Static Program Analysis

Mooly Sagiv

Static Analysis

- Automatically infer sound invariants from the code
- Prove the absence of certain program errors
- Prove user-defined assertions
- Report bugs before the program is executed

Simple Correct C code

```
main() {
    int i = 0,  *p =NULL, a[100];
    for (i=0 ;  i <100, i++) {
        a[i] = i;
        p = malloc(1, sizeof(int));
        *p = i;
        free(p);
        p = NULL;
       }
```

Simple Correct C code

```
main() {
    int i = 0,  *p=NULL, a[100];
    for (i=0 ;  i <100, i++) {
        { 0 <= i < 100}
        a[i] = i;
        { p == NULL:}
        p = malloc(1, sizeof(int));
        { alloc(p) }
        *p = i;
        {alloc(p)}
        free(p);
         {!alloc(p)}
        p = NULL;
         {p==NULL}
       }
```

Simple Incorrect C code

```
main() {
    int i = 0,  *p=NULL, a[100], j;
    for (i=0 ;  i <j , i++) {
      { 0 <= i < j}
      a[i] = i;
      p = malloc(1, sizeof(int));
      { alloc(p) }
     p = malloc(1, sizeof(int));
      { alloc(p) }
      free(p);
      free(p);
     }
```

Sound (Incomplete) Static Analysis

- It is undecidable to prove interesting program properties
- Focus on sound program analysis
  - When the compiler reports that the program is correct it is indeed correct for every run
  - The compiler may report spurious (false alarms)

A Simple False Alarm

```
int i, *p=NULL;
…
if (i >=5) {
    p = malloc(1, sizeof(int));
}
…
if (i >=5) {
   *p = 8;
}
…
if (i >=5) {
   free(p);
}
```

A Complicated False Alarm

```
int i, *p=NULL;
…
if (foo(i)) {
    p = malloc(1, sizeof(int));
}
…
if (bar(i )) {
   *p = 8;
}
…
if (zoo(i)) {
   free(p);
}
```

Foundation of Static Analysis

- Static analysis can be viewed as interpreting the program over an "abstract domain"
- Execute the program over larger set of execution paths
- Guarantee sound results
  - Whenever the analysis reports that an invariant holds it indeed hold

Even/Odd Abstract Interpretation

- Determine if an integer variable is even or odd at a given program point

Example Program

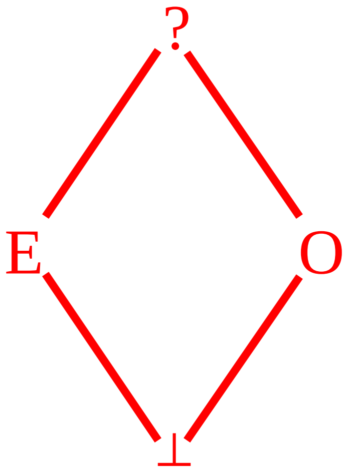
```
    /* x=? */

while  (x !=1)   do {  /* x=? */
          if   (x %2) == 0
    /* x=E */             { x := x / 2; }        /* x=? */
                    else
    /* x=O */           { x := x * 3 + 1;
                          assert (x %2 ==0); }  /* x=E */
}

/* x=O*/
```

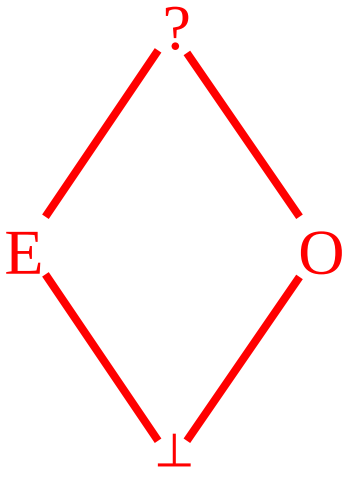
A Lattice of values

⊔	⊥	E	O	?
⊥				
E				
O				
?				



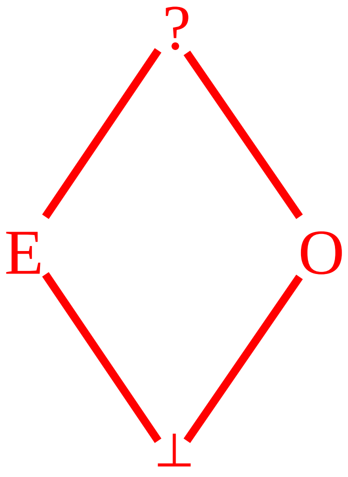
A Lattice of values

⊔	⊥	E	O	?
⊥	⊥	E	O	?
E	E	E	?	?
O	O	?	O	?
?	?	?	?	?



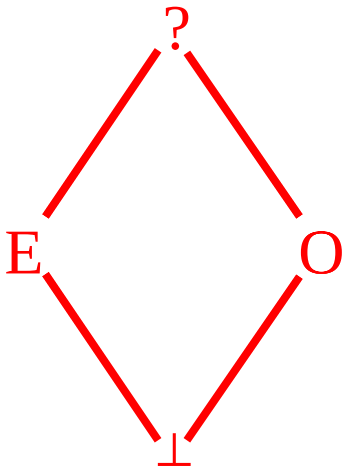
A Lattice of values

⊓	⊥	E	O	?
⊥				
E				
O				
?				

