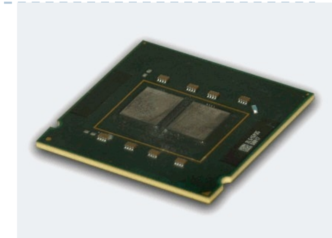
Guided Model Checking for Programs with Polymorphism

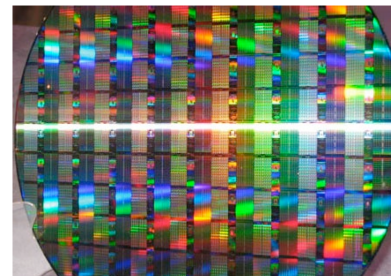
Neha Rungta & Eric Mercer
Computer Science Department, Brigham Young University, Provo, UT, USA

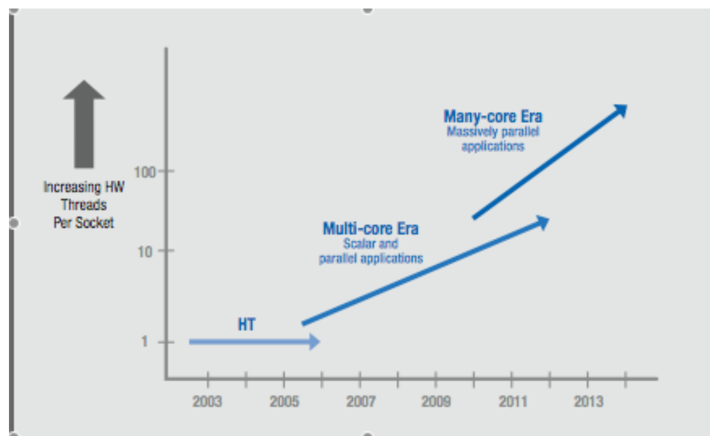
Current Trend

Dual and Quad Core Processors are becoming increasingly common



Intel's 80 core prototype





More processors on a single die *

* Image courtesy Intel white paper

Multi-threaded Programs

- Threads contend for shared resources
- Locks used to force exclusive access
- Incorrect usage leads to errors
- Deadlocks and race conditions are problematic
- Scheduling determines execution order
- Certain execution order cause errors

Software Model Checking

P_a

```
a0: while True do
a1:   wait (turn = 0)
a2:   turn = 1
      end while
```

P_b:

```
b0: while True do
b1:   wait (turn =1)
b2:   turn = 0
      end while
```

Software Model Checking

P_a
```
a0: while True do
a1:   wait (turn = 0)
a2:   turn = 1
      end while
```

P_b:
```
b0: while True do
b1:   wait (turn =1)
b2:   turn = 0
      end while
```

turn=0
L , L

turn=1
L , L

Software Model Checking

P_a

a_0: while *True* do

a_1: wait (turn = 0)

a_2: turn = 1

end while

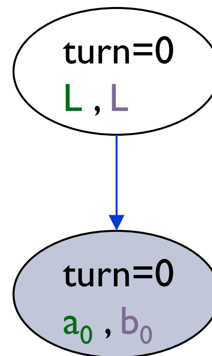
P_b:

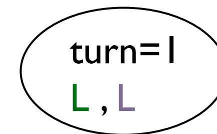
b_0: while *True* do

b_1: wait (turn =1)

b_2: turn = 0

end while



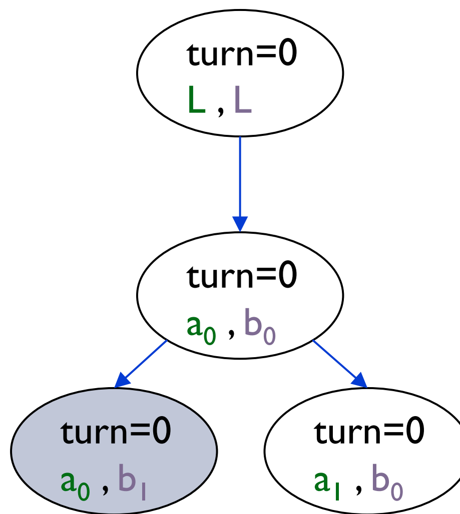


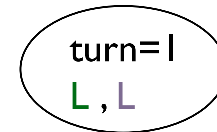
Software Model Checking

P_a

a_0: while *True* do
a_1: wait (turn = 0)
a_2: turn = 1
end while

P_b:
b_0: while *True* do
b_1: wait (turn = 1)
b_2: turn = 0
end while

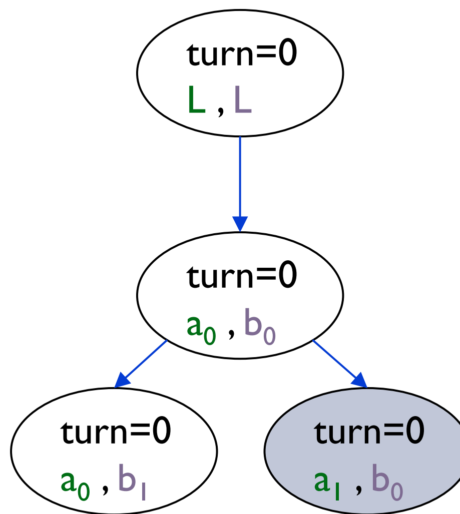


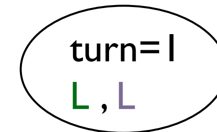


Software Model Checking

P_a
a_0: while *True* do
a_1: wait (turn = 0)
a_2: turn = 1
end while

P_b:
b_0: while *True* do
b_1: wait (turn =1)
b_2: turn = 0
end while





Software Model Checking

P_a

a_0: while *True* do

a_1: wait (turn = 0)

a_2: turn = 1

end while

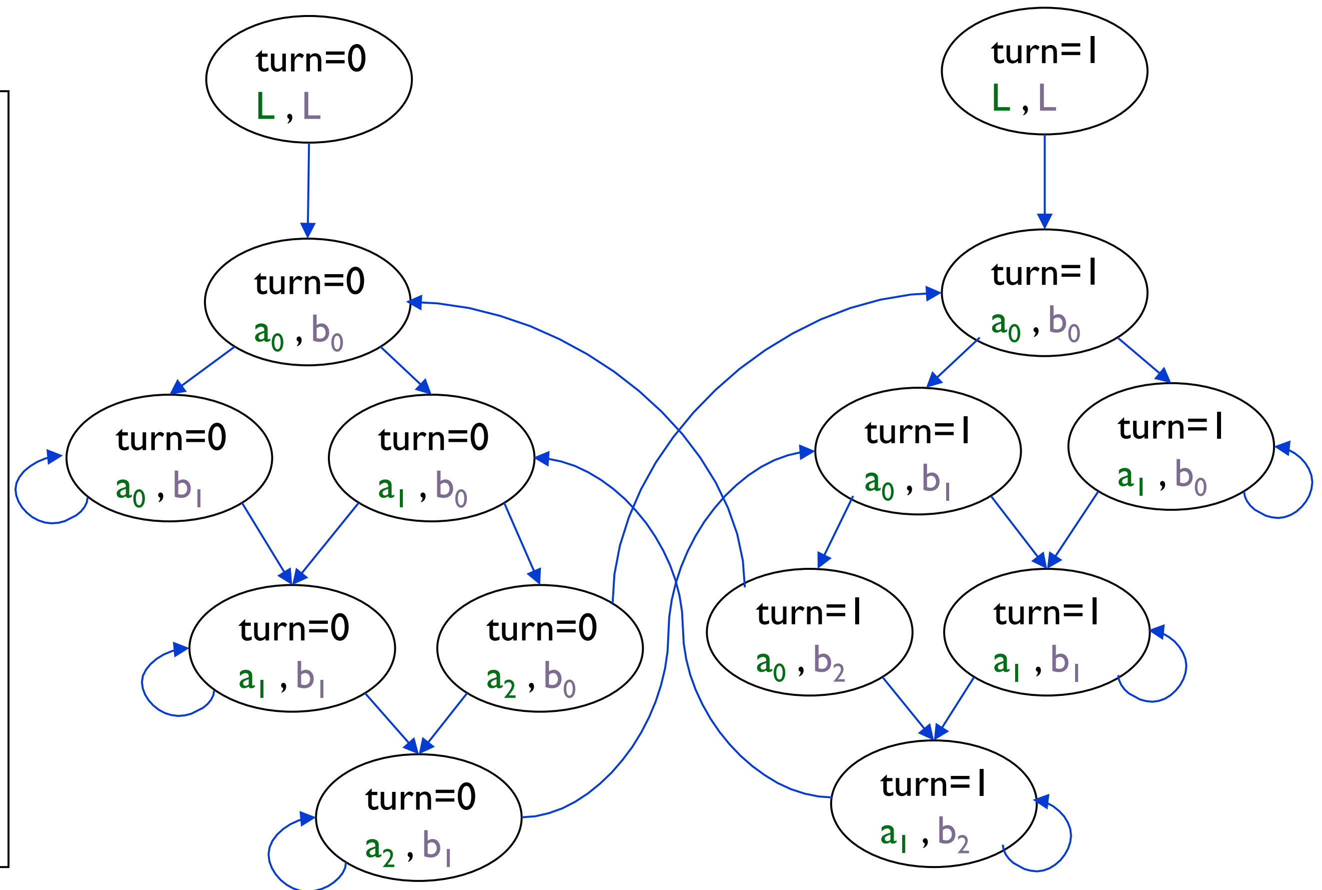
P_b:

