

Predicate-based Test Coverage and Generation

17-654/17-765

Analysis of Software Artifacts

Jonathan Aldrich

These slides prepared by Thomas Ball, with additional material from M. Young, A. Memon and MSR's FSE group. Used by permission.

MSIL Unit Test Tool

—————→ *a hybrid helper*

- Goal
 - capture developer knowledge **ASAP**
 - via a **strong** set of unit tests
 - to form a **specification** of the code's behavior
- How
 - generate tests based on analysis of MSIL
 - symbolic execution + constraint satisfaction
 - runtime analysis to check complicated invariants
- Facets
 - complements specification-based test generation
 - positive feedback cycle with programmer

What criteria should
guide unit test generation?

Predicate-complete Testing

- Predicates
 - relational expression such as $(x < 0)$
 - the expression $(x < 0) \parallel (y > 0)$ has two predicates
 - predicates come from program and safe runtime semantics
- Consider a program with m statements and n predicates
 - predicates partition input domain
 - $m \times 2^n$ possible *observable* states S
- Goal of Predicate-complete Testing:
 - cover all *reachable* observable states $R \subseteq S$

PCT Coverage

L2: if (A || B) S else T

L3: if (C || D) U else V

- PCT requires covering all logical combinations over {A,B,C,D} at
 - L2 and L3
 - S, T, U and V
- Some combinations may not be reachable

PCT Coverage Subsumes Statement, Edge Coverage

- Statement coverage
 - Must cover all statements with all predicates
- Edge coverage
 - Must cover if statements with all predicates
 - Therefore touch both branches
- Path coverage
 - Must cover every path through program
 - Infinite number of them!

PCT Coverage does not imply Path Coverage

```
L1:  if (x<0)      (L1, x<0)  (L1, !(x<0))
L2:    skip;      (L2, x<0)
      else
L3:    x = -2;    (L3, !(x<0))
L4:    x = x + 1; (L4, x<0)
L5:    if (x<0)  (L5, x<0)  (L5, !(x<0))
L6:    A;        (L6, x<0)
```

PCT Coverage does not imply Path Coverage

```
L1:  if (x<0)
L2:    skip;
     else
L3:    x = -2;
L4:    x = x + 1;
L5:    if (x<0)
L6:      A;
```

Handwritten annotations in parentheses:

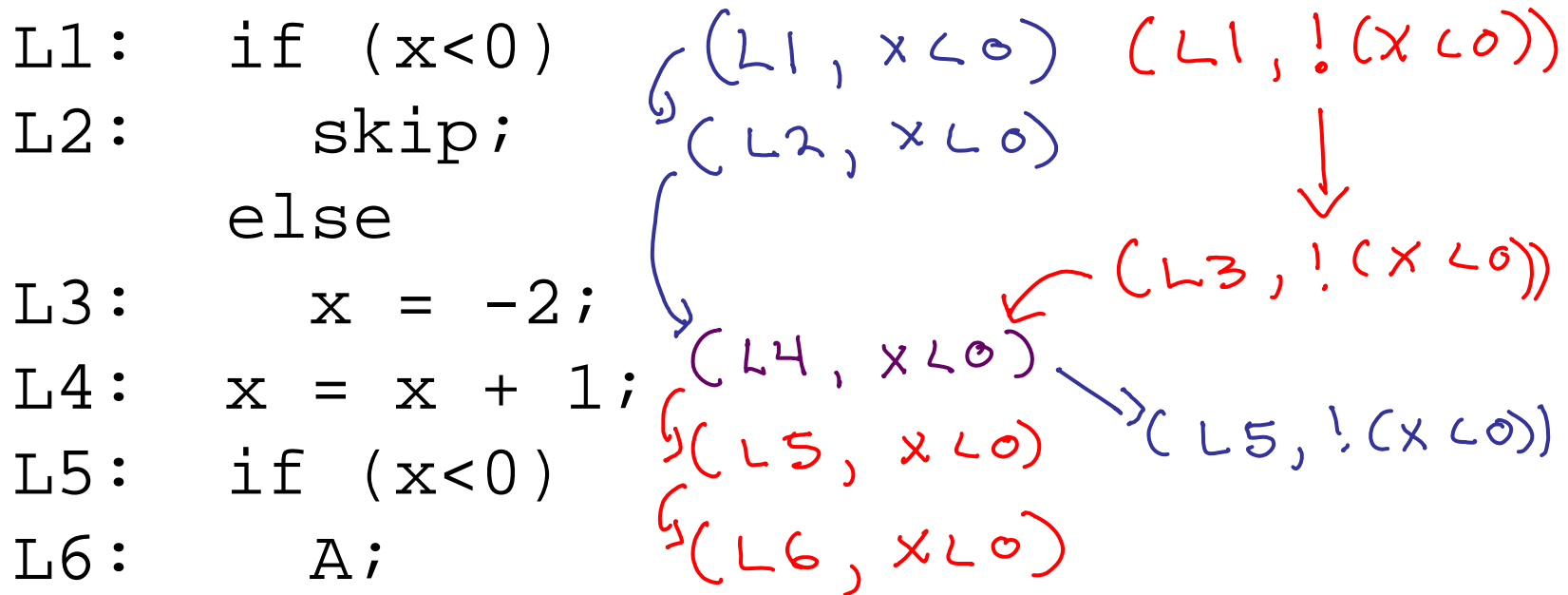
- (L1, x < 0) (L1, !(x < 0))
- (L2, x < 0)
- (L3, !(x < 0))
- (L4, x < 0)
- (L5, x < 0) (L5, !(x < 0))
- (L6, x < 0)

Arrows indicate flow from L1 to L2 and L3, and from L4 to L5.

