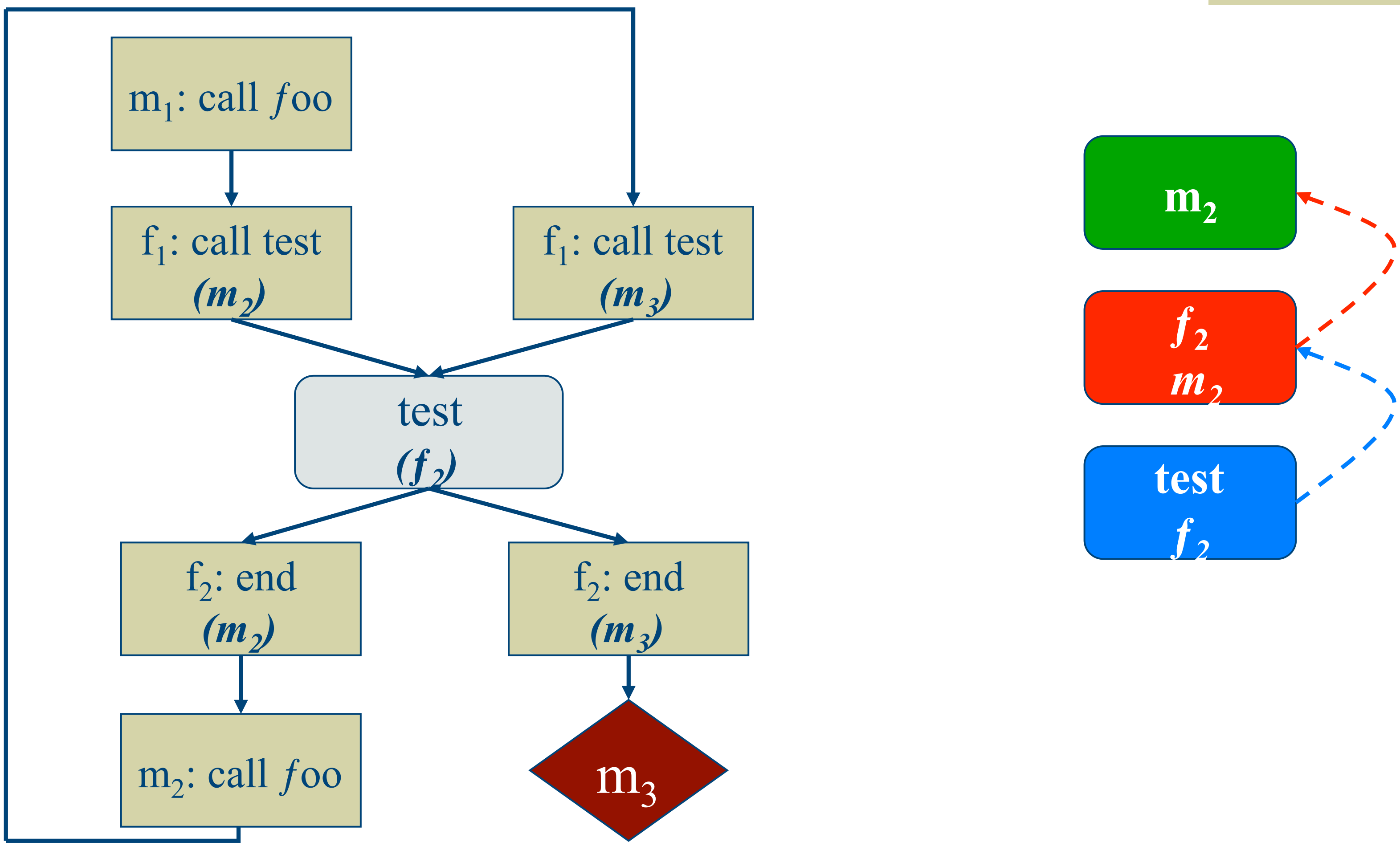


Recreate the call trace

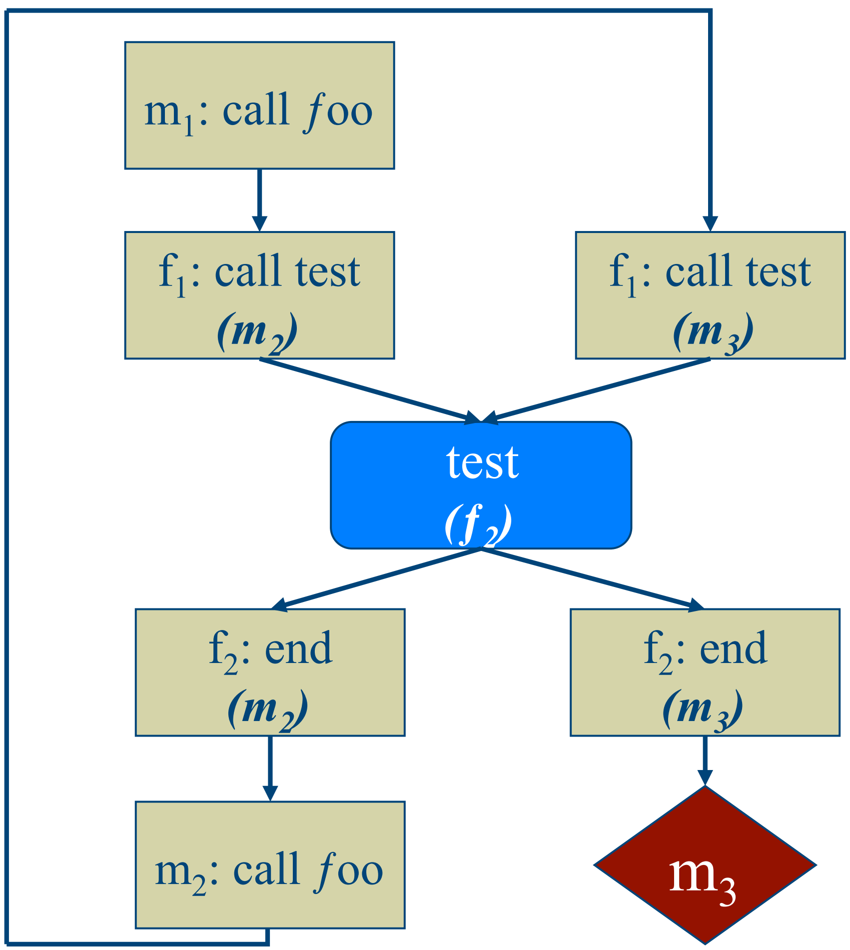
Stack frame for main
Return Add : m_2
Stack frame for *foo*
Return Add : f_2
Stack frame for test