b_0: while *True* do

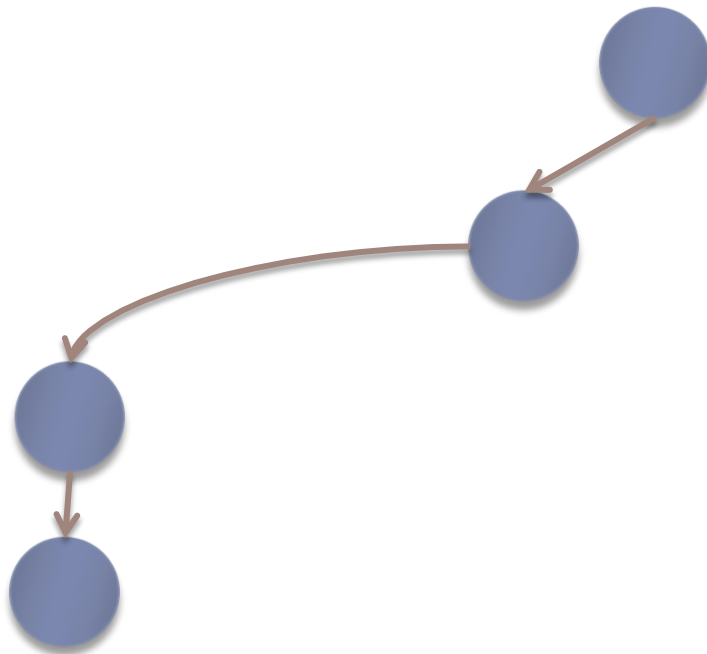
b_1: wait (turn = 1)

b_2: turn = 0

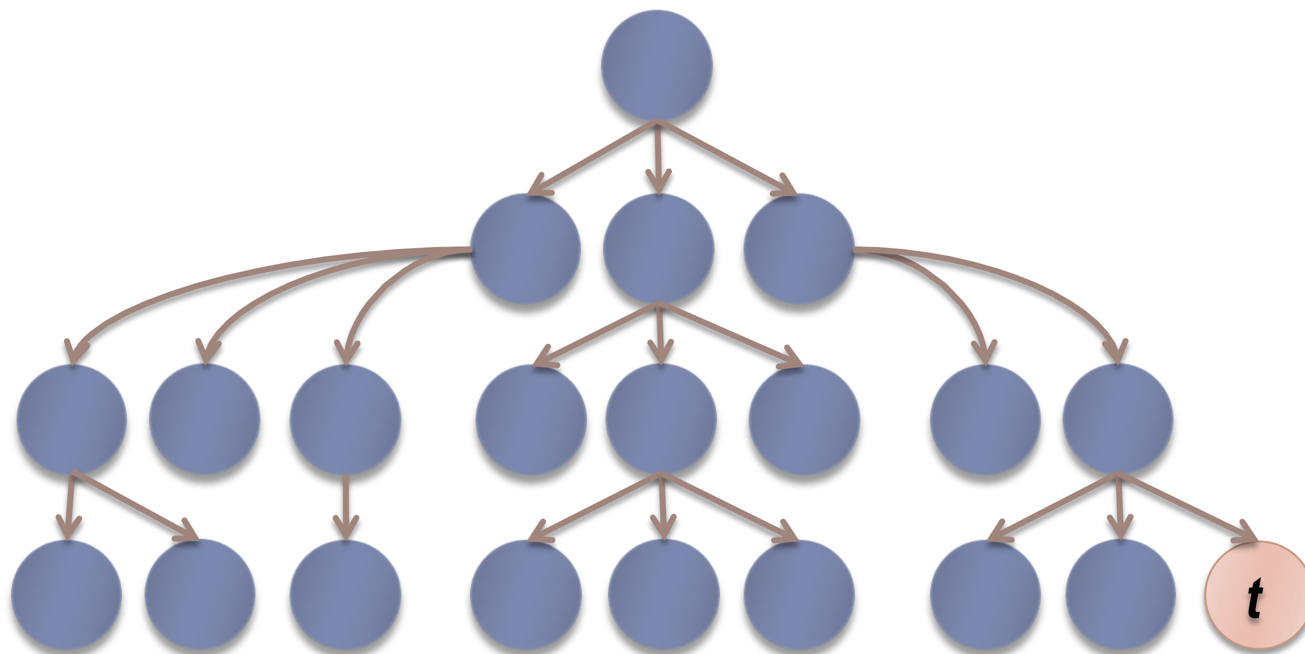
end while



Depth-first search

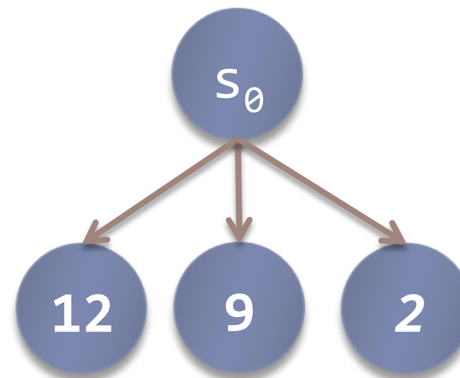


Depth-first search

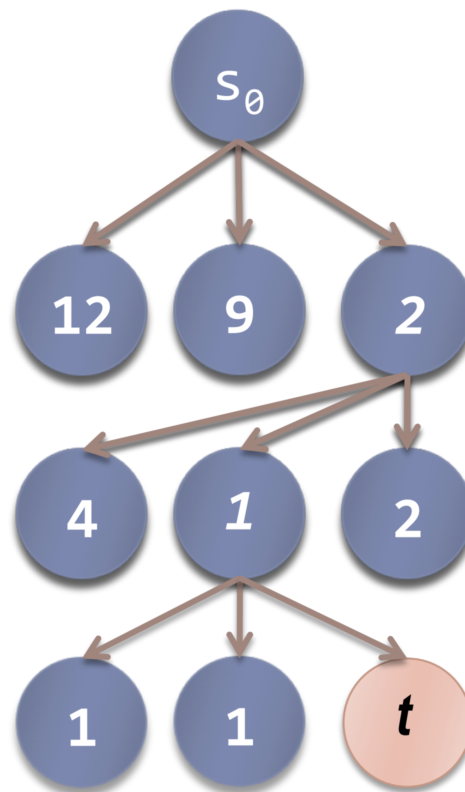


Guided Search

Guided Search

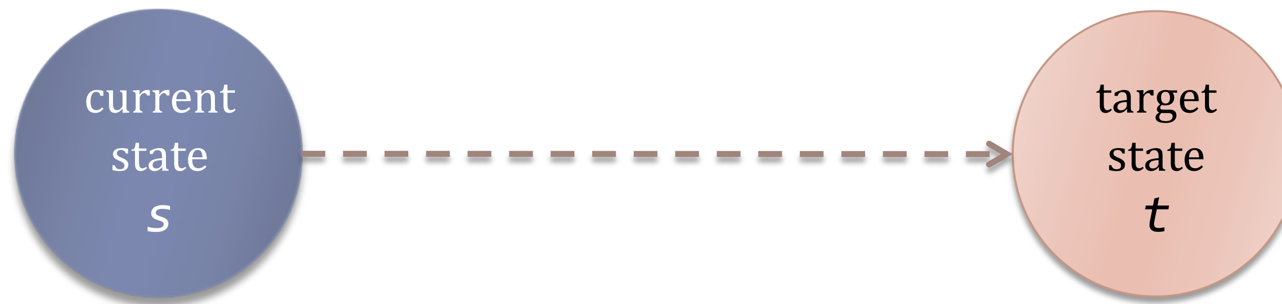


Guided Search

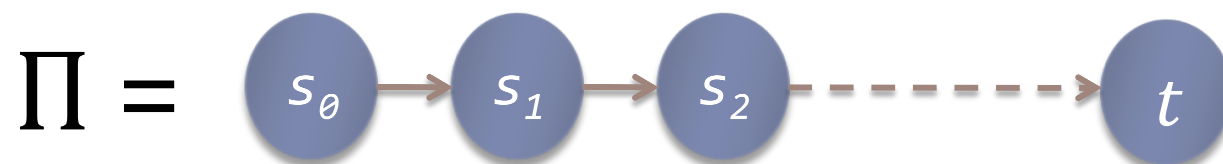


How to Compute the Heuristic

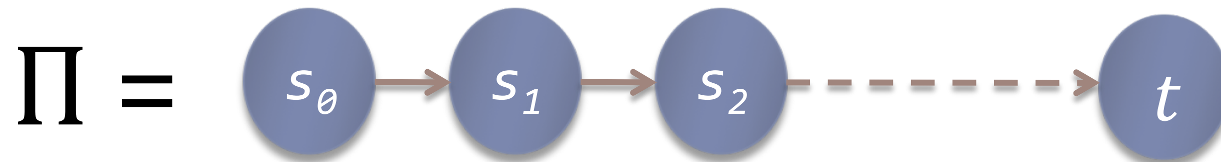
Distance to Target



Distance to Target



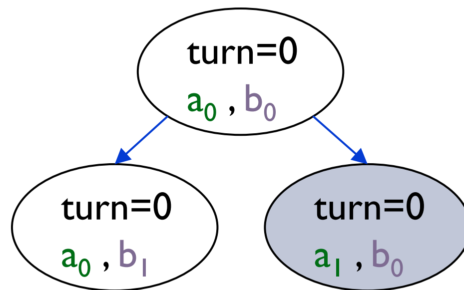
Distance to Target



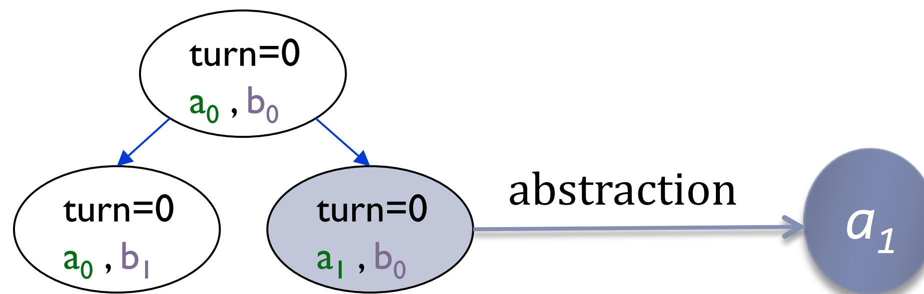
Actual Distance:

$d(s_0, t) = |\pi|$

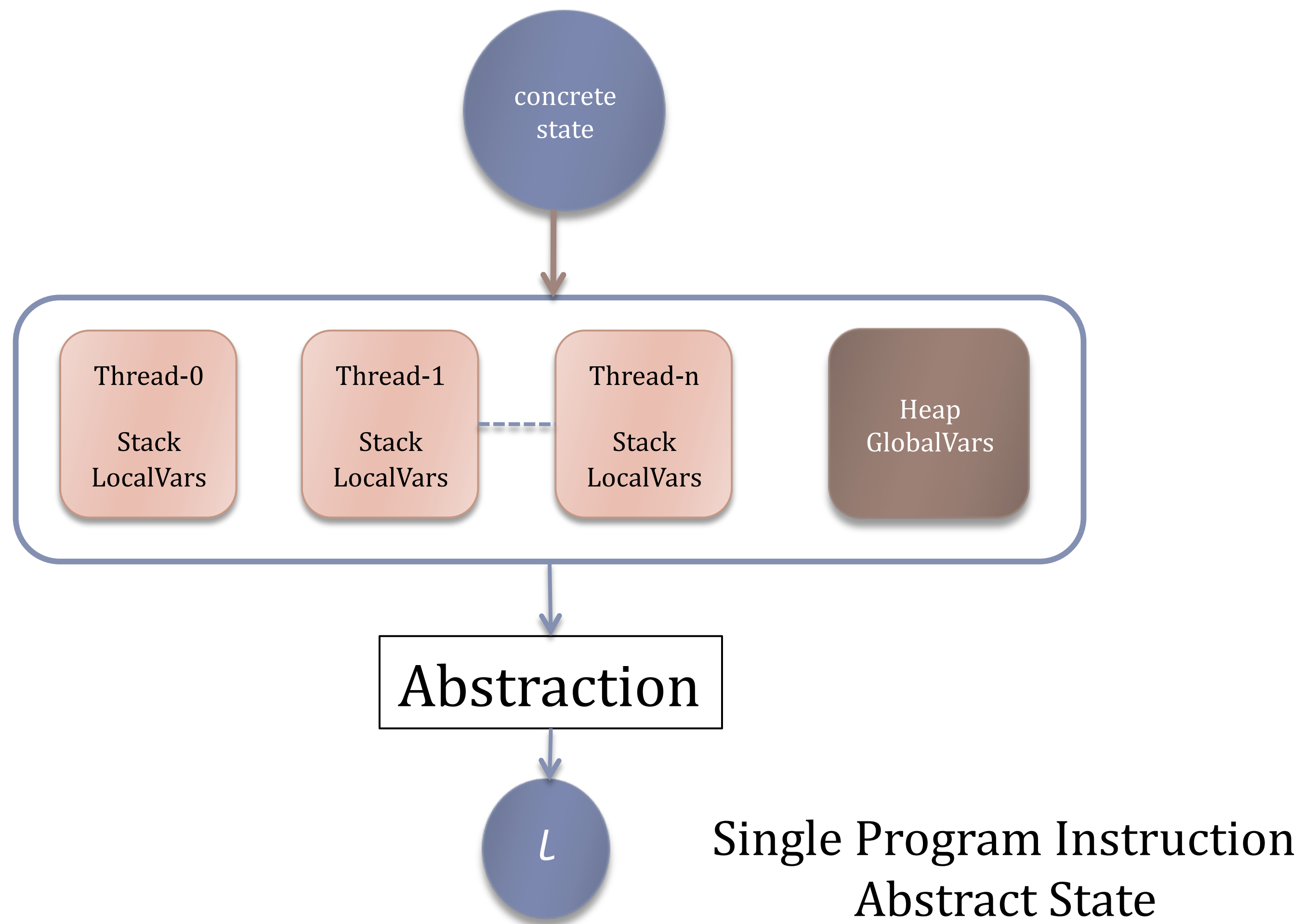
Abstraction



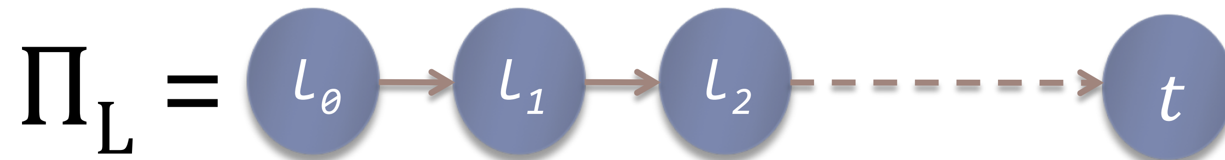
Abstraction



Abstraction



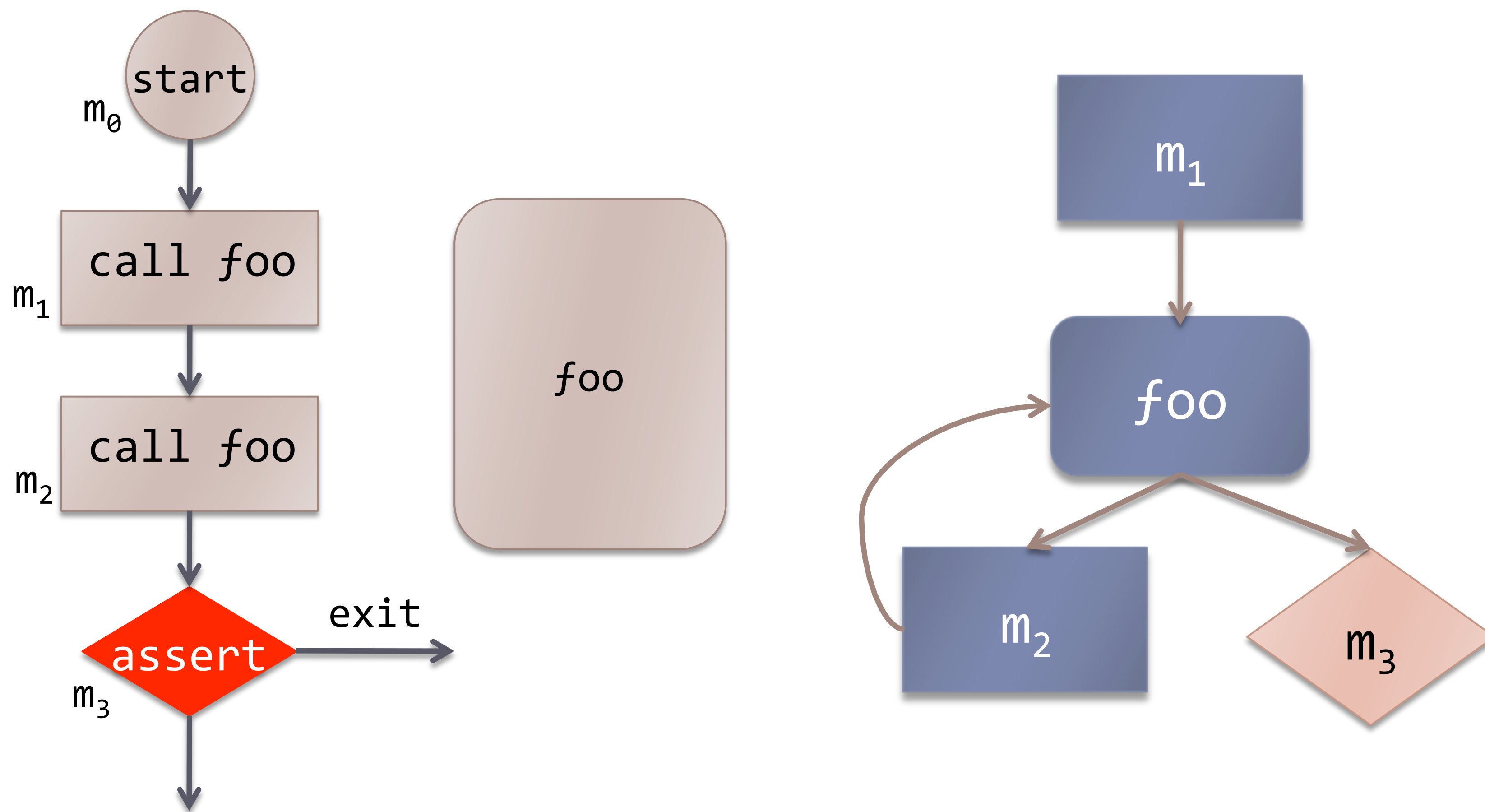
Heuristic: An Estimation



Estimated Distance:

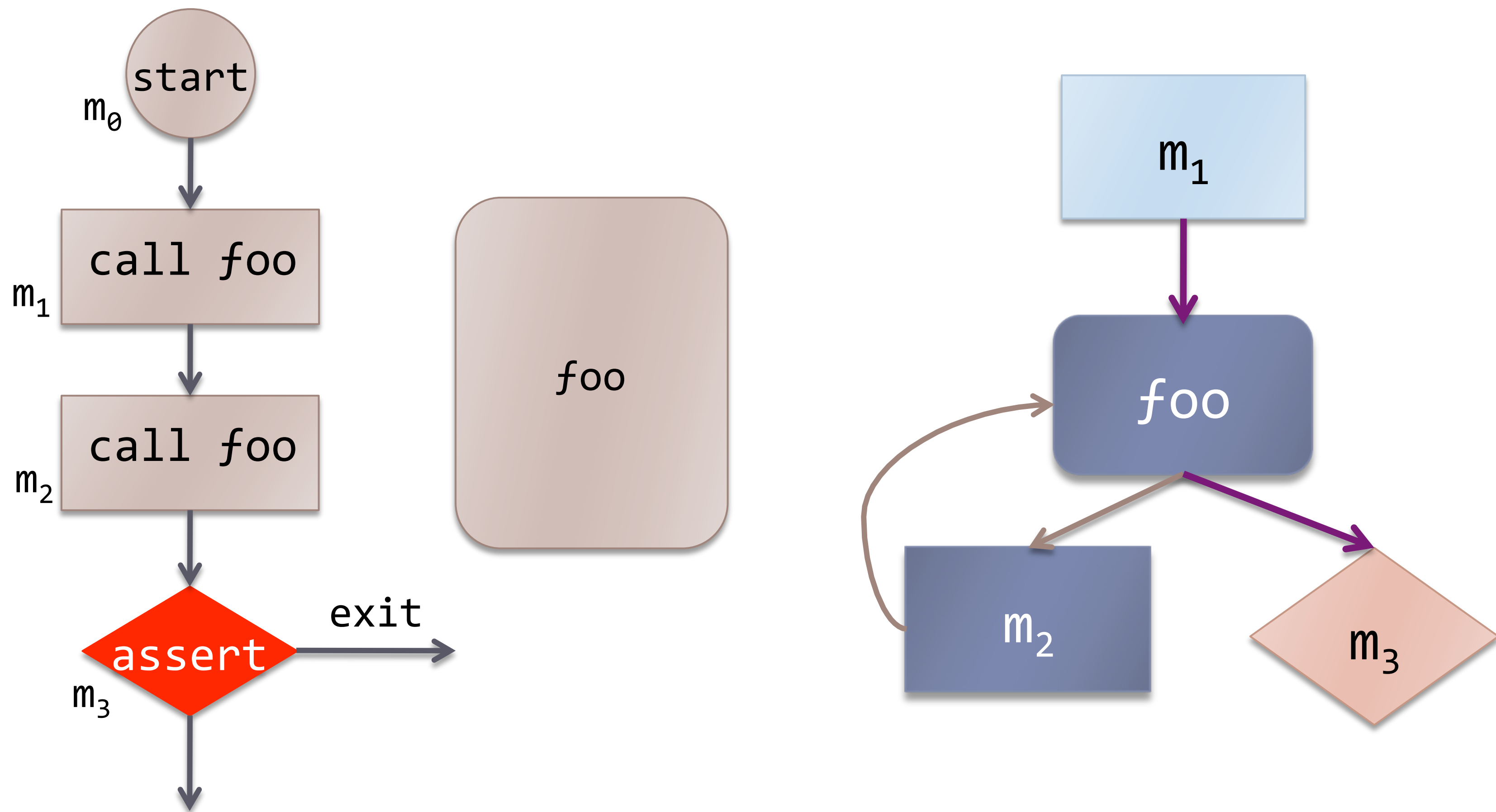
$h(l, t) = | \pi |$

FSM Distance Heuristic



(Edelkamp & Mehler 2003)

FSM Distance Heuristic



(Edelkamp & Mehler 2003)

Motivation

- Increasing use of Java and C#
- Inherently encourage use of Polymorphism
- Used to develop concurrent applications
- JDK Concurrent Library
- Dynamic Method Invocation
- Cannot resolve types statically

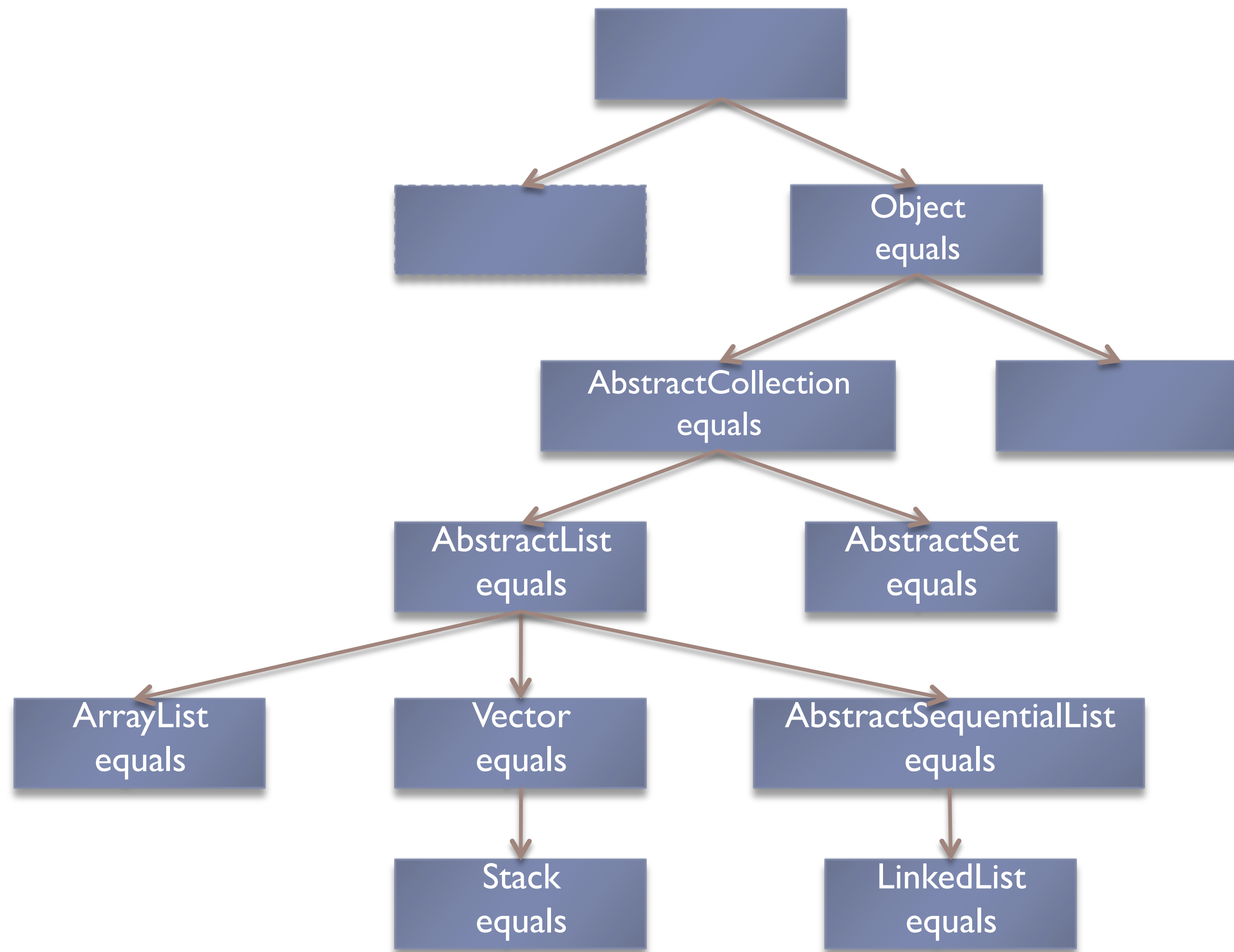
Motivation

```
class AbstractList implements List{
    …
 public boolean equals (Object o) {
l0:  if o == this then
l1:     return true;
l2:  if !(o instanceof List) then
l3:     return false;
l4:  ListIterator x:= ListIterator();
l5:  ListIterator y:= (List o).listIterator();
l6:  while x.hasNext() and y.hasNext()
l7:      Object o1 := x.next();
l8:      Object o2 := y.next();
l9:     if !(o1 == null ? o2 == null : o1.equals(o2)) then
l10:        return false;
l11: return !(x.hasNext() || y.hasNext())
   }
}
```

Motivation

```
class AbstractList implements List{
    …
 public boolean equals (Object o) {
l0:  if o == this then
l1:     return true;
l2:  if !(o instanceof List) then
l3:     return false;
l4:  ListIterator x:= ListIterator();
l5:  ListIterator y:= (List o).listIterator();
l6:  while x.hasNext() and y.hasNext()
l7:      Object o1 := x.next();
l8:      Object o2 := y.next();
l9:     if !(o1 == null ? o2 == null : o1.equals(o2)) then
l10:         return false;
l11: return !(x.hasNext() || y.hasNext())
   }
}
```

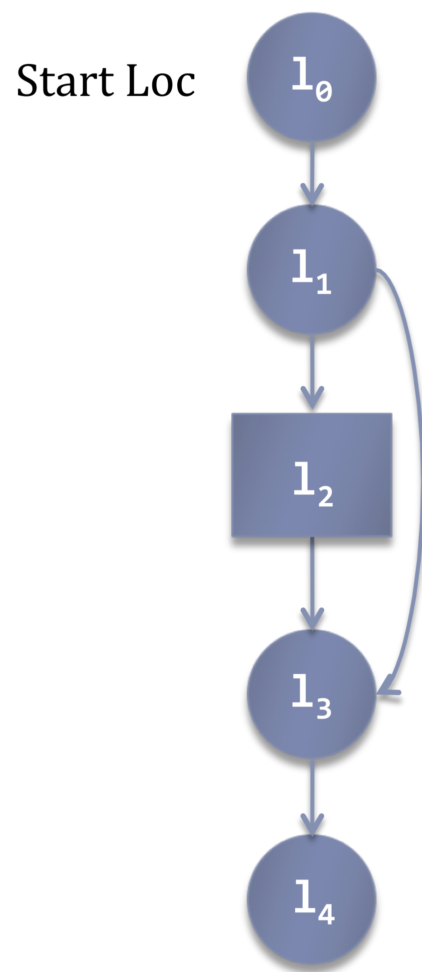
Motivation



Polymorphic Distance Heuristic (PFSM)

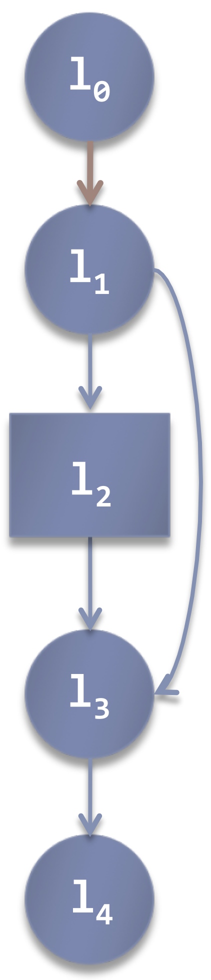
- Static Analysis Phase
- Compute a lower-bound
- Dynamic Heuristic Computation
- Use runtime information to resolve types
- Refine distance estimates

Static Analysis Phase



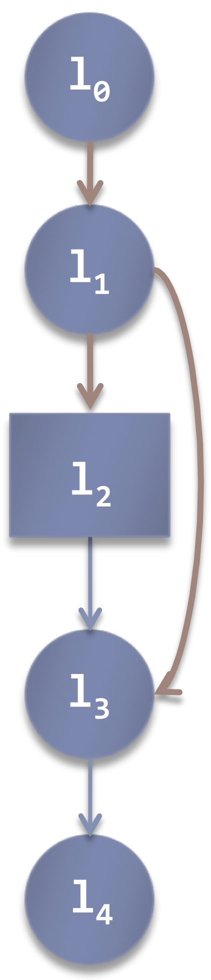
	l_0	l_1	l_2	l_3	l_4
l_0	0	∞	∞	∞	∞
l_1	∞	0	∞	∞	∞
l_2	∞	∞	0	∞	∞
l_3	∞	∞	∞	0	∞
l_4	∞	∞	∞	∞	0