Three tests

x → -1

PCT Coverage does not imply Path Coverage



Three tests

$x \rightarrow -1$

$x \rightarrow 2$

PCT Coverage does not imply Path Coverage

L1: if (x<0)

L2: skip;

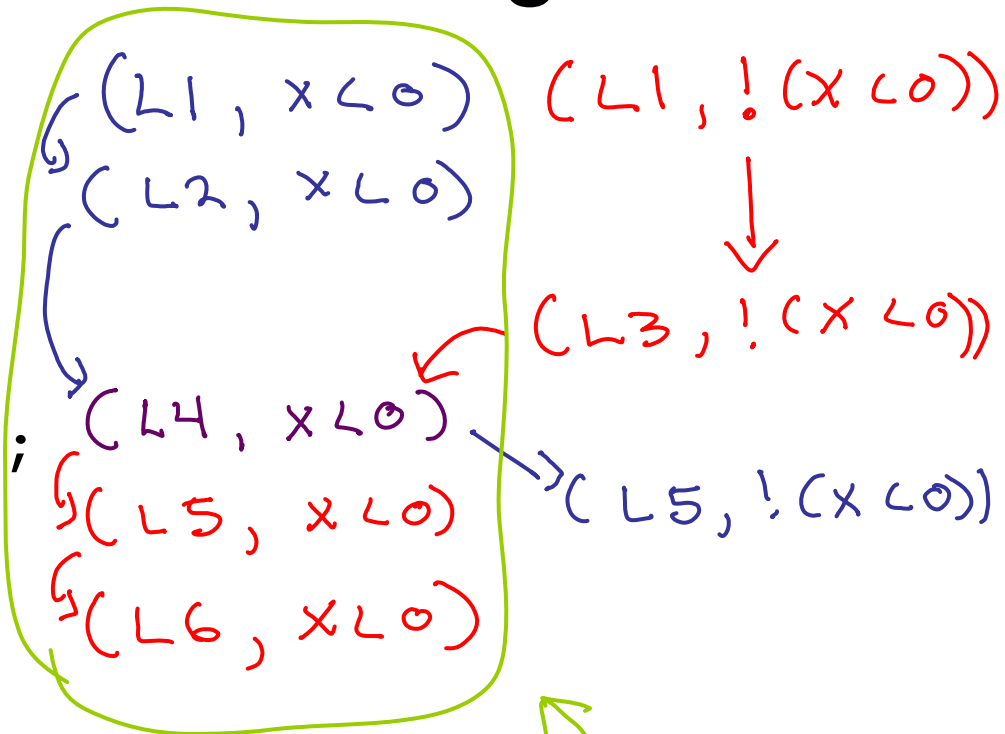
else

L3: x = -2;

L4: x = x + 1;

L5: if (x<0)

L6: A;