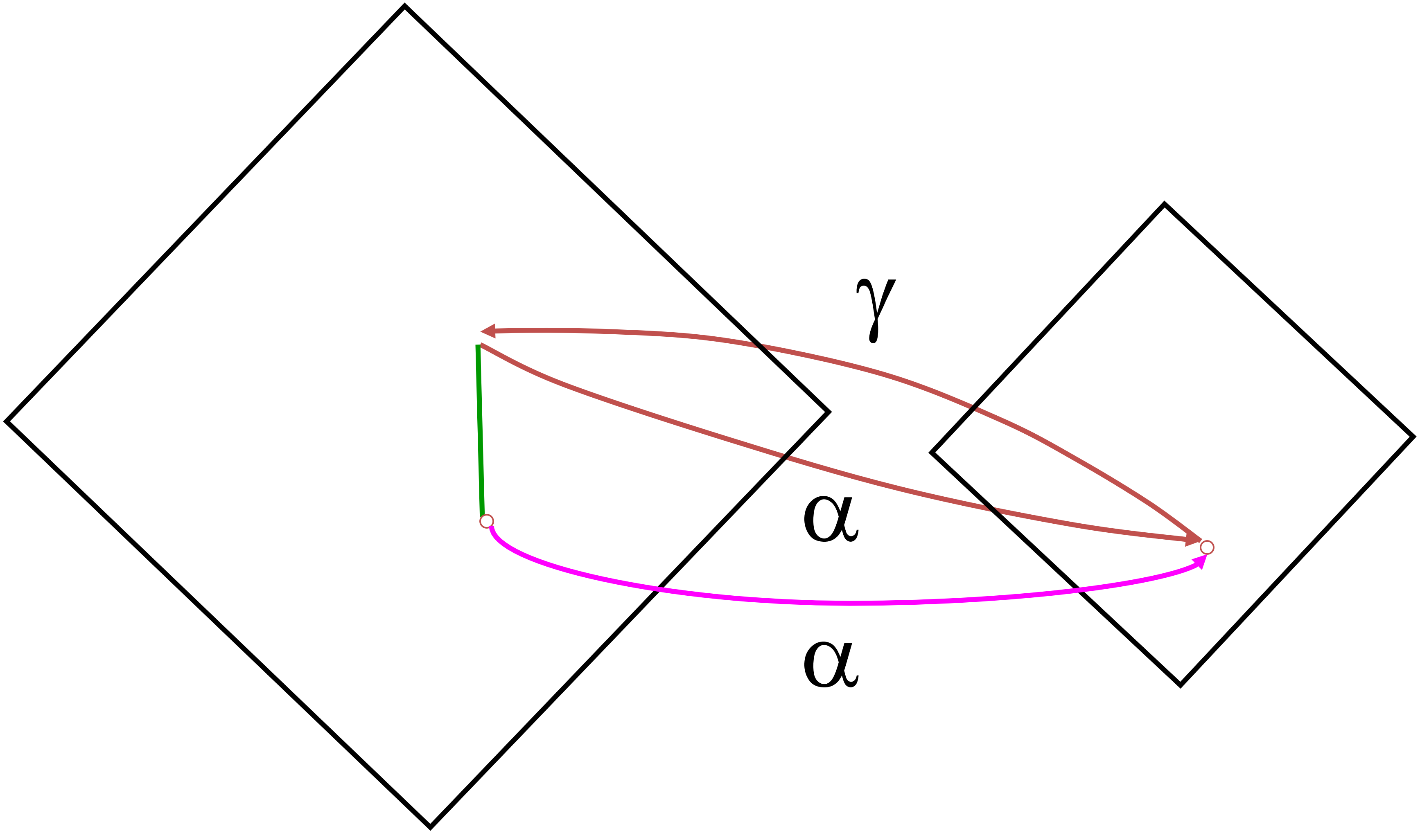


A Lattice of values

⊓	⊥	E	O	?
⊥	⊥	⊥	⊥	⊥
E	⊥	E	⊥	E
O	⊥	⊥	O	O
?	⊥	E	O	?