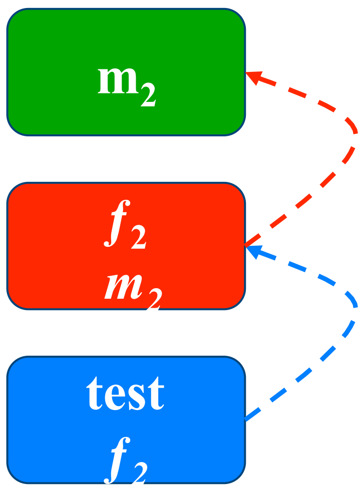


EFSM Distance Heuristic

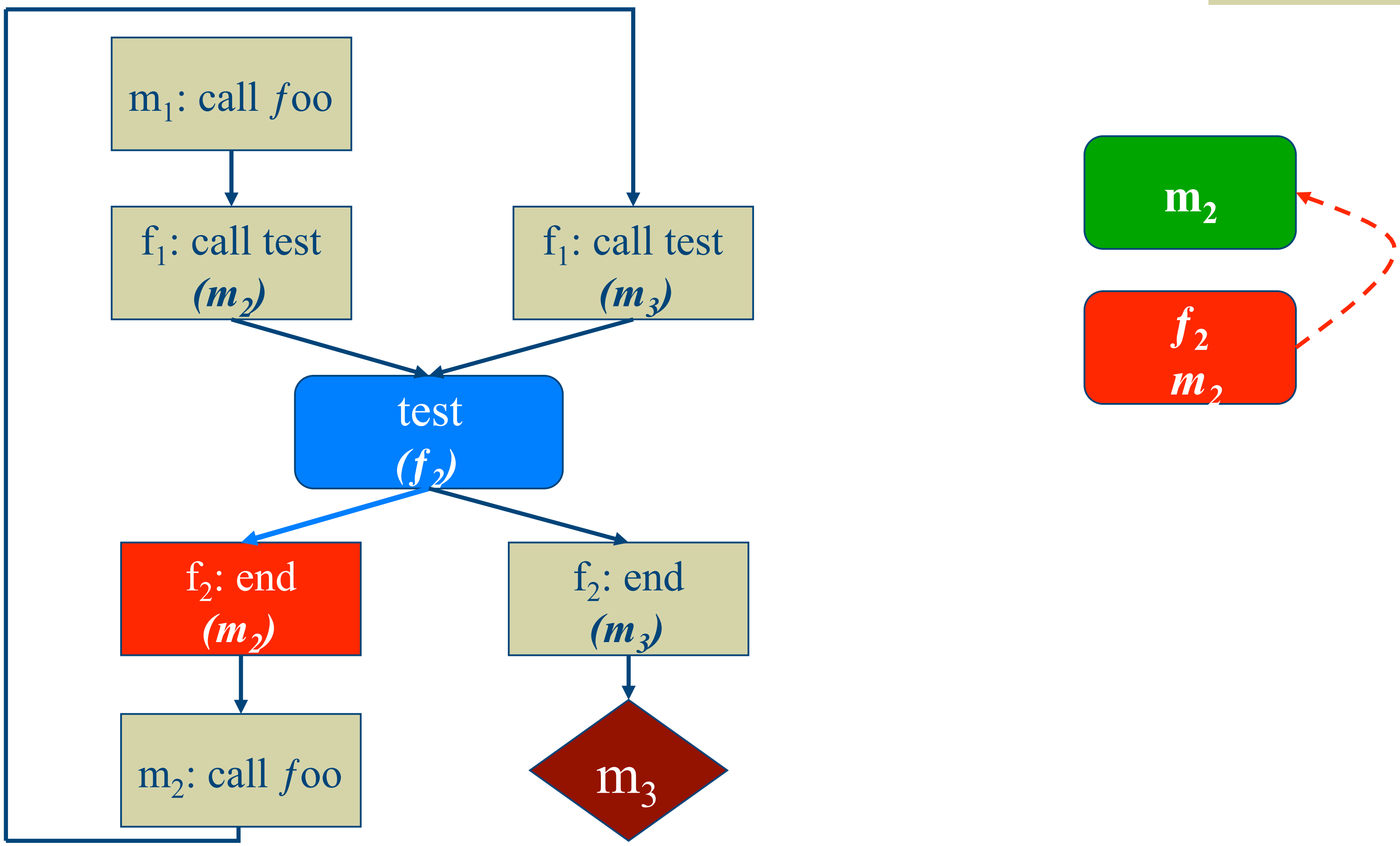


EFSM Distance Heuristic

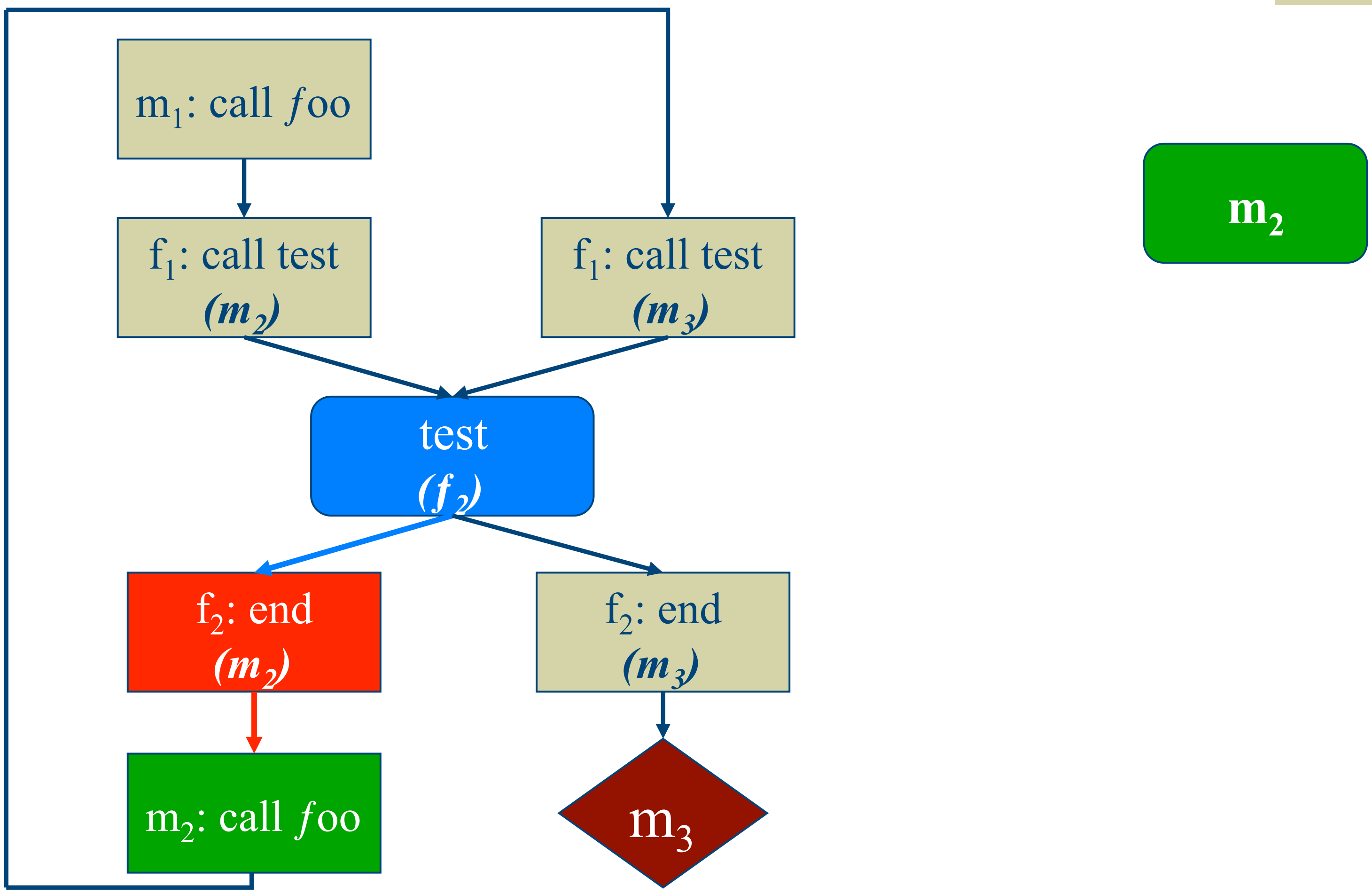




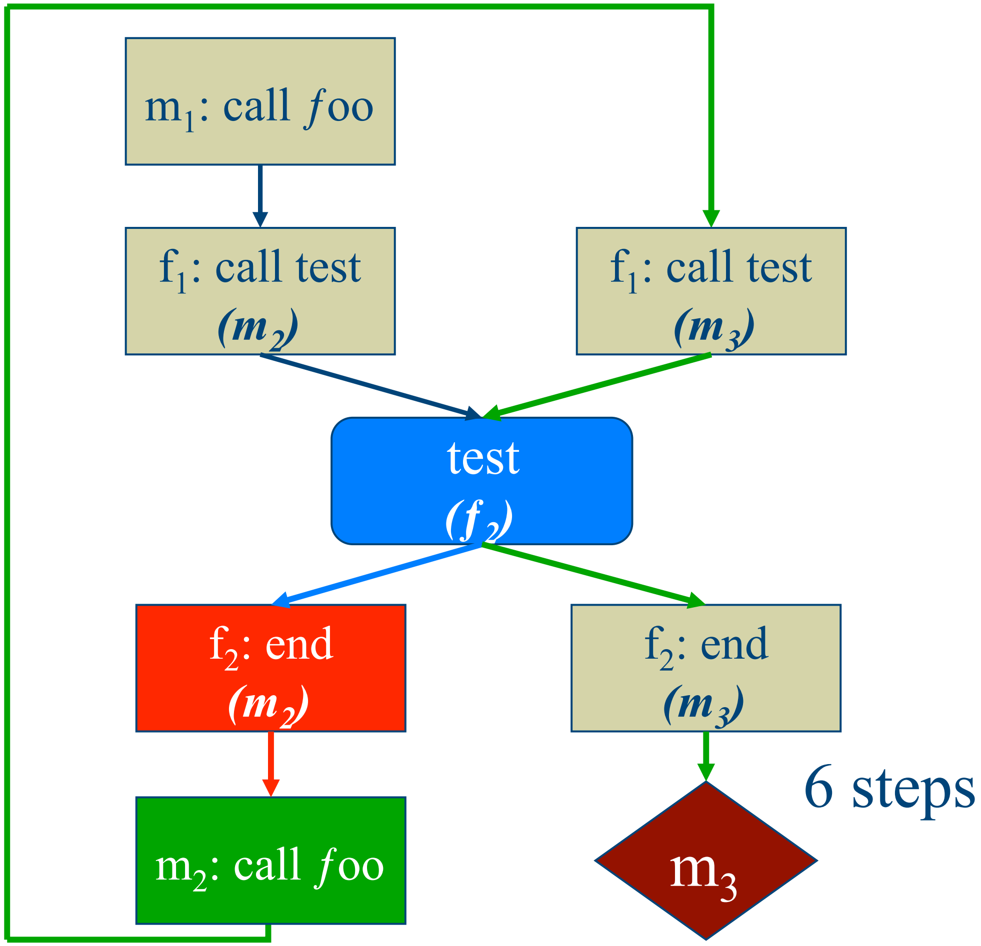