Three tests

x → -1

x → 2

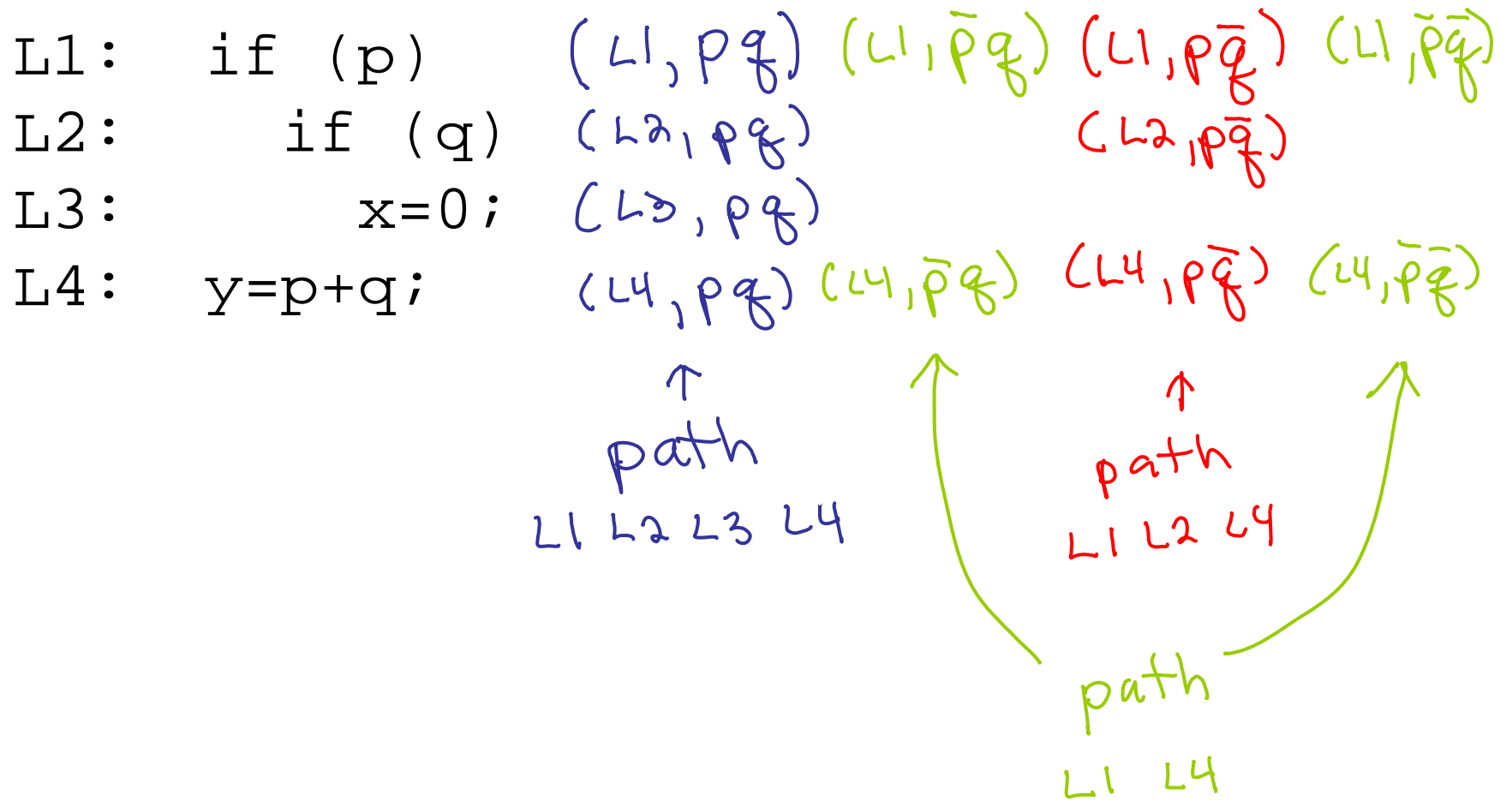
x → -3

Path Coverage does not imply PCT Coverage

L1:	if (p)	(L1, pq)	(L1, $\bar{p}q$)	(L1, $p\bar{q}$)	(L1, $\bar{p}\bar{q}$)
L2:	if (q)	(L2, pq)		(L2, $p\bar{q}$)	
L3:	x=0;	(L3, pq)			
L4:	y=p+q;	(L4, pq)	(L4, $\bar{p}q$)	(L4, $p\bar{q}$)	(L4, $\bar{p}\bar{q}$)

		↑		↑	
		path		path	
		L1 L2 L3 L4		L1 L2 L4	

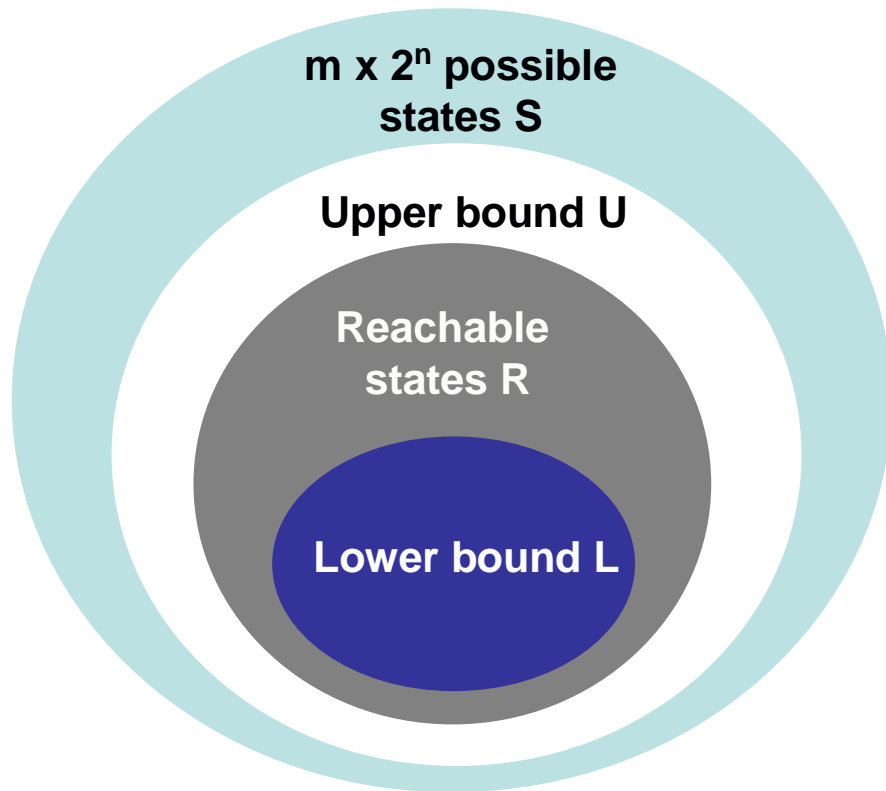
Path Coverage does not imply PCT Coverage