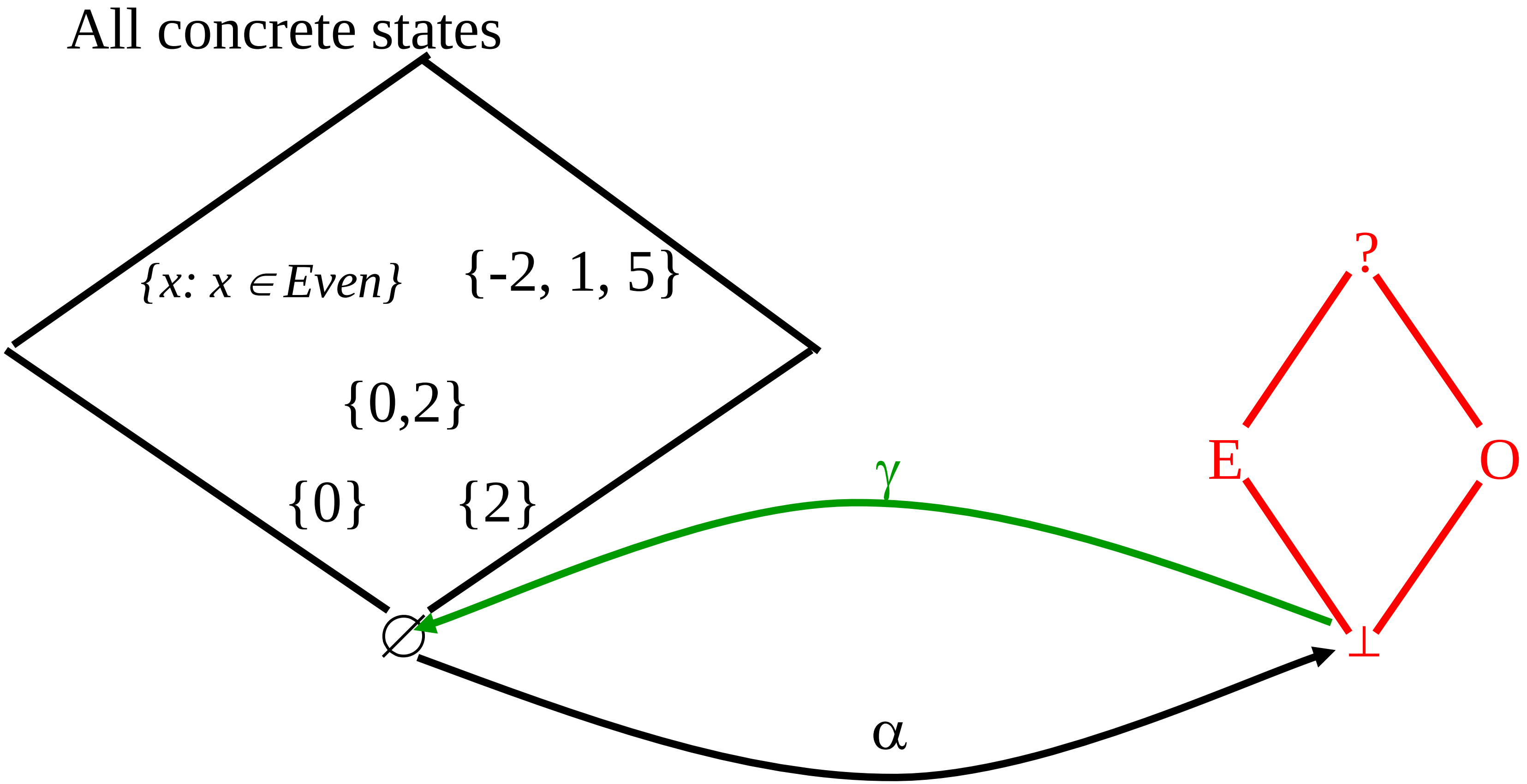


Abstract Interpretation

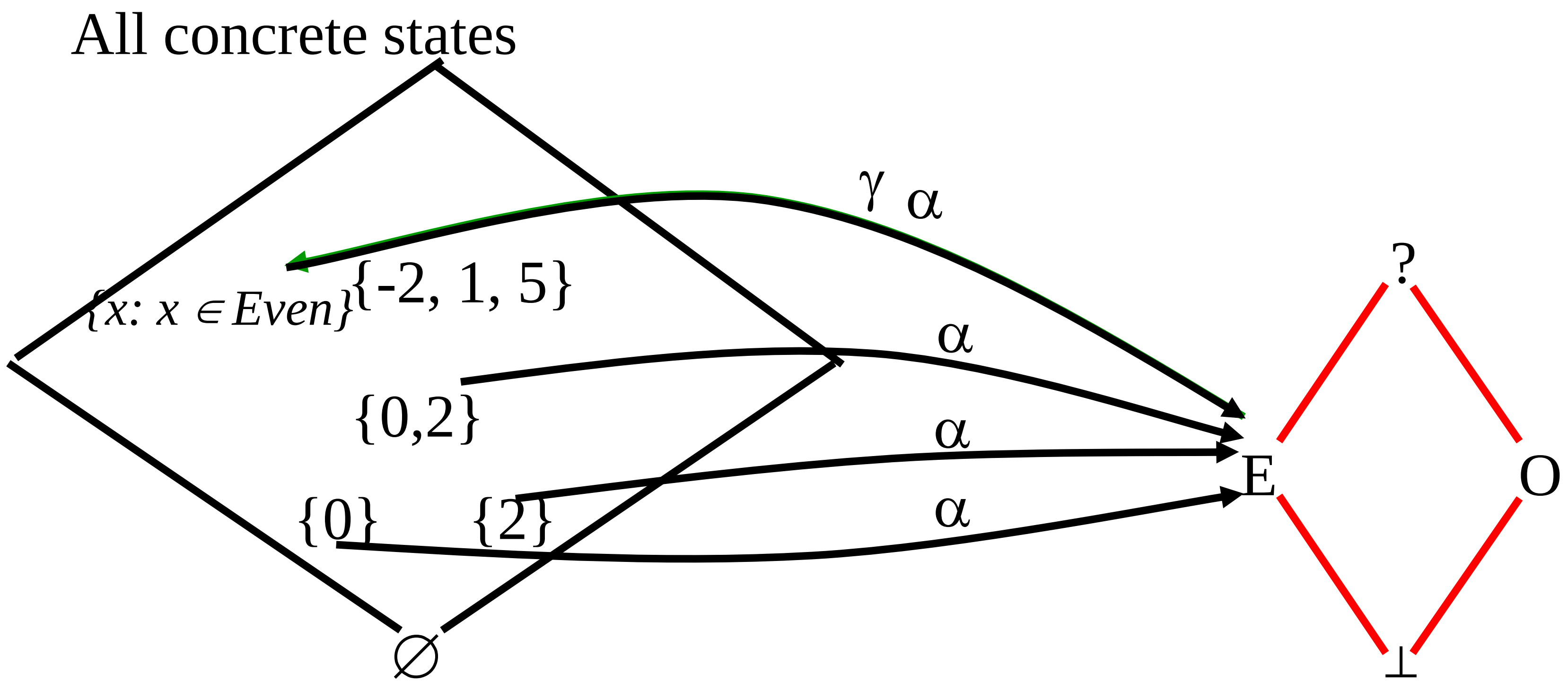


Sets of stores $\xrightarrow{\alpha}$ *Descriptors of sets of stores*

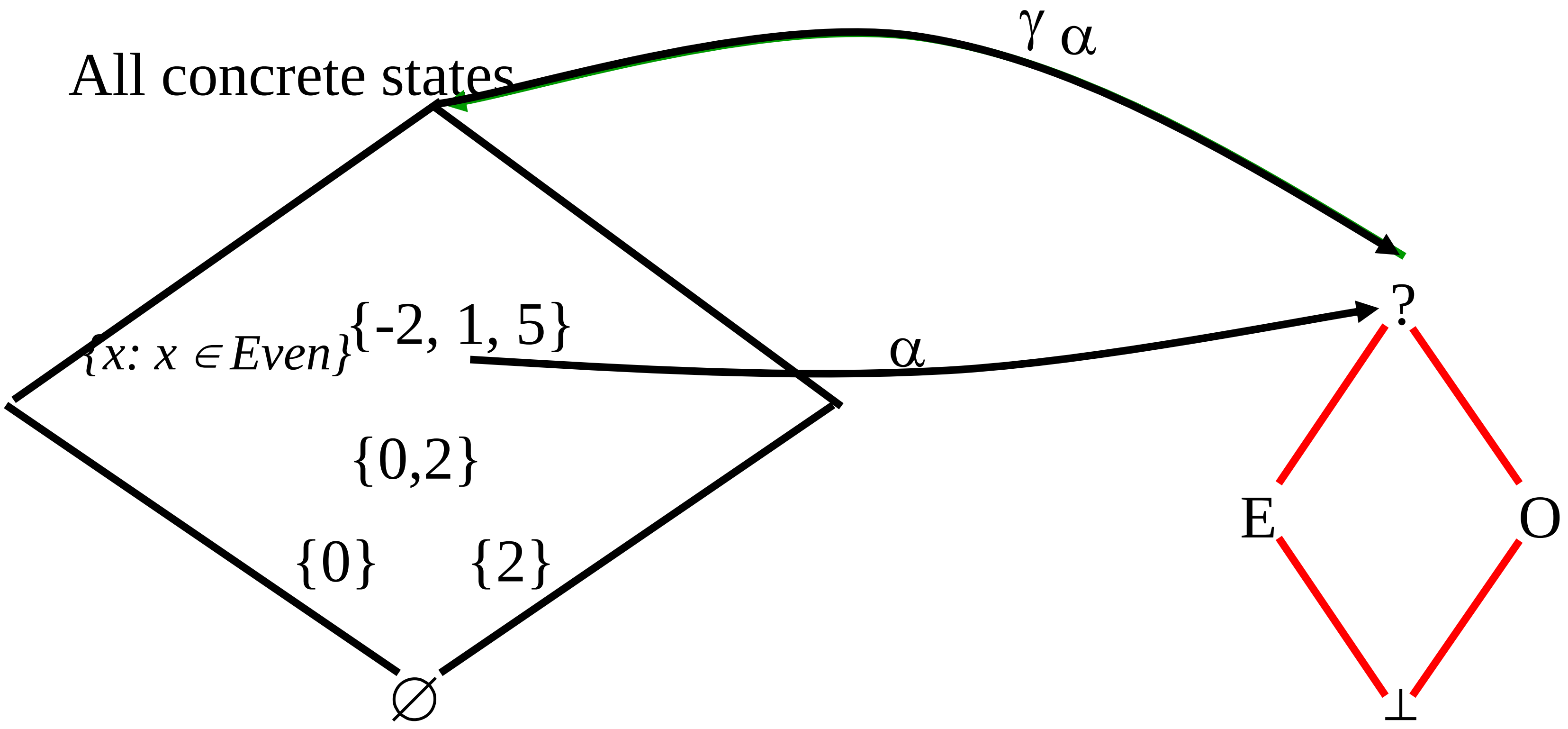
Odd/Even Abstract Interpretation



Odd/Even Abstract Interpretation



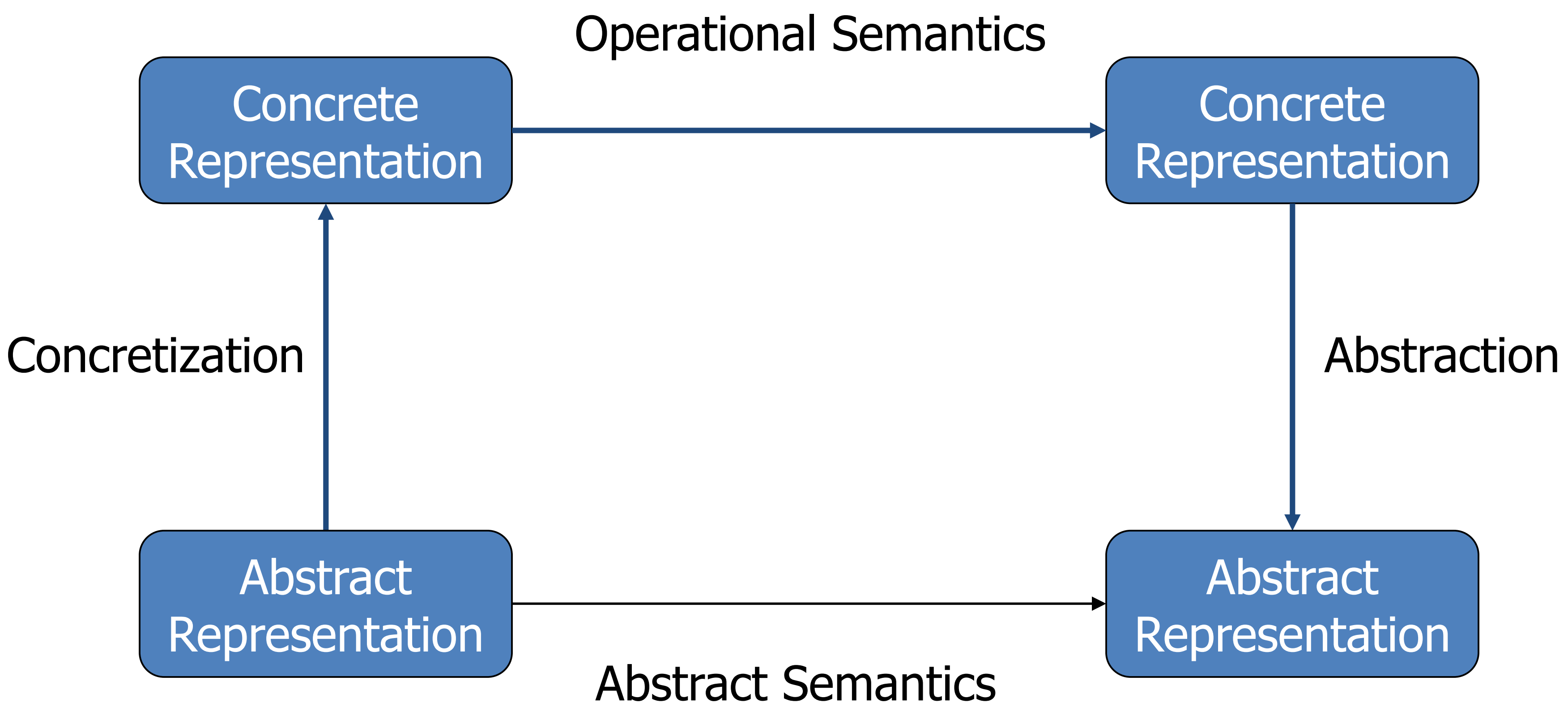
Odd/Even Abstract Interpretation



Example Program

```
while  (x !=1)   do {
          if   (x %2) == 0
                              { x := x / 2; }
                    else
       /* x=O */   { x := x * 3 + 1;        /* x=E */
                              assert (x %2 ==0); }
}
```

(Best) Abstract Transformer



Runtime vs. Static Testing

	Runtime	Static Analysis
Effectiveness	Missed Errors	False alarms
		Locate rare errors
Cost	Proportional to program’s execution	Proportional to program’s size
	No need to efficiently handle rare cases	Can handle limited classes of programs and still be useful

Static Analysis Algorithms

- Generate a system of equations over the abstract values
- Iteratively compute the least solution to the system
- The solution is guaranteed to be sound
- The correctness of the invariants can be conservatively checked

Example Constant Propagation

- For every variable v and a program point pt, find if v has a constant value every time the program reaches pt

A Simple Example

```
l1: z = 3
l2: x = 1
while (l3: x > 0) {
        l4: if (x == 1) l5: y = 7
                l6: else y = z + 4
        l7:x = 3
}
l8:
```

label	X	y	z
l1	0	0	0
l2	0	0	3
l3	?	?	3
l4	?	?	3
l5	?	?	3
l6	?	?	3
l7	?	7	3
l8	?	?	3

A Lattice of Values (per variable)

?