Static Analysis Phase



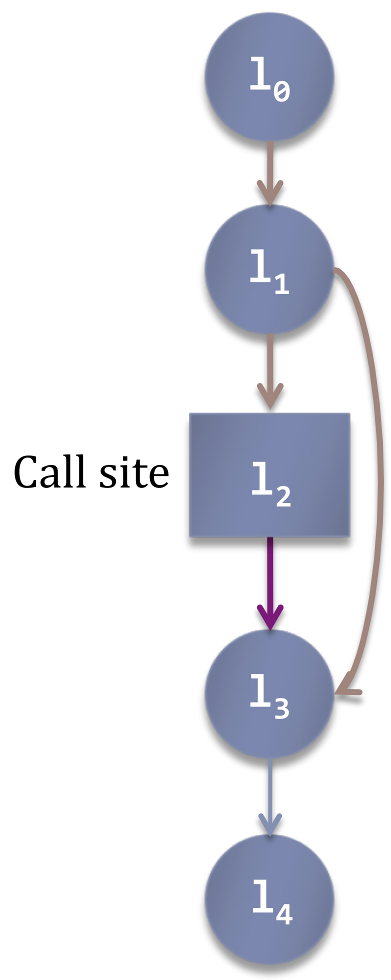
	l_0	l_1	l_2	l_3	l_4
l_0	0	1	∞	∞	∞
l_1	∞	0	∞	∞	∞
l_2	∞	∞	0	∞	∞
l_3	∞	∞	∞	0	∞
l_4	∞	∞	∞	∞	0

Static Analysis Phase



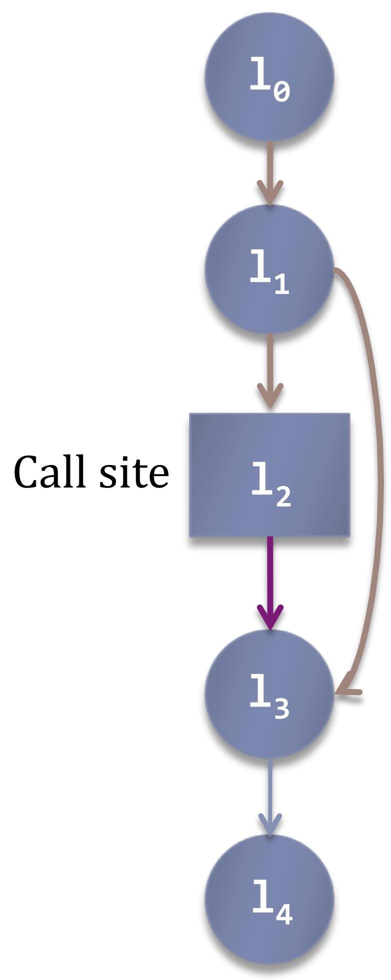
	l_0	l_1	l_2	l_3	l_4
l_0	0	1	∞	∞	∞
l_1	∞	0	1	1	∞
l_2	∞	∞	0	∞	∞
l_3	∞	∞	∞	0	∞
l_4	∞	∞	∞	∞	0

Static Analysis Phase



	l_0	l_1	l_2	l_3	l_4
l_0	0	1	∞	∞	∞
l_1	∞	0	1	1	∞
l_2	∞	∞	0	∞	∞
l_3	∞	∞	∞	0	∞
l_4	∞	∞	∞	∞	0

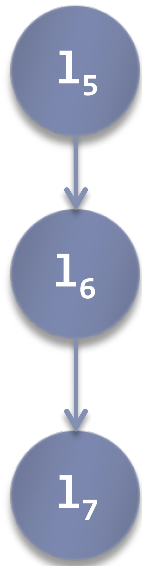
Static Analysis Phase



	l_0	l_1	l_2	l_3	l_4
l_0	0	1	∞	∞	∞
l_1	∞	0	1	1	∞
l_2	∞	∞	0	∞	∞
l_3	∞	∞	∞	0	∞
l_4	∞	∞	∞	∞	0

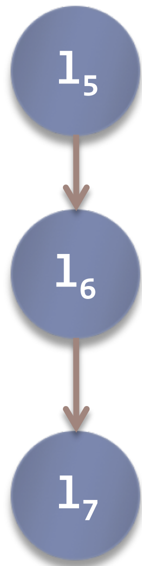
Case 1: The target of the call site is resolved

Reverse Invocation Order



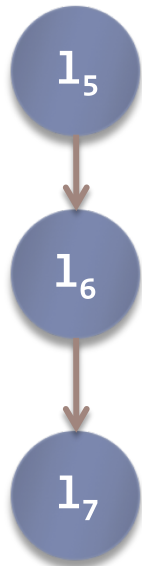
	l_5	l_6	l_7
l_5	0	∞	∞
l_6	∞	0	∞
l_7	∞	∞	0

Reverse Invocation Order



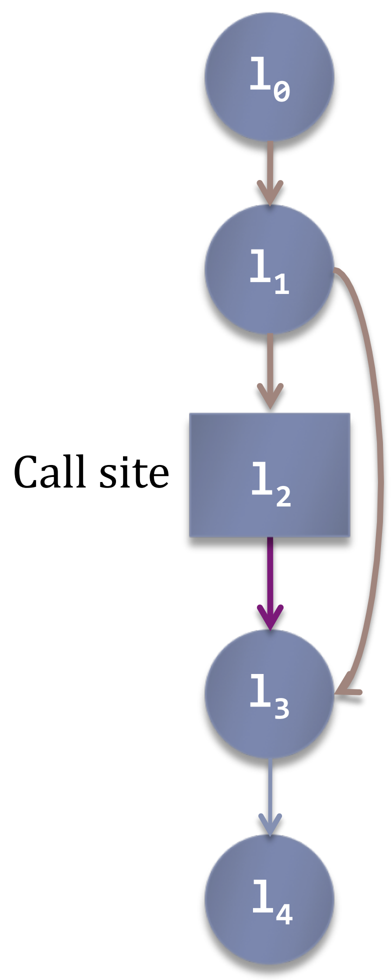
	l_5	l_6	l_7
l_5	0	1	∞
l_6	∞	0	1
l_7	∞	∞	0

Reverse Invocation Order



	l_5	l_6	l_7
l_5	0	1	2
l_6	∞	0	1
l_7	∞	∞	0

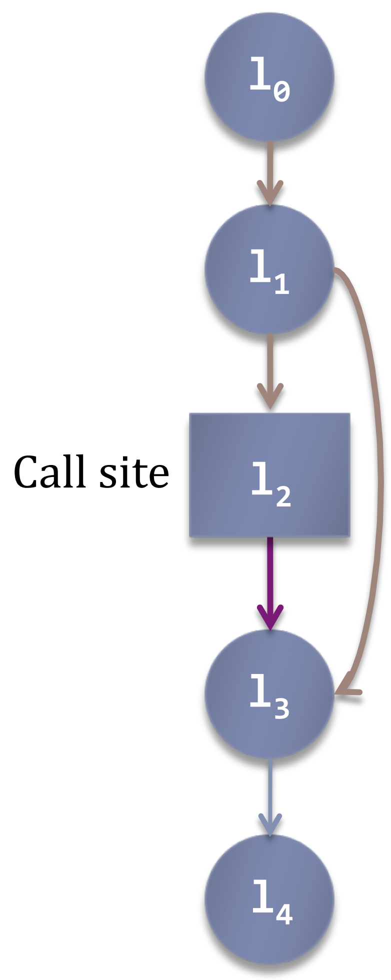
Static Analysis Phase



	l_0	l_1	l_2	l_3	l_4
l_0	0	1	∞	∞	∞
l_1	∞	0	1	1	∞
l_2	∞	∞	0	2+2	∞
l_3	∞	∞	∞	0	∞
l_4	∞	∞	∞	∞	0

Case 1: The target of the call site is resolved

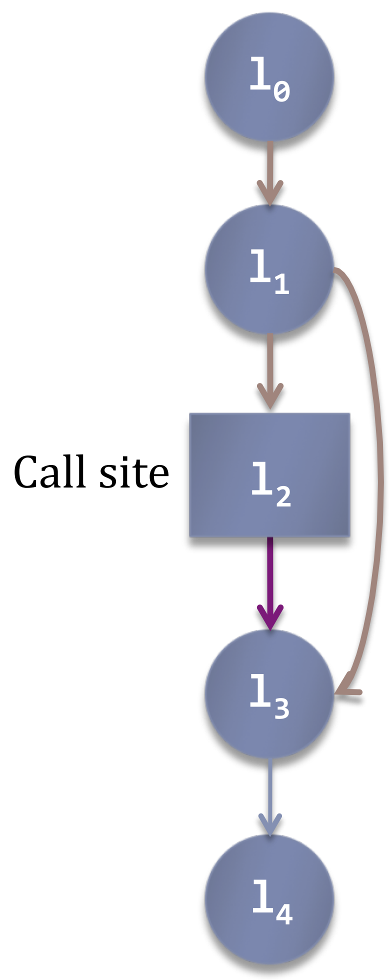
Static Analysis Phase



	l_0	l_1	l_2	l_3	l_4
l_0	0	1	∞	∞	∞
l_1	∞	0	1	1	∞
l_2	∞	∞	0	∞	∞
l_3	∞	∞	∞	0	∞
l_4	∞	∞	∞	∞	0

Case 2: The target of the call site is not resolved

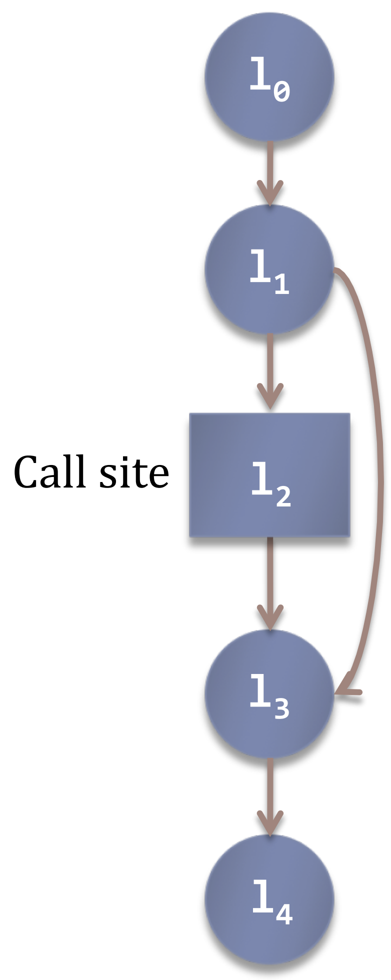
Static Analysis Phase



	l_0	l_1	l_2	l_3	l_4
l_0	0	1	∞	∞	∞
l_1	∞	0	1	1	∞
l_2	∞	∞	0	2	∞
l_3	∞	∞	∞	0	∞
l_4	∞	∞	∞	∞	0