Denominator Problem

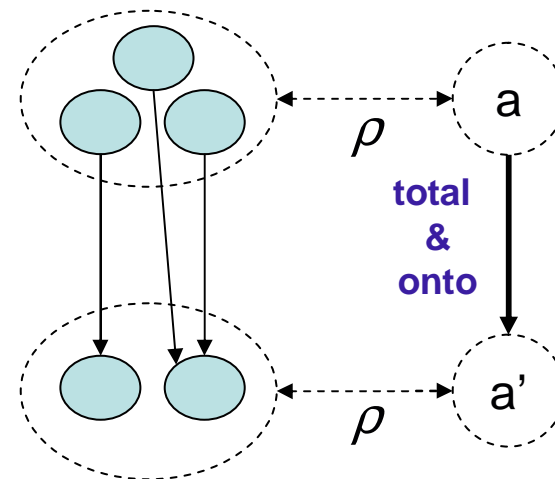
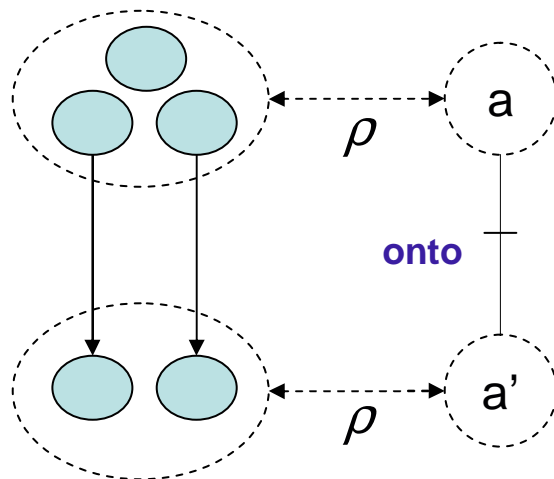
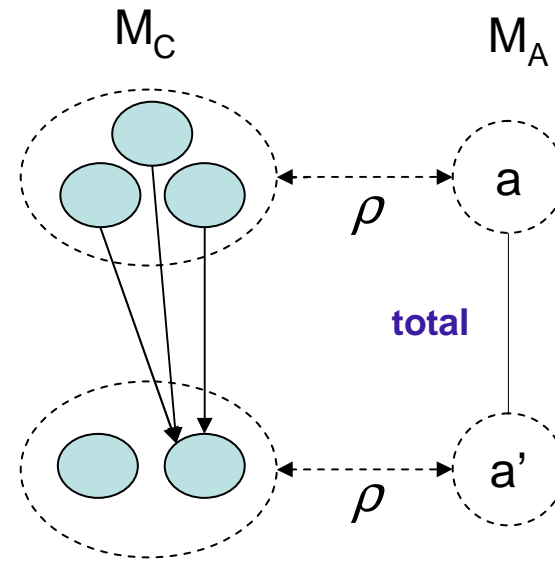
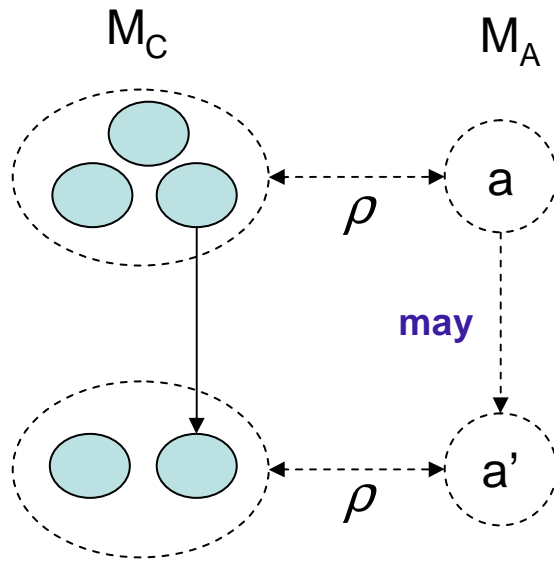
- Coverage metrics require a denominator
 - e.g. statements executed / total statements
- Easy to define for observable states
 - executed observable states / ($m \times 2^n$)
- But ($m \times 2^n$) is not a very good denominator!
 - most observable states will not be reachable
 - $R \lll S$

Upper and Lower Bounds

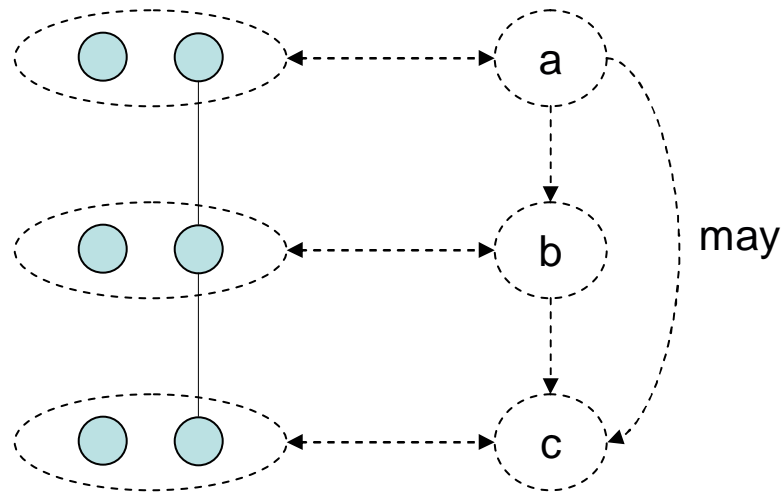
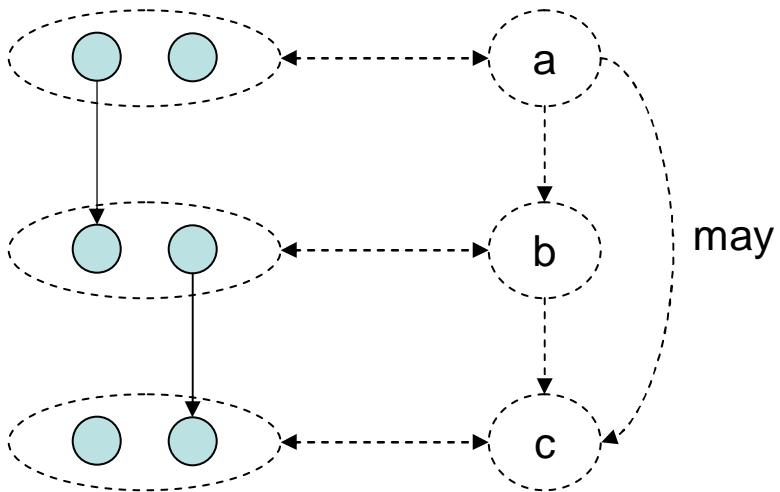