… -2 -1 0 1 2 …

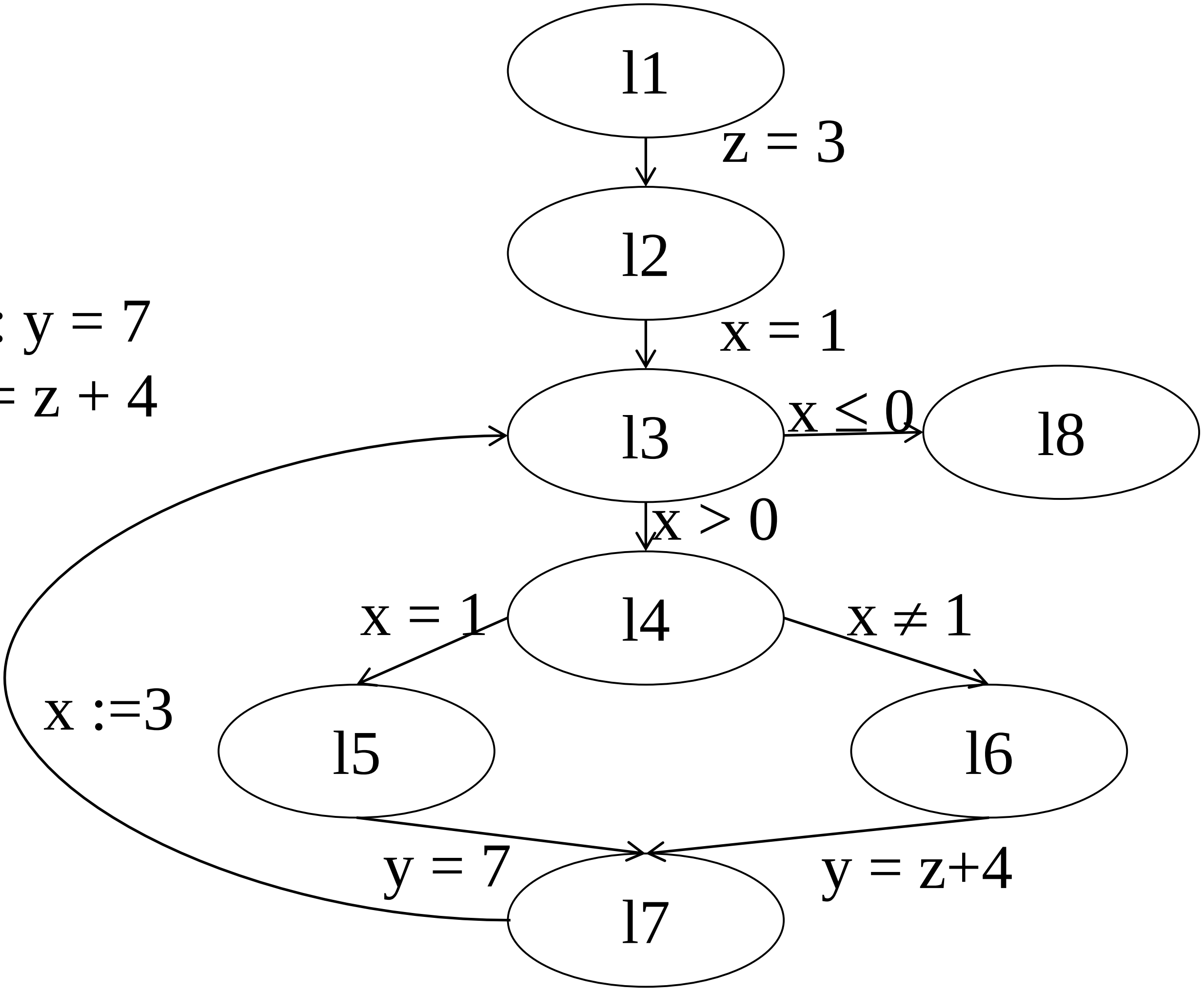
⊥

Computing Constants

- Construct a control flow graph (CFG)
- Associate transfer functions with control flow graph edges
- Define a system of equations
- Compute the simultaneous least fixed point via Chaotic iterations
- The solution is unique
  - But order of evaluation may affect the number of iterations

A Simple Example

l1: z = 3
l2: x = 1
while (l3: x > 0) {
    l4: if (x == 1) l5: y = 7
        l6: else y = z + 4
    l7:x = 3
}
l8:



A Simple Example: System of Equations

$DF[l1] = [x \mapsto 0, z \mapsto 0]$

$DF[2] = DF[l1][\![z \mapsto 3]\!]^{\#}$

$DF[l4] = DF[l3][\![x>0]\!]^{\#} \sqcup DF[l7][\![x:=3]\!]^{\#}$

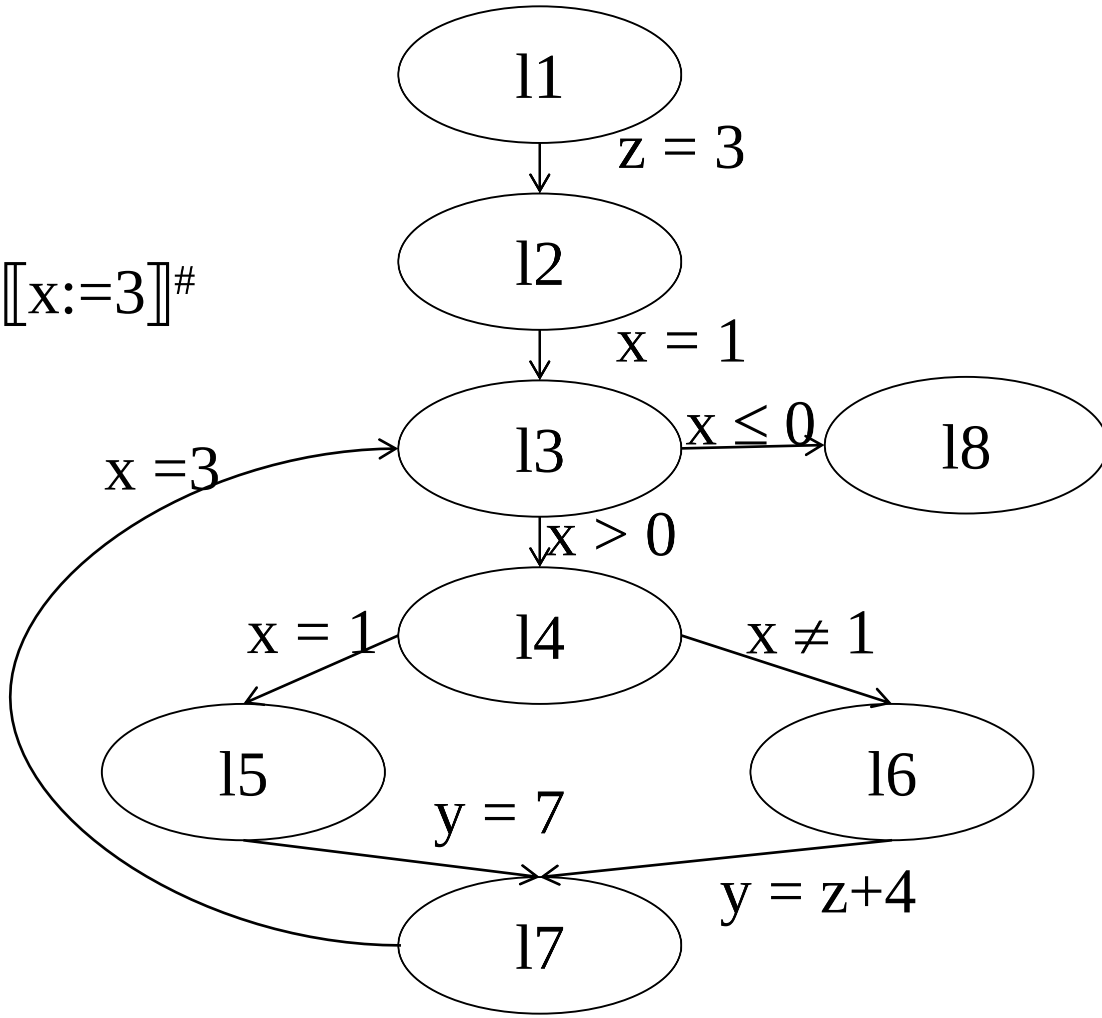
$DF[l3] = DF[l2][\![x \mapsto 1]\!]^{\#}$