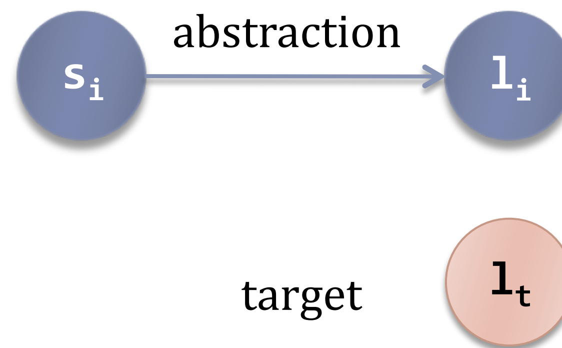
Case 2: The target of the call site is not resolved

Static Analysis Phase



	l_0	l_1	l_2	l_3	l_4
l_0	0	1	2	2	3
l_1	∞	0	1	1	2
l_2	∞	∞	0	2	3
l_3	∞	∞	∞	0	1
l_4	∞	∞	∞	∞	0

Assigning Heuristic Values



Current Location: l_i

Target location: l_t

Dynamic Heuristic Computation

Current Location: $l_i == l_0$

Target location: $l_t == l_3$

	l_0	l_1	l_2	l_3	l_4
l_0	0	1	2	2	3
l_1	∞	0	1	1	2
l_2	∞	∞	0	2	3
l_3	∞	∞	∞	0	1
l_4	∞	∞	∞	∞	0

Dynamic Heuristic Computation

Current Location: $l_i == l_0$

Target location: $l_t == l_3$

	l_0	l_1	l_2	l_3	l_4
l_0	0	1	2	2	3
l_1	∞	0	1	1	2
l_2	∞	∞	0	2	3
l_3	∞	∞	∞	0	1
l_4	∞	∞	∞	∞	0

Dynamic Heuristic Computation

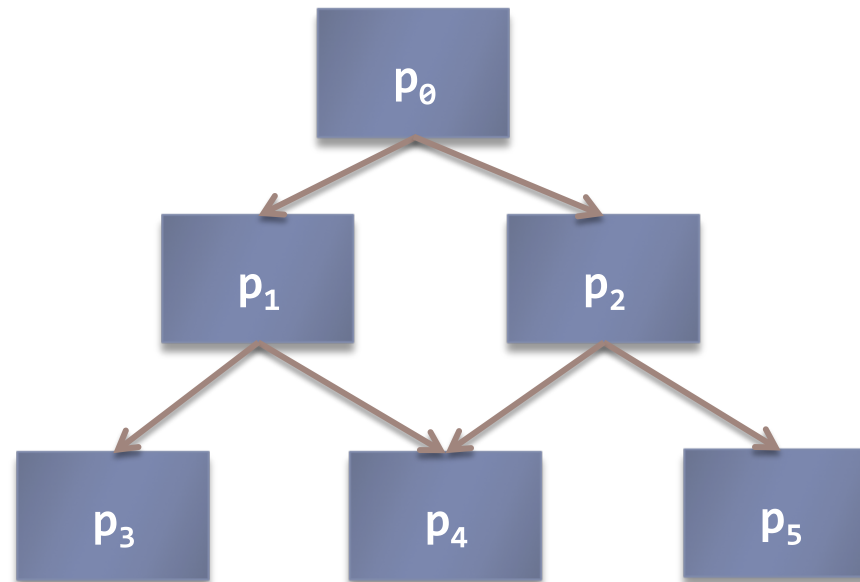
Current Location: $l_i == l_1$

Target location: $l_t == l_{18}$

Dynamic Heuristic Computation

Current Location: $l_i == l_1$

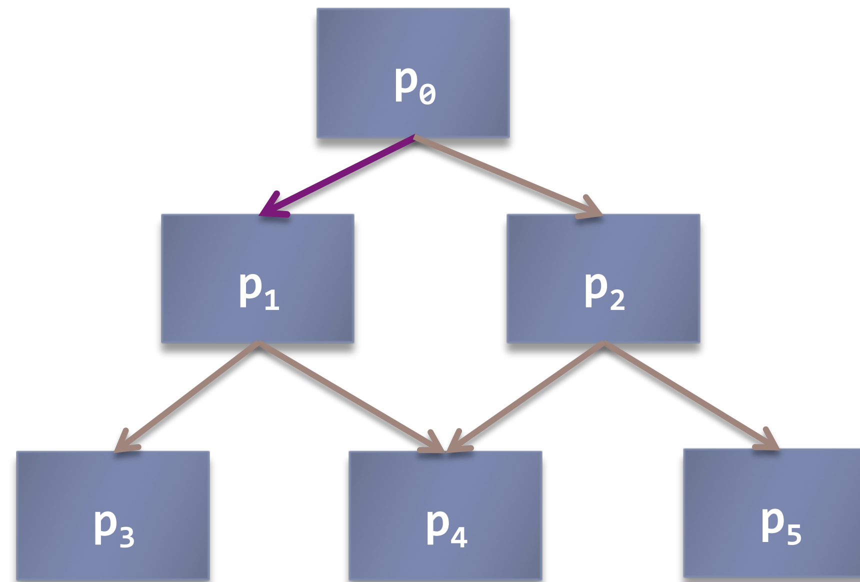
Target location: $l_t == l_{18}$



Dynamic Heuristic Computation

Current Location: $\mathbf{l_i} == \mathbf{l_1}$

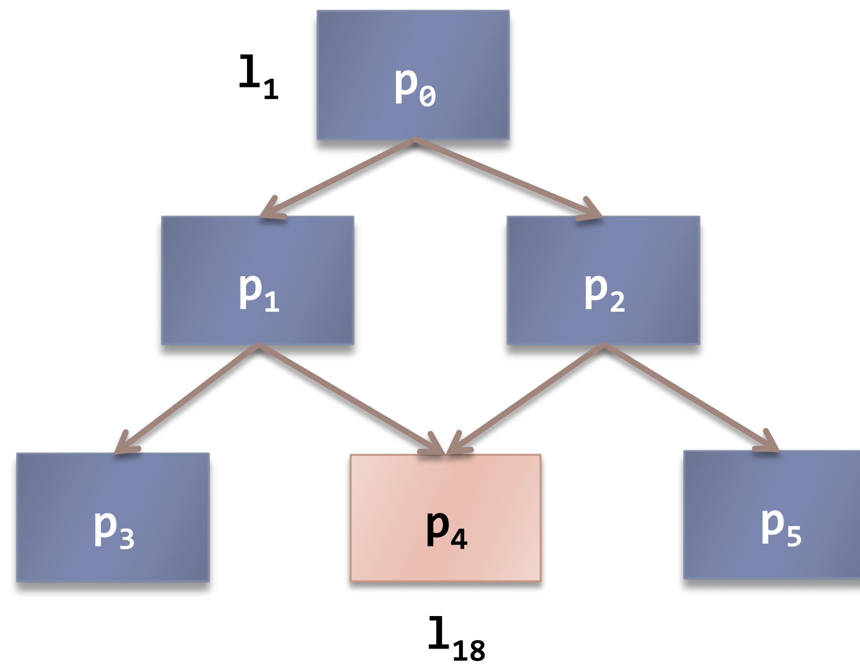
Target location: $\mathbf{l_t} == \mathbf{l_{18}}$