- Bound reachable observable states
 - modal transition systems and predicate abstraction
 - $|L| / |U|$ defines “goodness” of abstraction
- Test generation using lower bound L
- Refinement to increase $|L| / |U|$ ratio

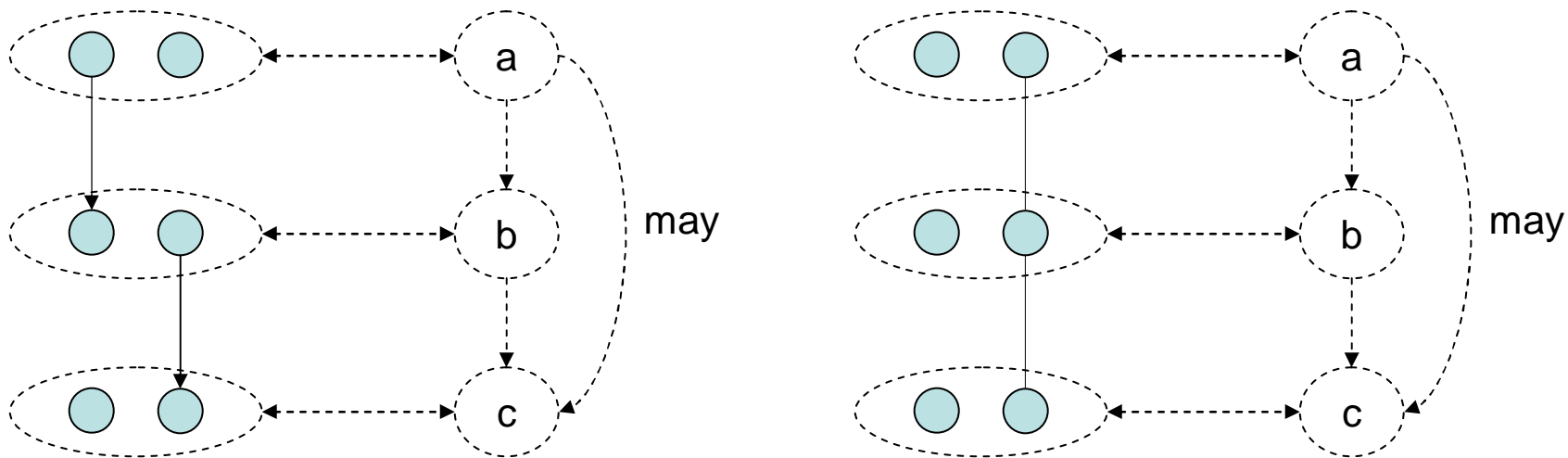
Abstraction Construction



Upper Bound: May-Reachability

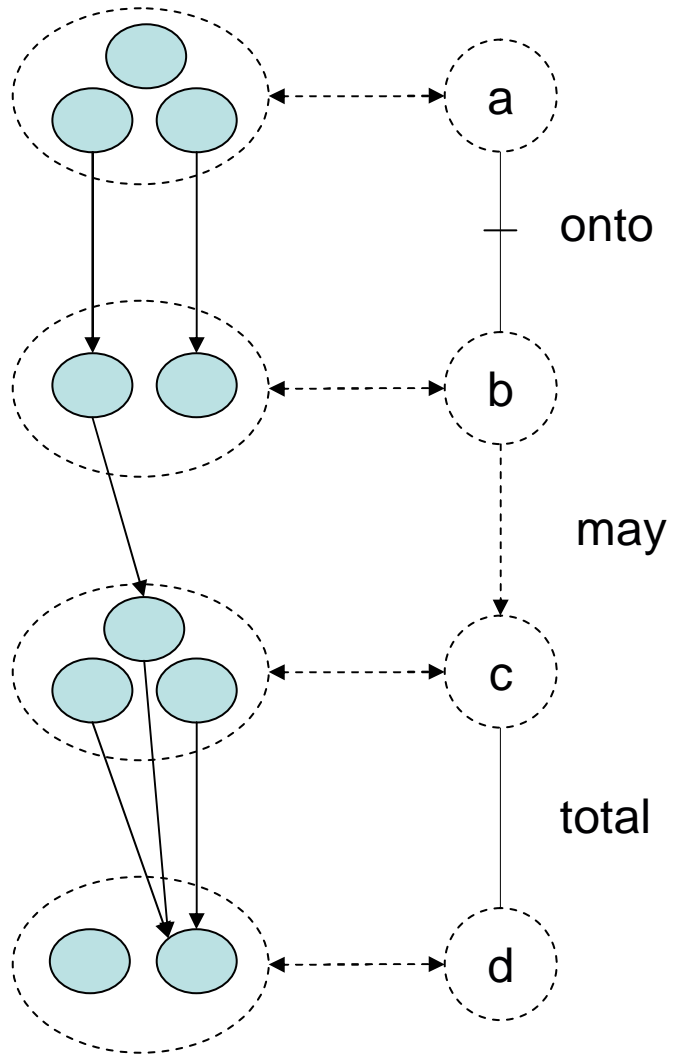


Upper Bound: May-Reachability

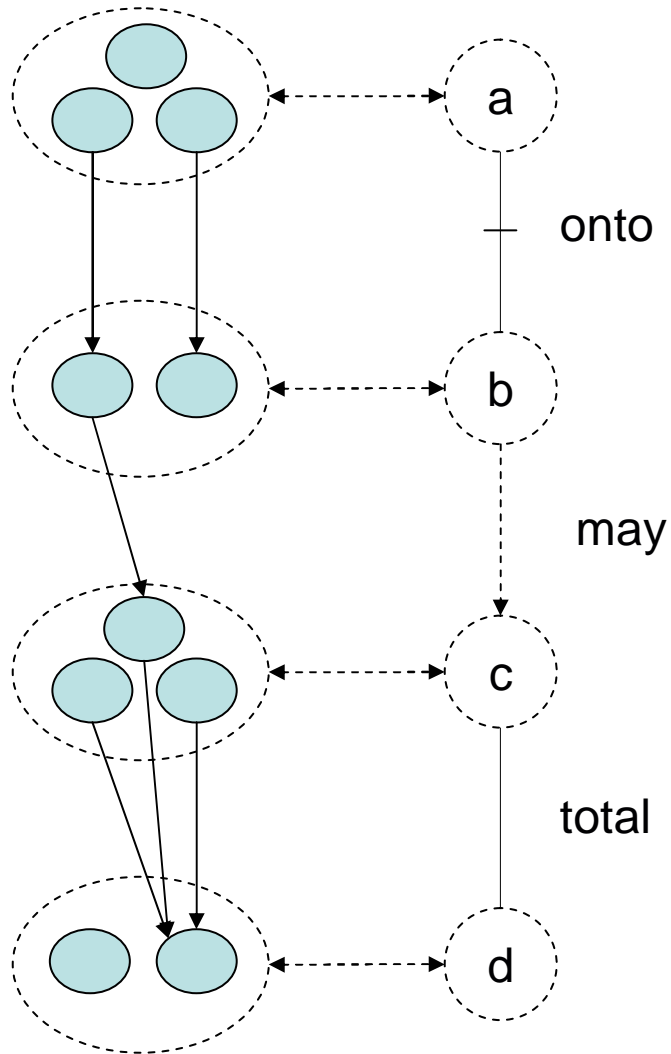


$$U = (\text{may})^*$$

Pessimistic Lower Bound



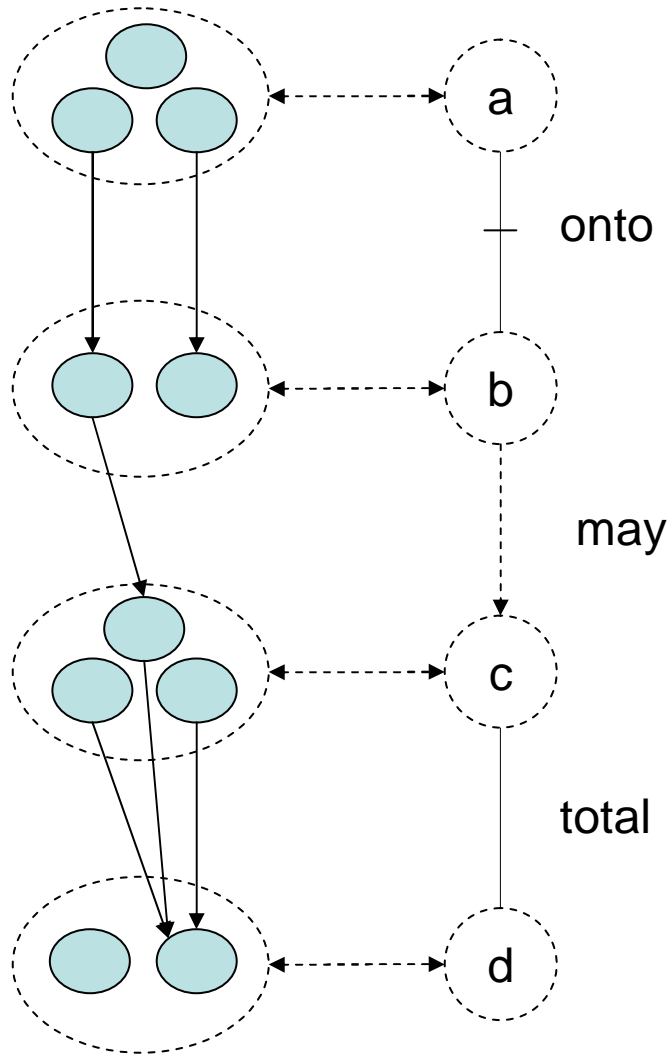
Pessimistic Lower Bound



$(\text{onto})^*$ is onto

$(\text{total})^*$ is total

Pessimistic Lower Bound



$(\text{onto})^*$ is onto

$(\text{total})^*$ is total

$$L_p = (\text{onto})^* \text{ may? } (\text{total})^*$$

Example

```
void partition(int a[]) {  
    assume(a.length()>2);  
    int pivot = a[0];  
    int lo = 1;  
    int hi = a.length()-1;  
    while (lo<=hi) {  
        while (a[lo]<=pivot)  
            lo++;  
        while (a[hi]>pivot)  
            hi--;  
        if (lo<hi)  
            swap(a,lo,hi);  
    }  
}
```