$DF[l5] = DF[l4][\![x \neq 1]\!]^{\#}$

$DF[l6] = DF[l4][\![x=1]\!]^{\#}$

$DF[l7] = DF[l5][\![y=7]\!]^{\#} \sqcup DF[l6][\![y=z+4]\!]^{\#}$

$DF[l8] = DF[l]$



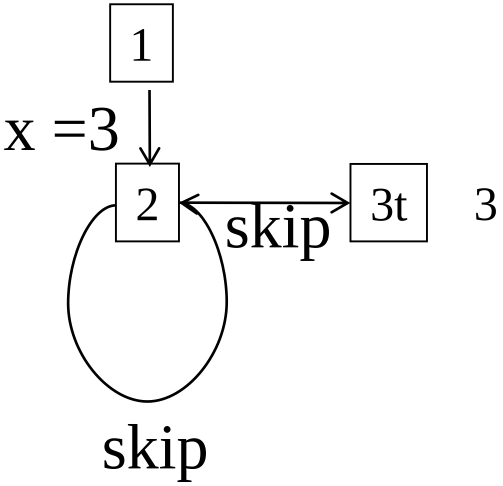
Chaotic Iterations

```
Chaotic(G(V, E): Graph, s: Node, L: Lattice, ι: L, f: E →(L →L) ){
  for each v in V to n do df_entry[v] := ⊥
  df[s] = ι
  WL = {s}
  while (WL ≠  ∅ )  do
    select and remove an element u ∈ WL
    for each v, such that. (u, v) ∈E do
          temp = f(e)(df_entry[u])
          new := df_entry(v)⊔ temp
          if (new ≠ df_entry[v]) then
                df_entry[v] := new;
                WL := WL ∪{v}
```

Solving the system of equations

- Every solution to the system of equations is sound
- Non-solution may not be sound
- Compute a simultaneous least solution iteratively from below
  - Intermediate solutions are not sound

Example Constant Propagation



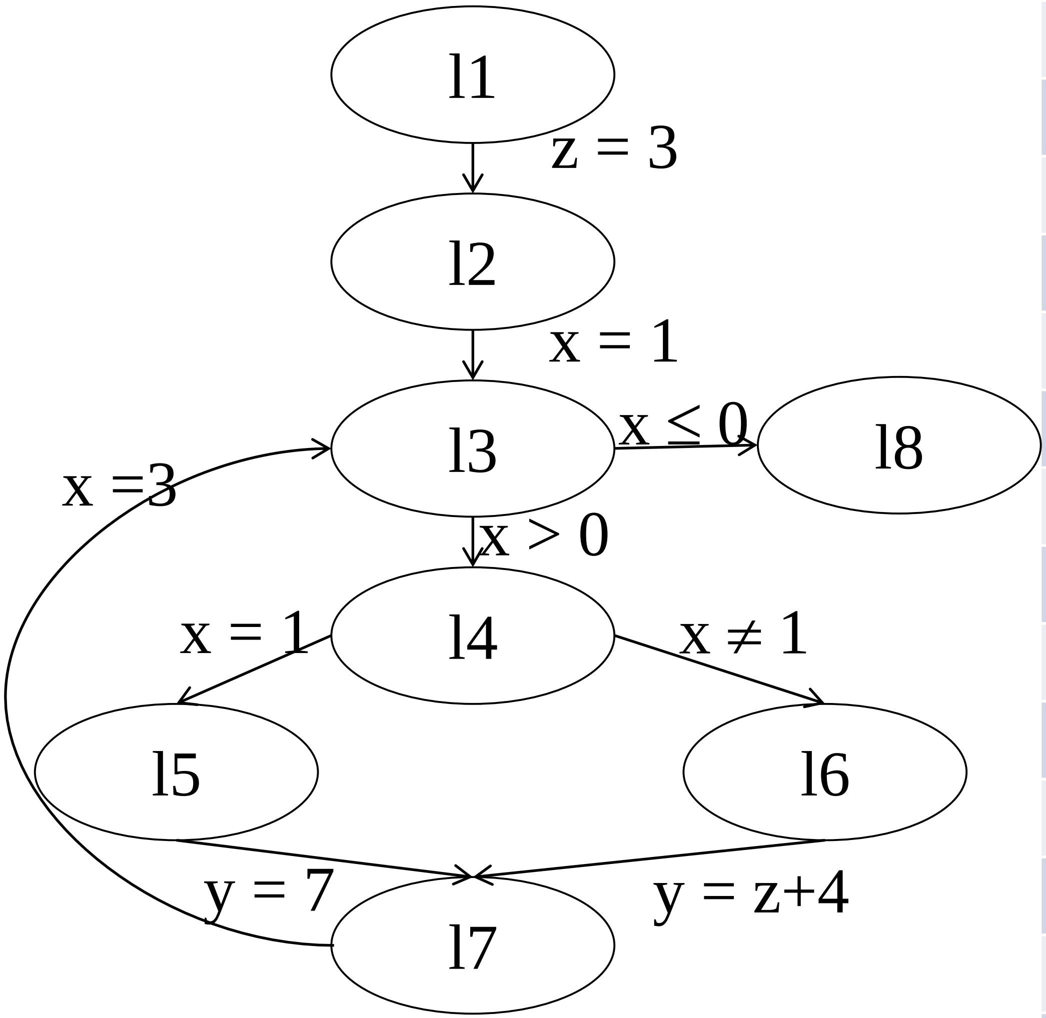
$$DF(1) = [x \mapsto 0]$$

$$DF(2) = DF(1)[x \mapsto 3] \sqcup DF(2)$$

$$DF(3) = DF(2)$$

DF[1]	DF[2]	DF[3]
$[x \mapsto 0]$	$[x \mapsto 3]$	$[x \mapsto 3]$
$[x \mapsto 0]$	$[x \mapsto ?]$	$[x \mapsto ?]$
$[x \mapsto 7]$	$[x \mapsto 9]$	$[x \mapsto 7]$
$[x \mapsto ?]$	$[x \mapsto 3]$	$[x \mapsto 3]$