Dynamic Heuristic Computation

Current Location: $l_i == l_1$

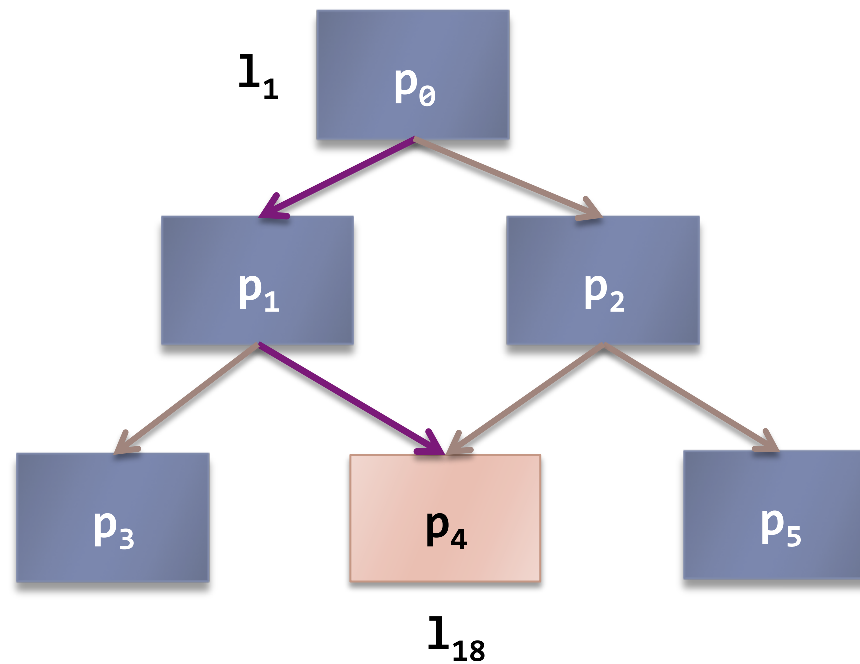
Target location: $l_t == l_{18}$



Dynamic Heuristic Computation

Current Location: $l_i == l_1$

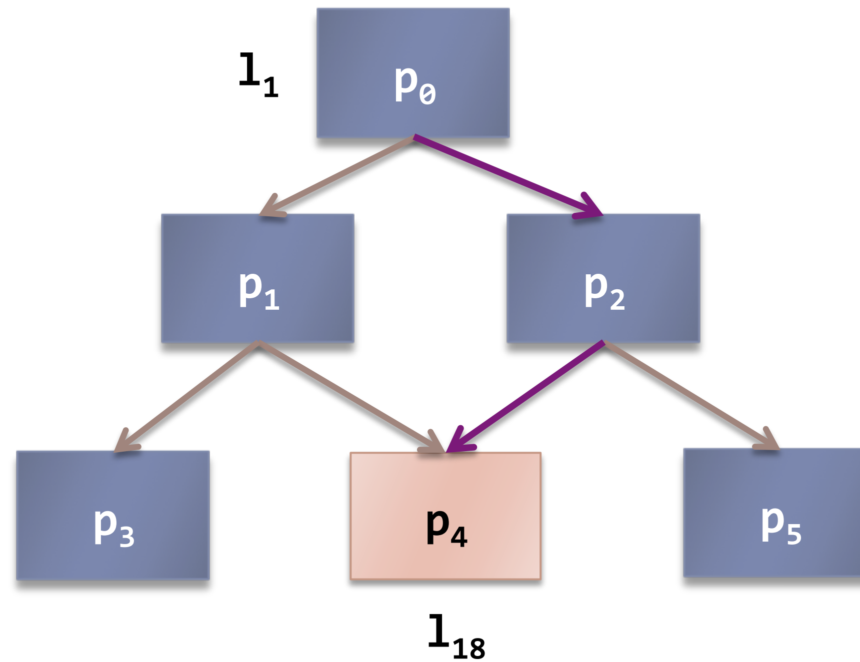
Target location: $l_t == l_{18}$



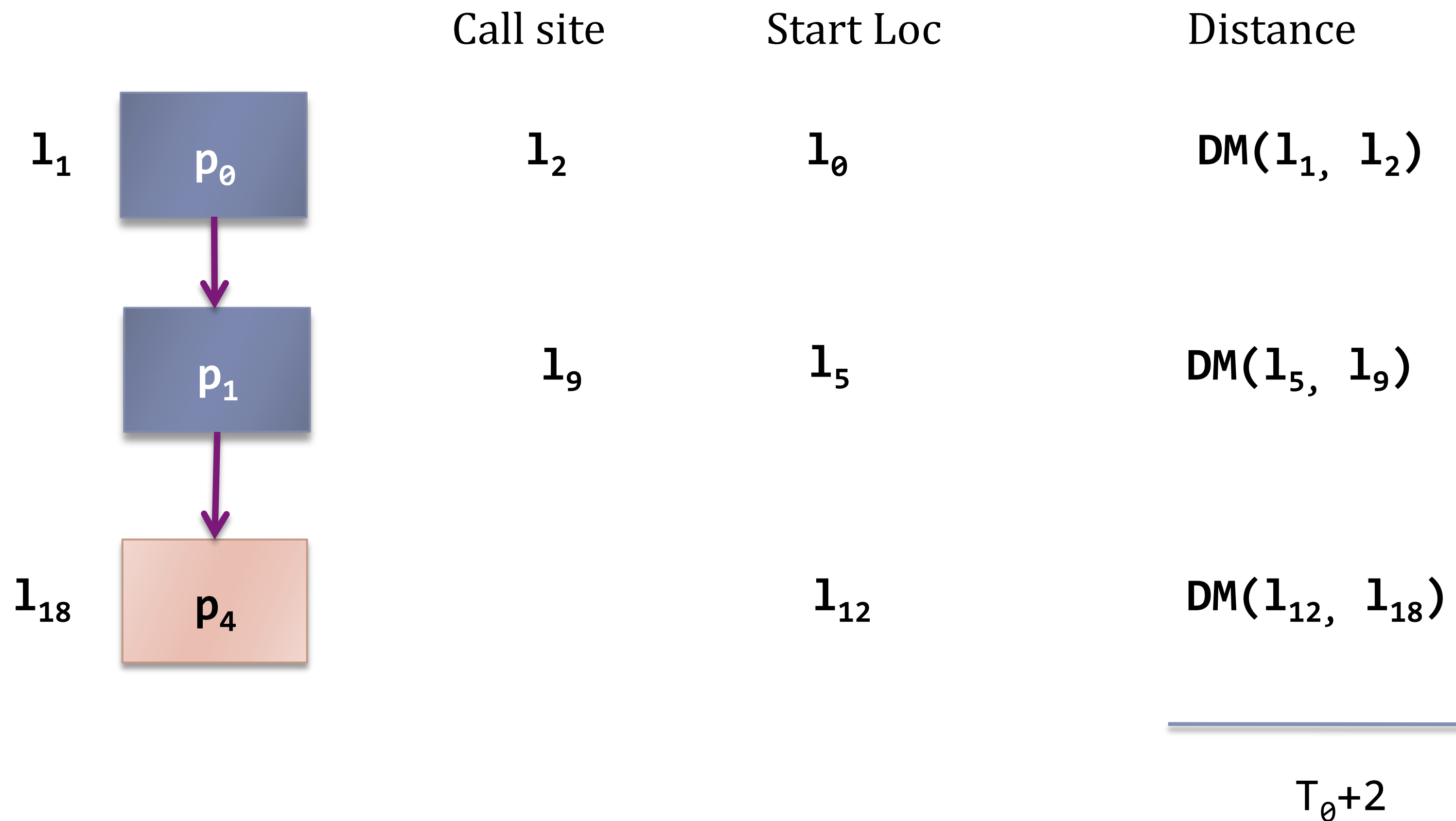
Dynamic Heuristic Computation

Current Location: $l_i == l_1$

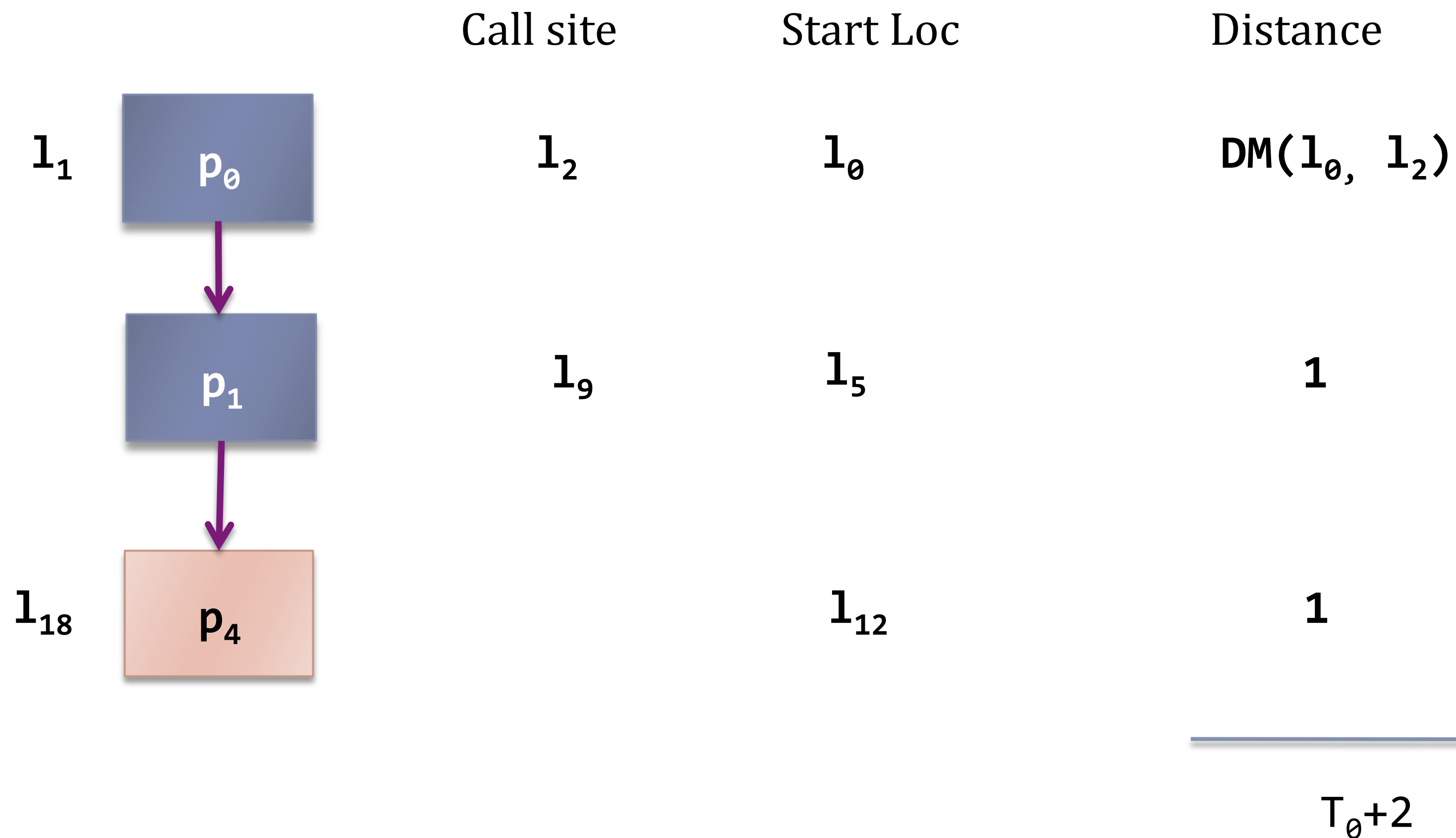
Target location: $l_t == l_{18}$



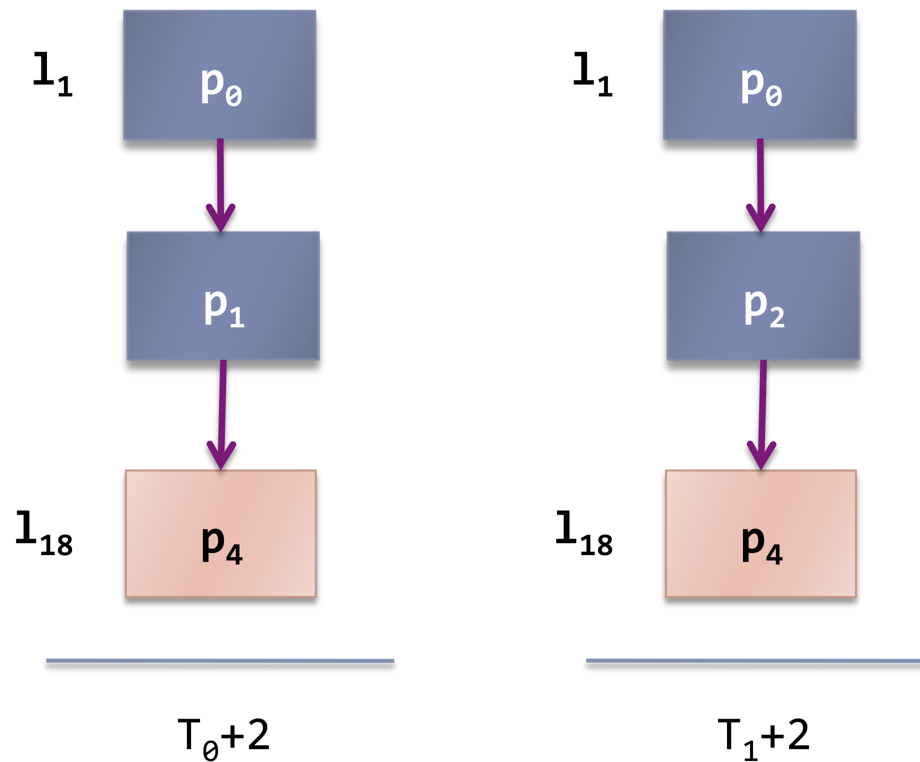
Dynamic Heuristic Computation



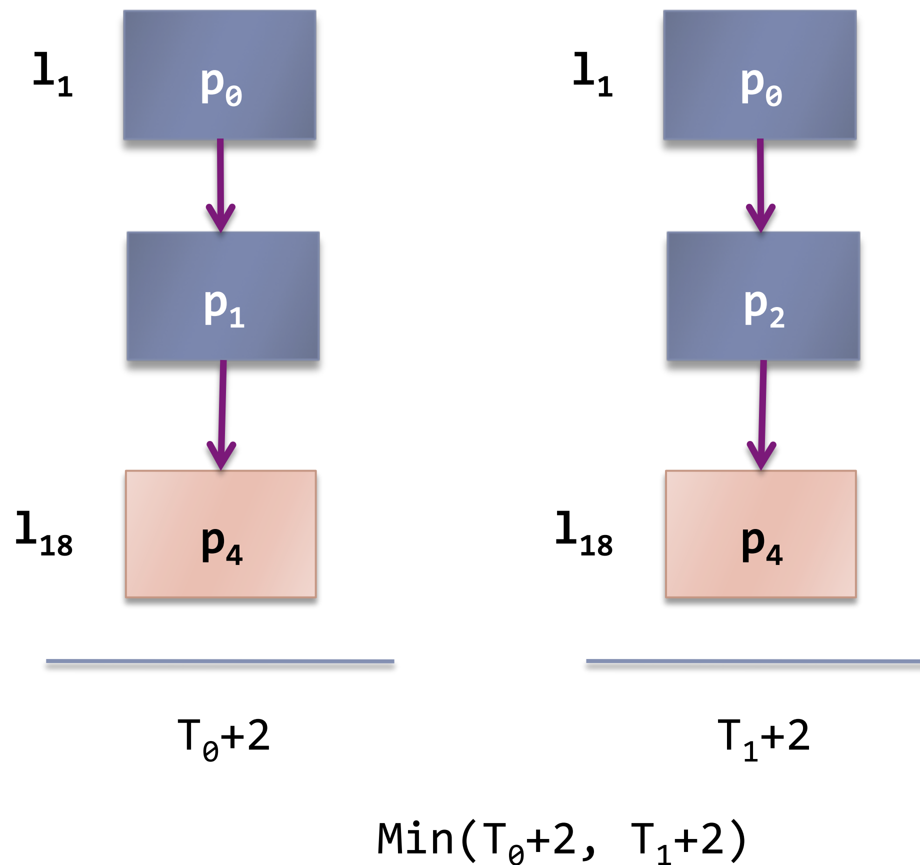
Dynamic Heuristic Computation



Dynamic Heuristic Computation



Dynamic Heuristic Computation



PFSM Distance Heuristic

- Computes a lower bound on the distance estimate
- Admissible and Consistent
- In an A* search optimal counter-example
- Lower Complexity than FSM distance heuristic
- Tighter lower-bound than FSM distance heuristic

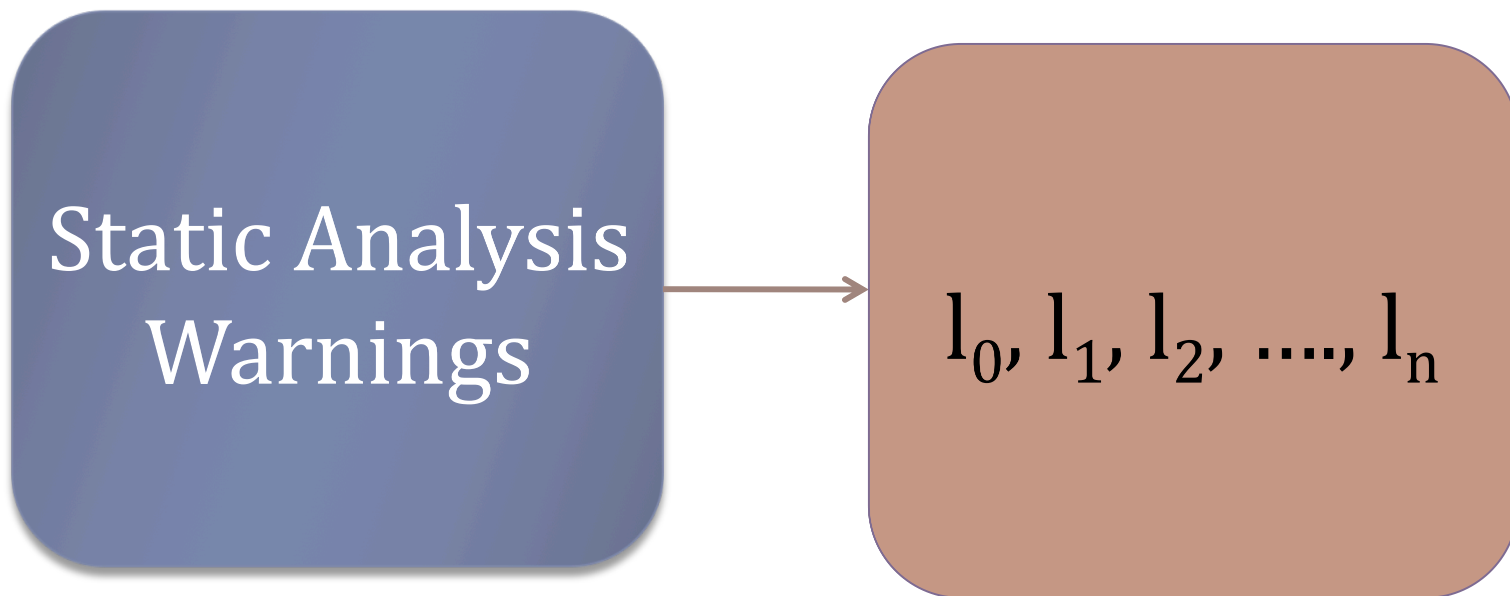
Empirical Evaluation

- 100 Greedy depth-first trials
- Heuristic ties are randomly broken
- Time bound of 1 hour
- Consistent with other recent studies
- Use JPF model checker
- No data non-determinism
- 8 GB RAM and 2.6 GHz processor

Multi-threaded Java programs

- TwoStage Benchmark
  SLOC: 52, Null Pointer Exception
- Reorder Benchmark
  SLOC: 44, Null Pointer Exception
- Wronglock Benchmark
  SLOC: 38, Deadlock
- AbsList Real JDK 1.4 concurrent library
  SLOC: 7267, Race in AbstractList class
- AryList Real JDK 1.4 concurrent library
  SLOC: 7169, Race in ArrayList class

Generating Interesting Locations