Observation Vector

[$lo < hi$, $lo \leq hi$, $a[lo] \leq pivot$, $a[hi] > pivot$]

- $lo < hi \Rightarrow lo \leq hi$
- $\neg lo < hi \wedge lo \leq hi \Rightarrow (a[lo] \leq pivot \wedge \neg a[hi] > pivot) \vee (\neg a[lo] \leq pivot \wedge a[hi] > pivot)$

Only 10/16 observations possible

```
void partition(int a[]) {
    assume(a.length()>2);
    int pivot = a[0];
    int lo = 1;
    int hi = a.length()-1;
```

13 labels x 10
observations =
130 observable states

```
L0: while (lo<=hi) {
L1:     ;
L2:     while (a[lo]<=pivot) {
L3:         lo++;
L4:     ;}
L5:     while (a[hi]>pivot) {
L6:         hi--;
L7:     ;}
L8:     if (lo<hi) {
L9:         swap(a,lo,hi);
LA:     ;}
LB: ;}
LC: ;
}
```

But, program
constrains **reachable**
observable states
greatly.

Boolean Program

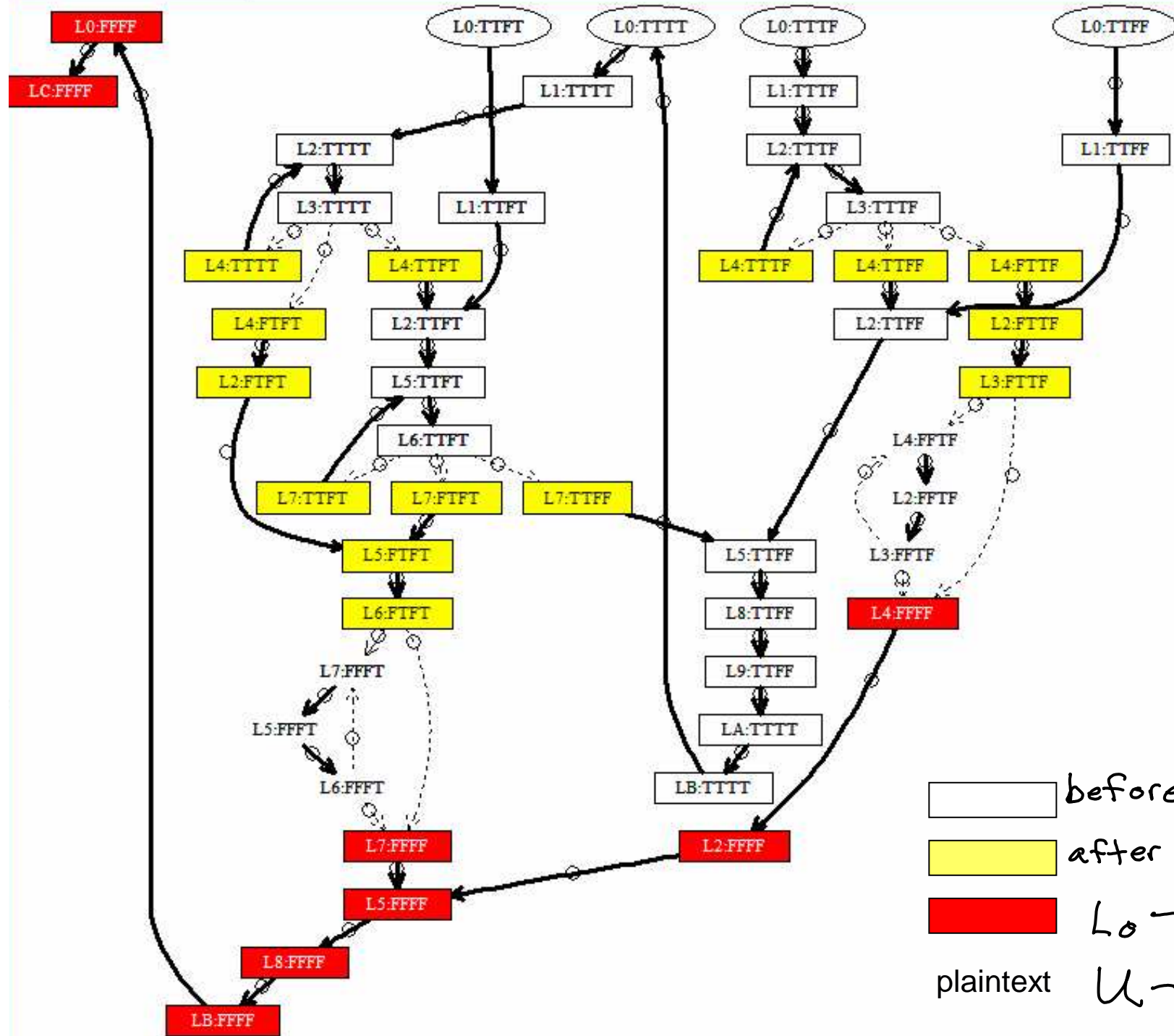
```
void partition() {
  decl lt, le, al, ah;
  enforce ( (lt=>le) &
            ((!lt&le)=>(al&!ah)|(!al&ah)) );
  lt,le,al,ah := T,T,*,*;
L0: while (le) {
L1:   ;
L2:   while (al) {
L3:     lt,le,al := (!lt ? F:*), lt, *;
L4:   ;}
L5:   while (ah) {
L6:     lt,le,ah := (!lt ? F:*), lt, *;
L7:   ;}
L8:   if (lt) {
L9:     al,ah := !ah,!al;
LA:   ;}
LB: ;}
LC: ;
}
```