EFSM Distance Heuristic



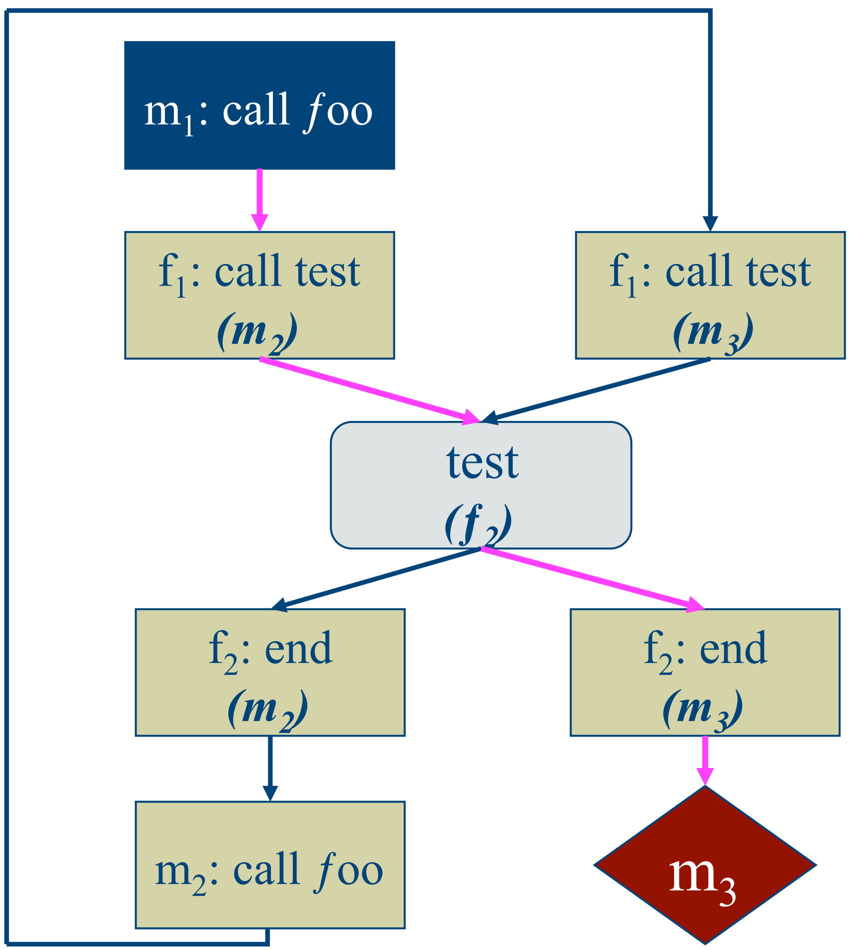
EFSM Distance Heuristic

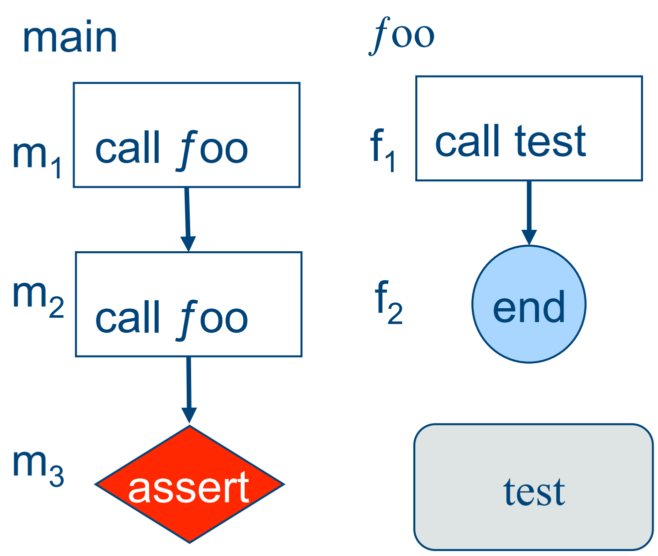


EFSM Distance Heuristic the Heuristic



Forward estimates are inaccurate

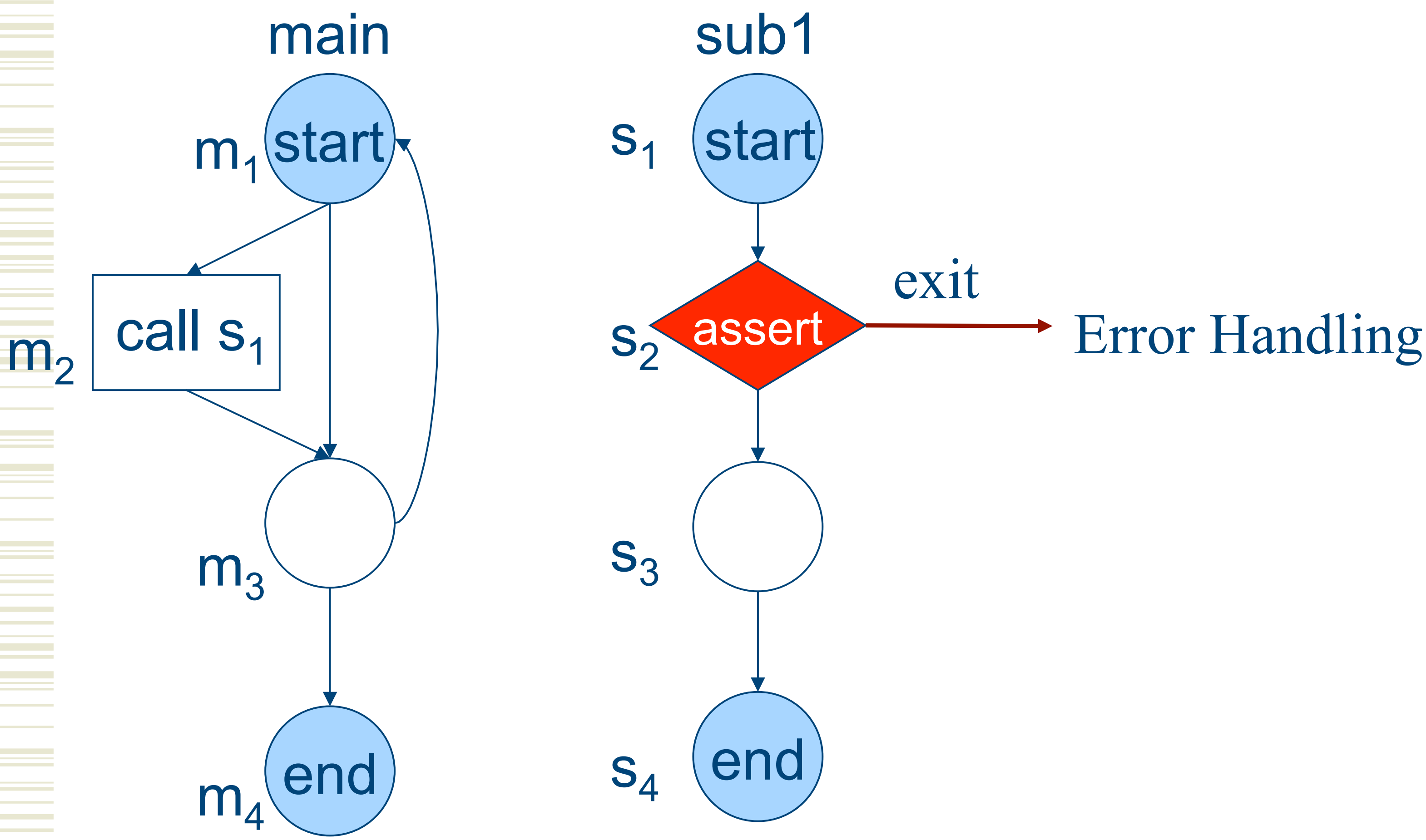