Avg. States Explored

Subject	Guided Search		
	PFSM	Random	FSM
TwoStage(7,1)	213	109,259	30,193
TwoStage(8,1)	251	204,790	46,259
TwoStage(10,1)	335	364,859	156,697
WrongLock(1,10)	3,781	7,064	196
Reorder(8,1)	197	34,193	24,022
AryList(1,10)	5,216	15,972	-
AbsList(1,10)	982	10,497,302	-

Average states generated in error discovering trials

Avg. Total Time taken in Seconds

Subject	Guided Search		
	PFSM	Random	FSM
TwoStage(7,1)	0.42	40.14	39.11
TwoStage(8,1)	0.41	76.24	41.87
TwoStage(10,1)	0.46	132.08	59.90
WrongLock(1,10)	1.66	2.85	10.70
Reorder(8,1)	0.39	9.70	13.90
AryList(1,10)	13.60	7.95	-
AbsList(1,10)	4.92	2585.79	-

Related Work

- Hamming Distance Heuristics
  (Yang & Dill 1998)
- Property-based Heuristics
  (Groce & Visser 2002)
- Trail-directed model checking
  (Edelkamp *et. al* 2001)
- Deterministic Execution technique
  (Harvey & Strooper 2001)
- Abstraction guided concrete execution
  (Nanshi & Somenzi 2006, Paula & Hu 2007)
- Concolic Testing
  (Sen *et. al* 2005; Sen & Agha 2007)

Conclusions & Future work

- Distance heuristic for programs with polymorphism
- Conservative estimates in initial static analysis
- Targets of dynamic method invocation resolved
- Compute distance estimates with more information
- PFSM outperforms Random and FSM Heuristic

- Study trade-off between accuracy and performance
- Propagate types to improve accuracy of PFSM heuristic

Questions?

Neha Rungta (neha@cs.byu.edu)
Eric Mercer (egm@cs.byu.edu)

Verification & Validation Lab
Brigham Young University
Provo, UT 84606, USA

http://neharungta.com