State Space of Boolean Program

[$lo < hi$, $lo \leq hi$, $a[lo] \leq pivot$, $a[hi] > pivot$]

	TTTT	TTTF	FTTF	FFTF	TTFT	FTFT	FFFT	TTF	FFFF	FFTT
L0	x	x			x			x	x	
L1	x	x			x			x		
L2	x	x	x	x	x	x		x	x	
L3	x	x	x	x						
L4	x	x	x	x	x	x		x	x	
L5					x	x	x	x	x	
L6					x	x	x			
L7					x	x	x	x	x	
L8								x	x	
L9								x		
LA	x									
LB	x									x
LC										x

Upper Bound = 49 states



before may
 after may
 L0-Lp
 plaintext U-L0

Test Generation

- DFS of L_p generates covering set of paths
- Symbolically execute paths to generate tests
- Run program on tests to find errors and compute coverage of observable states

Generated Inputs

```
void partition(int a[]) {
  assume(a.length()>2);
  int pivot = a[0];
  int lo = 1;
  int hi = a.length()-1;

  L0: while (lo<=hi) {
  L1:   ;
  L2:   while (a[lo]<=pivot) {
  L3:     lo++;
  L4:   ;}
  L5:   while (a[hi]>pivot) {
  L6:     hi--;
  L7:   ;}
  L8:   if (lo<hi) {
  L9:     swap(a,lo,hi);
  LA:  ;}
  LB: ;}
  LC: ;
}
```

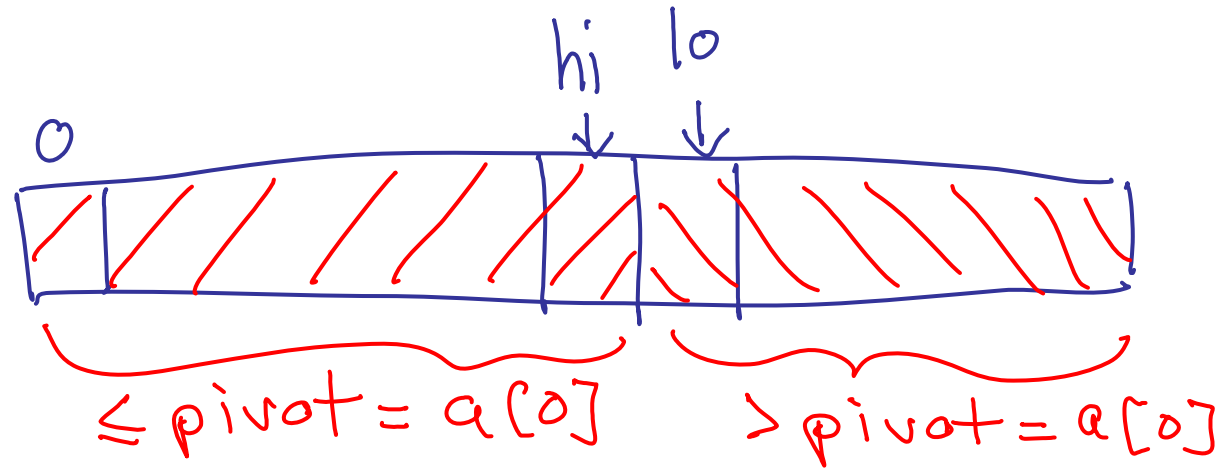
Array bounds violations

(L0:TTTT,L4:FTFT)	{ 0,-8,1 }
(L0:TTTT,L4:TTFT)	{ 0,-8,2,1 }
(L0:TTTT,L4:TTTT)	{ 0,-8,-8,1 }
(L0:TTTF,L4:TTFF)	{ 1,-7,3,0 }
(L0:TTTF,L4:FTTF)	{ 0,-7,-8 }
(L0:TTTF,L4:TTTF)	{ 1,-7,-7,0 }
(L0:TTFT,L7:TTFF)	{ 0,2,-8,1 }
(L0:TTFT,L7:FTFT)	{ 0,1,2 }
(L0:TTFT,L7:TTFT)	{ 0,3,1,2 }
(L0:TTFF,L0:TTTT)	{ 1,2,-1,0 }

Results

- Buggy partition function
 - U=49, L=43, Tested=42
- Fixed partition function
 - U=56, L=37, Tested=43
- What about the remaining 13 states?

Refinement



Existing predicates do not precisely track boundary condition

$$lo < hi, lo \leq hi, a[lo] \leq \text{pivot}, a[hi] > \text{pivot}$$

New Observation Vector

[$lo < hi$, $lo \leq hi$, $lo = hi + 1$,
 $a[lo] \leq pivot$, $a[hi] > pivot$,
 $a[lo - 1] \leq pivot$, $a[hi + 1] > pivot$
]

Only 48/128 observations possible

For this set of predicates, $L_p = U$

Conclusions

- PCT coverage
 - new form of state-based coverage
 - similar to path coverage but finite
- Upper and lower bounds
 - computed using predicate abstraction and modal transitions
 - use lower bound to guide test generation
 - refine bounds