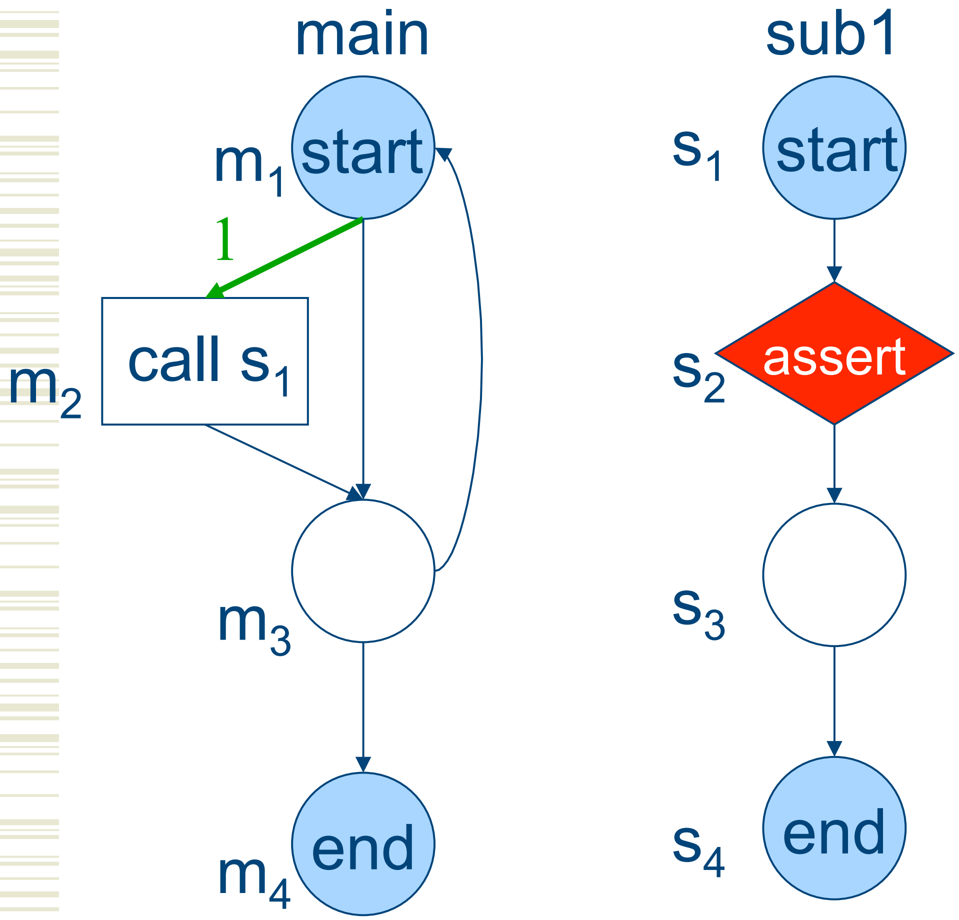
Full Context Aware (FCA)

- Assume no recursion and resolved call-sites
- Statically compute distance estimates
- Full context information in the forward direction

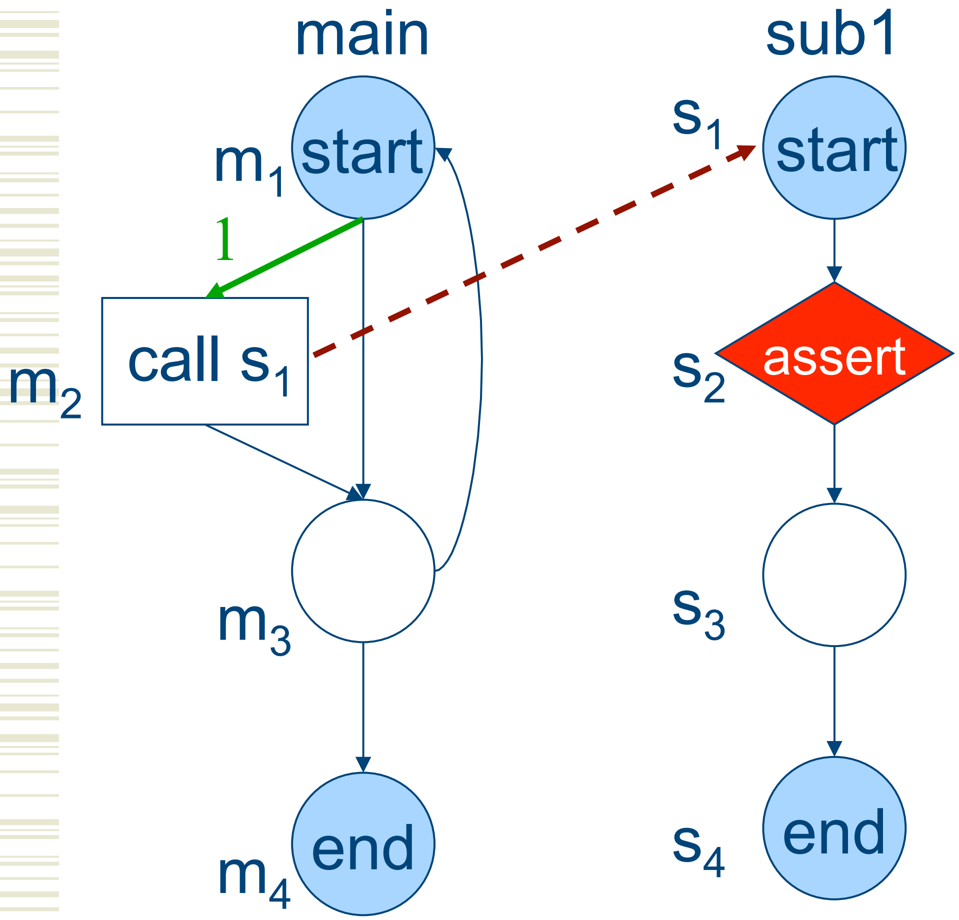
Full Context Example



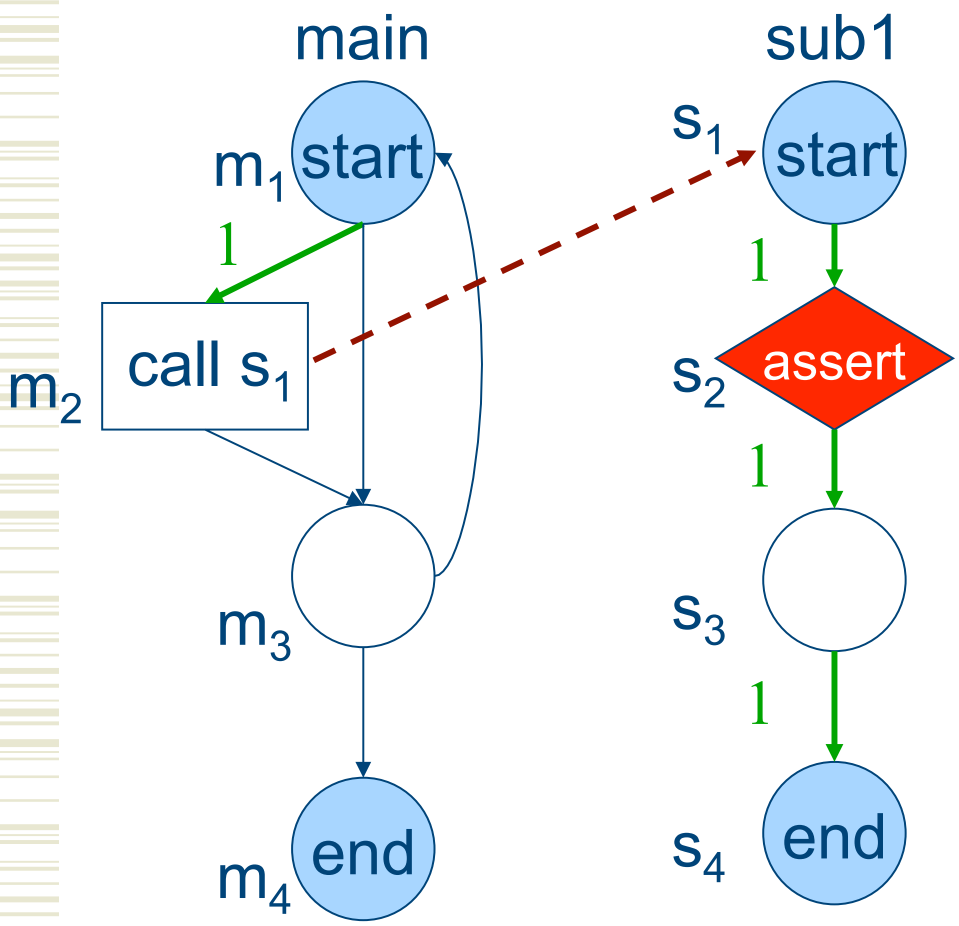
Start DFS at Main CFG



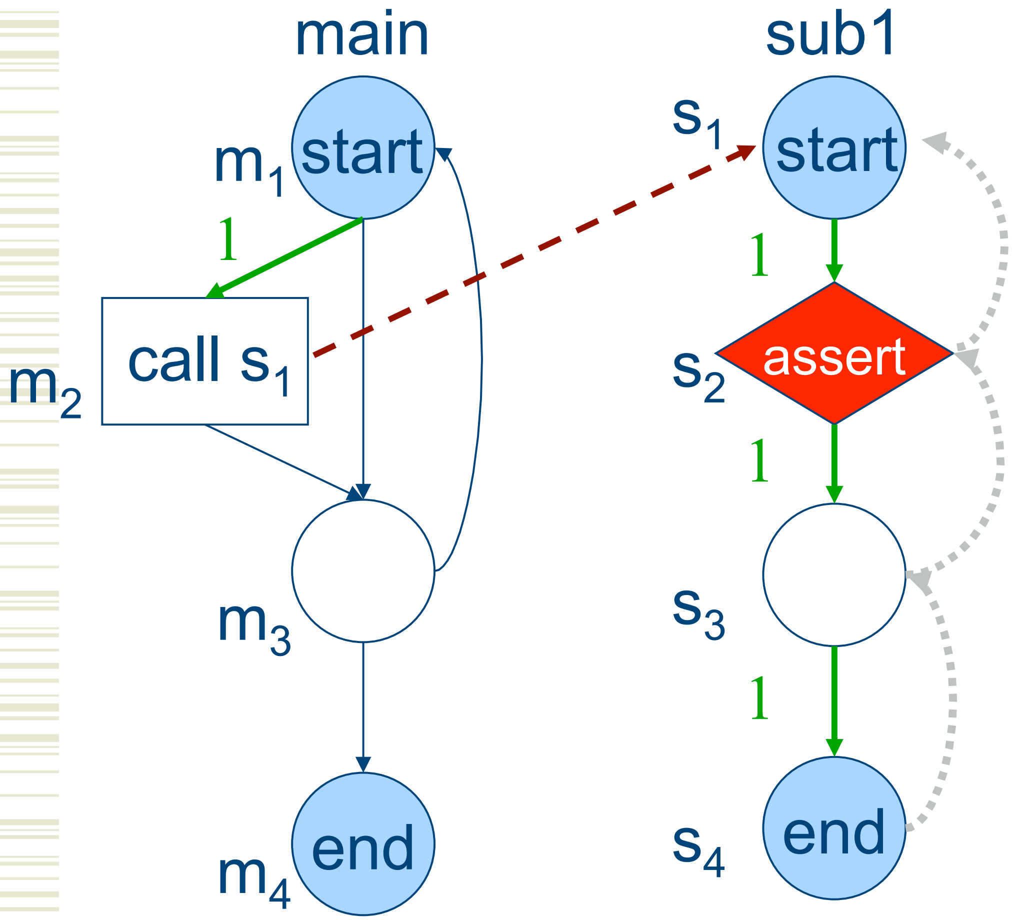
At Call node move to Target



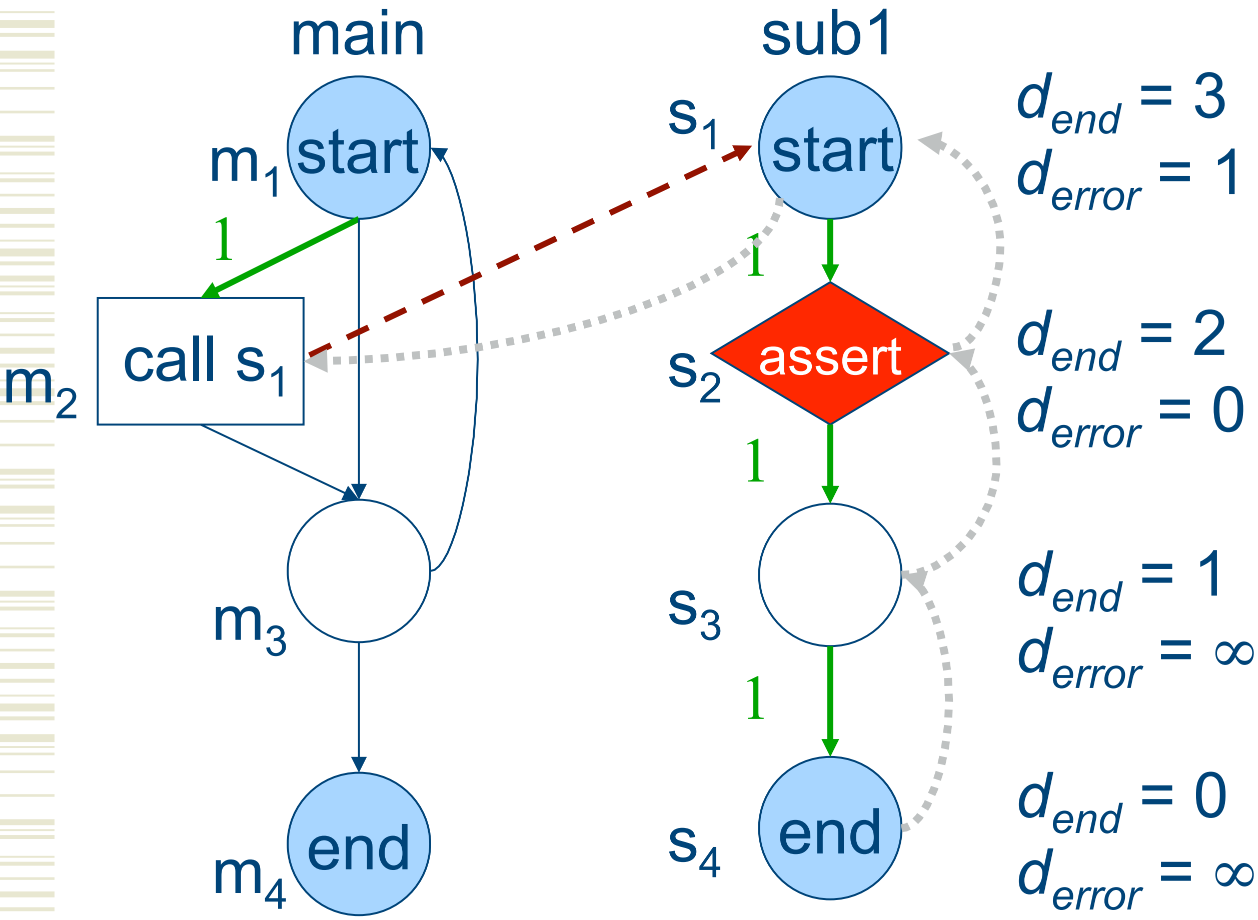
Note the edge costs of nodes



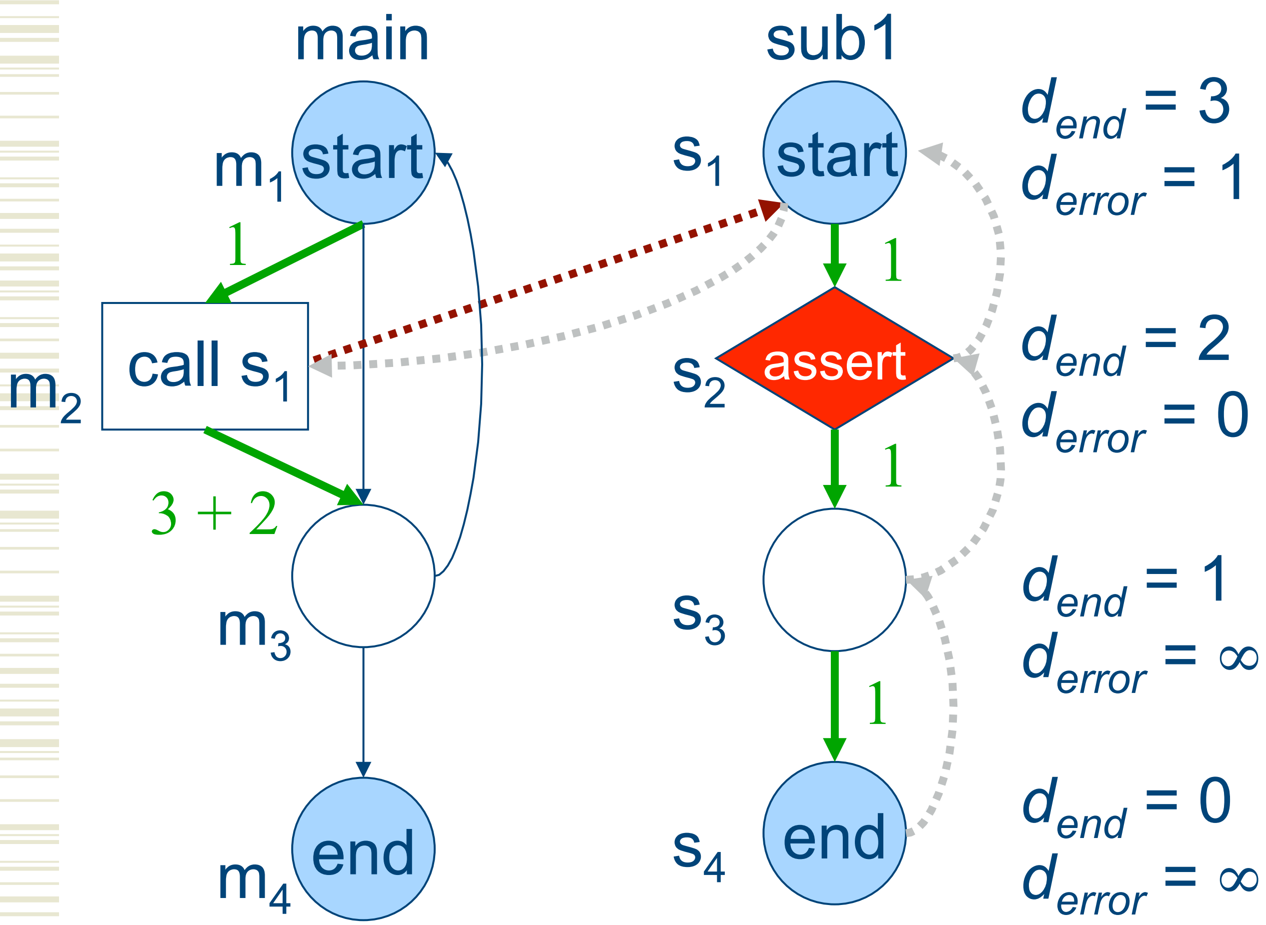
Backtrack at end nodes



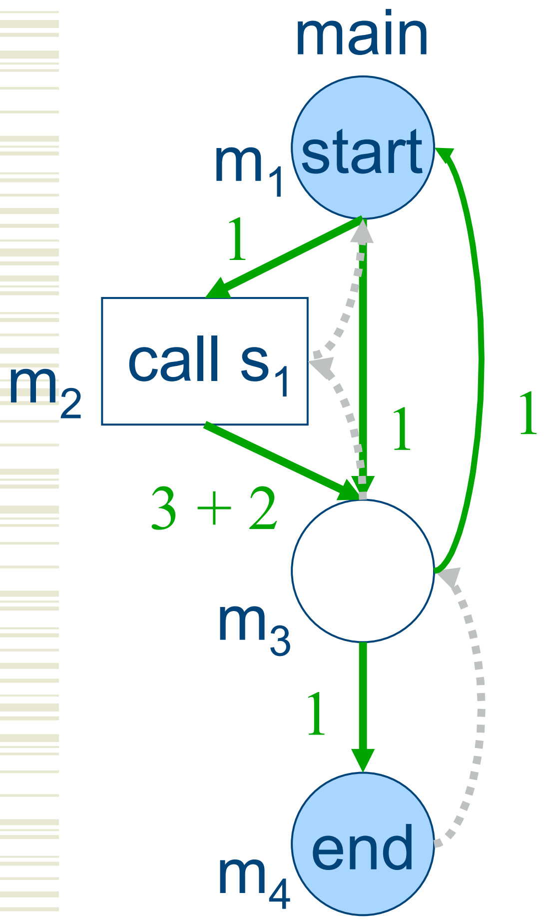
All-pairs Analysis out of Start



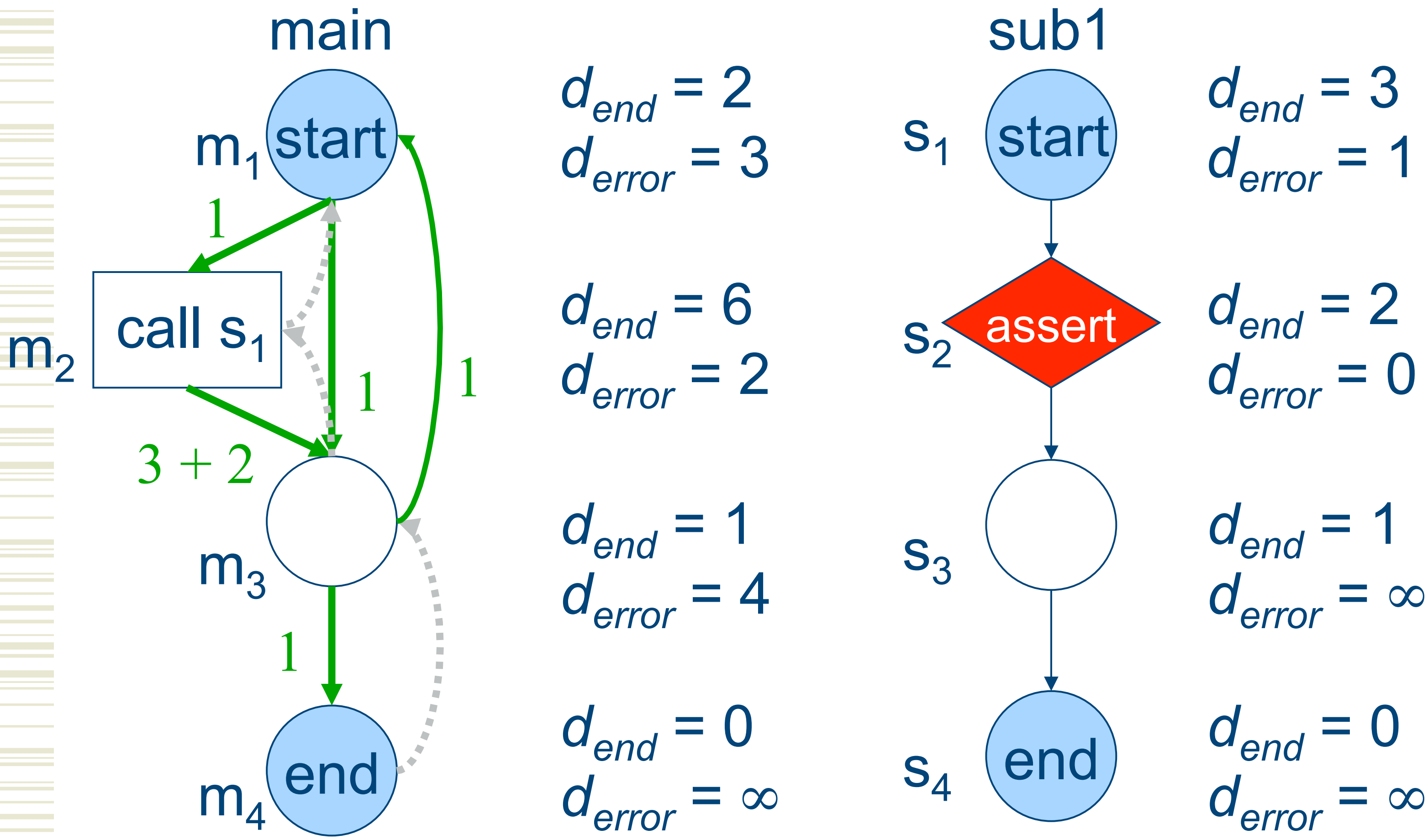
Move Cost of Call to Call-site



Continue Traversal



Trigger All-pairs on Main CFG



Complexity

- Depth-first traversal: $O(N+E)$
- N and E for all nodes and edges
- All-pairs on local CFGs: $O(N_i^3)$
- No exponential growth like the k-bound approach
- Scales better than k-bound approach
- Not limited to k anymore

Performance Analysis

- M. Dwyer, S. Person, and S. Elbaum (FSE '06)
- Understand "hardness" of benchmark
- Measure error density with depth-bounded randomized DFS
- At each DFS level, pick random successor
- Run 1000 experiments on a cluster of machines
- Count number of experiments that find error
- Error density is ratio of error discovery runs to total experiments
- Hardness is inversely proportional to error density

Super Computer

- Marylou 4 (among the top 50 supercomputers)
- 630 nodes with 2 dual core processors at 2.6GHz
- Each node has 8 GB RAM
- One hour time bound
- We get 1024 processors at a time.
- Makes testing go quickly for random experiments

Random DFS vs. e-FCA

Depth = 2 Barbershop	e-FCA	Rand DFS **Min over 1000 Runs**	Rand DFS **Average over 1000 Runs**	Error Density **over 1000 runs**
T = 5	814	1,570	255,720	99.2%
T = 9	1,070	2,258	47,017	92.0%
T = 15	1,448	2,988	37,195	80.3%
T = 20	1,767	3,844	69,445	22.5%
T = 30	2,401	4,412	5,161	2.00%
T = 40	3,736	3,894	5,135	0.30%

Guided Search Results

- FCA estimates used with runtime trace is e-FCA
- Use the gnu-debugger based model checker Estes
- Benchmark set of programs with concurrency errors
- Pentium III, 1.5 GHz processor with 2 GB of RAM

- NOTE: Still depend on default search order because we do not randomize ties in priority queue

Time in seconds: Static analysis

Model	FSM	EFSM	**e-FCA**
Hyman (2) K=1 max=3	0	3	0
Hyman (2) K= 1 max=4	1	11	0
Hyman (2) K=1 max=5	1	27	0
Dining Phil (3) K=1 max=2	1	76	0
Dining Phil (3) K=1 max=3	1	146	0
Dining Phil (3) K=0 max=4	1	4	1
Dining Phil (3) K=0 max=5	2	7	1

Time in seconds: Static analysis

Model	FSM	EFSM	**e-FCA**
Hyman (2) K=1 max=3	0	3	0
Hyman (2) K= 1 max=4	1	11	0
Hyman (2) K=1 max=5	1	27	0
Dining Phil (3) K=1 max=2	1	76	0
Dining Phil (3) K=1 max=3	1	146	0
Dining Phil (3) K=0 max=4	1	4	1
Dining Phil (3) K=0 max=5	2	7	1

Time in seconds: Static analysis

Model	FSM	EFSM	**e-FCA**
Hyman (2) K=1 max=3	0	3	0
Hyman (2) K= 1 max=4	1	11	0
Hyman (2) K=1 max=5	1	27	0
Dining Phil (3) K=1 max=2	1	76	0
Dining Phil (3) K=1 max=3	1	146	0
Dining Phil (3) K=0 max=4	1	4	1
Dining Phil (3) K=0 max=5	2	7	1

Time in seconds: Static analysis

Model	FSM	EFSM	**e-FCA**
Hyman (2) K=1 max=3	0	3	0
Hyman (2) K= 1 max=4	1	11	0
Hyman (2) K=1 max=5	1	27	0
Dining Phil (3) K=1 max=2	1	76	0
Dining Phil (3) K=1 max=3	1	146	0
Dining Phil (3) K=0 max=4	1	4	1
Dining Phil (3) K=0 max=5	2	7	1

States Generated before Error Discovery

Model	FSM	EFSM	**e-FCA**
Hyman (2) K=1 max=3	10,227	7,160	3,817
Hyman (2) K= 1 max=4	41,791	21,909	13,529
Hyman (2) K=1 max=5	123,743	59,951	38,745
Dining Phil (3) K=1 max=2	53,897	4,594	1,626
Dining Phil (3) K=1 max=3	54,725	13,830	3,816
Dining Phil (3) K=0 max=4	186,419	36,467	13,696
Dining Phil (3) K=0 max=5	334,198	400,474	55,876

States Generated before Error Discovery

Model	FSM	EFSM	**e-FCA**
Hyman (2) K=1 max=3	10,227	7,160	3,817
Hyman (2) K= 1 max=4	41,791	21,909	13,529
Hyman (2) K=1 max=5	123,743	59,951	38,745
Dining Phil (3) K=1 max=2	53,897	4,594	1,626
Dining Phil (3) K=1 max=3	54,725	13,830	3,816
Dining Phil (3) K=0 max=4	186,419	36,467	13,696
Dining Phil (3) K=0 max=5	334,198	400,474	55,876

States Generated before Error Discovery

Model	FSM	EFSM	**e-FCA**
Hyman (2) K=1 max=3	10,227	7,160	3,817
Hyman (2) K= 1 max=4	41,791	21,909	13,529
Hyman (2) K=1 max=5	123,743	59,951	38,745
Dining Phil (3) K=1 max=2	53,897	4,594	1,626
Dining Phil (3) K=1 max=3	54,725	13,830	3,816
Dining Phil (3) K=0 max=4	186,419	36,467	13,696
Dining Phil (3) K=0 max=5	334,198	400,474	55,876

Time taken in seconds before Error Discovery

Model	FSM	EFSM	**e-FCA**
Hyman (2) K=1 max=3	4	6	1
Hyman (2) K= 1 max=4	17	21	5
Hyman (2) K=1 max=5	49	56	16
Dining Phil (3) K=1 max=2	31	79	1
Dining Phil (3) K=1 max=3	28	155	3
Dining Phil (3) K=0 max=4	113	27	8
Dining Phil (3) K=0 max=5	178	388	32

Time taken in seconds before Error Discovery

Model	FSM	EFSM	**e-FCA**
Hyman (2) K=1 max=3	4	6	1
Hyman (2) K= 1 max=4	17	21	5
Hyman (2) K=1 max=5	49	56	16
Dining Phil (3) K=1 max=2	31	79	1
Dining Phil (3) K=1 max=3	28	155	3
Dining Phil (3) K=0 max=4	113	27	8
Dining Phil (3) K=0 max=5	178	388	32

Barbershop: scalability across threads

Thread No	Depth =2	Depth =5	Depth=9
5	814	7,064	92,434
15	1,448	7,698	93,608
20	1,767	8,071	93,927
25	2.086	8,336	94,246
30	2,401	8,970	94,561
40	3,040	9,603	92,500

Barbershop: scalability across threads

Thread No	Depth =2	Depth =5	Depth=9
5	814	7,064	92,434
15	1,448	7,698	93,608
20	1,767	8,071	93,927
25	2.086	8,336	94,246
30	2,401	8,970	94,561
40	3,040	9,603	92,500

Conclusions and Future Work

- e-FCA more efficient in our benchmarks
- Has some hope to scale to larger systems
- Works in the presence of PO reduction

- What else in the concrete state of use?
- What if we increase error locations?

Questions

Software Model Checking Lab
Computer Science Department
Brigham Young University
Provo, Utah

Neha Rungta: neha@cs.byu.edu
Eric G. Mercer: egm@cs.byu.edu

http://vv.cs.byu.edu