A Simple Example: Chaotic Iterations



N	DF[N]	WL
		{1}
1	[x↦0, y↦0, z ↦0]	{2}
2	[x↦0, y↦0, z ↦3]	{3}
3	[x↦1, y↦0, z ↦3]	{4, 8}
4	[x↦1, y↦0, z ↦3]	{5, 6, 8}
5	[x↦1, y↦0, z ↦3]	{6, 7, 8}
6		{7, 8}
7	[x↦1, y↦7, z ↦3]	{3, 8}
3	[x↦⊤, y ↦⊤, z ↦3]	{4, 8}
4	[x↦⊤, y↦⊤, z ↦3]	{5, 6, 8}
5	[x↦1, y↦⊤, z ↦3]	{6, 7, 8}
6	[x↦⊤, y↦⊤, z ↦3]	{7, 8}
7	[x↦⊤, y↦7, z ↦3]	{4, 8}
4		{8}
8	[x↦⊤, y ↦⊤, z ↦3]	{}

When do we loose precision

- Dynamic vs. Static values
- Correlated branches
- Locality of transformers (Join over all path)

```
if (…)
     x = 5; y= 7;
 else
     x= 7; y = 5;
  l: z= x + y;
```

- Initial value

Example Interval Analysis

- Find a lower and an upper bound of the value of a single variable
- Can be generalized to multiple variables

Simple Correct C code

```
main() {
    int i = 0, a[100];
    { [-minint, maxint] }
    for (i=0 ;  i <100, i++) {
        {[0, 99]}
        a[i] = i;
         {[0, 99]}
                }
{[100, 100]}
```

The Power of Interval Analysis

```
int f(x) {
 {[minint , maxint]}
  if (x > 100) {
       {[101, maxint]}
       return x -10 ;
       {[91, maxint-10];}
     }
     else {
     {[minint, 100] }
     return f(f(x+11))
     { [91, 91]}
     }
```

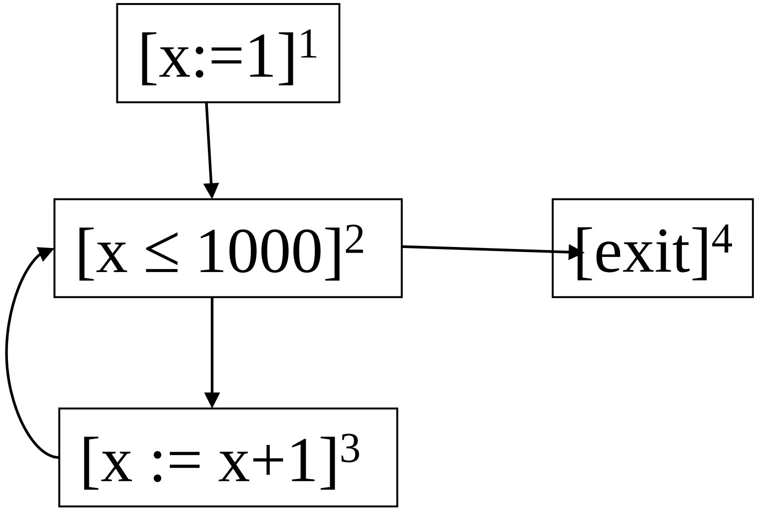
Example Program
Interval Analysis

$[x := 1]^1$;

while $[x \leq 1000]^2$

do

$\quad [x := x + 1;]^3$



Abstract Interpretation of Atomic Statements

$$[\![\text{skip}]\!]^{\#}[l, u] = [l, u]$$

$$[\![x := 1]\!]^{\#}[l, u] = [1, 1]$$

$$[\![x := x + 1]\!]^{\#}[l, u] = [l, u] + [1, 1] = [l + 1, u + 1]$$

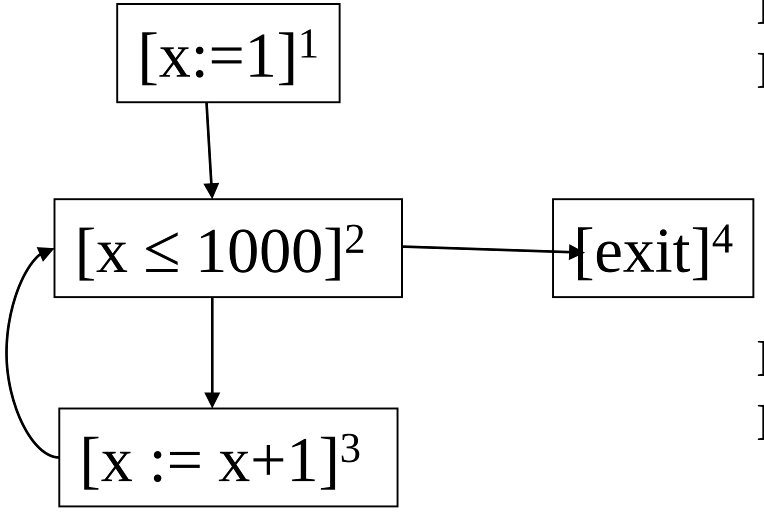
Equations Interval Analysis

[x := 1][1] ;

while [x $\leq$ 1000][2]

do

    [x := x + 1;][3]



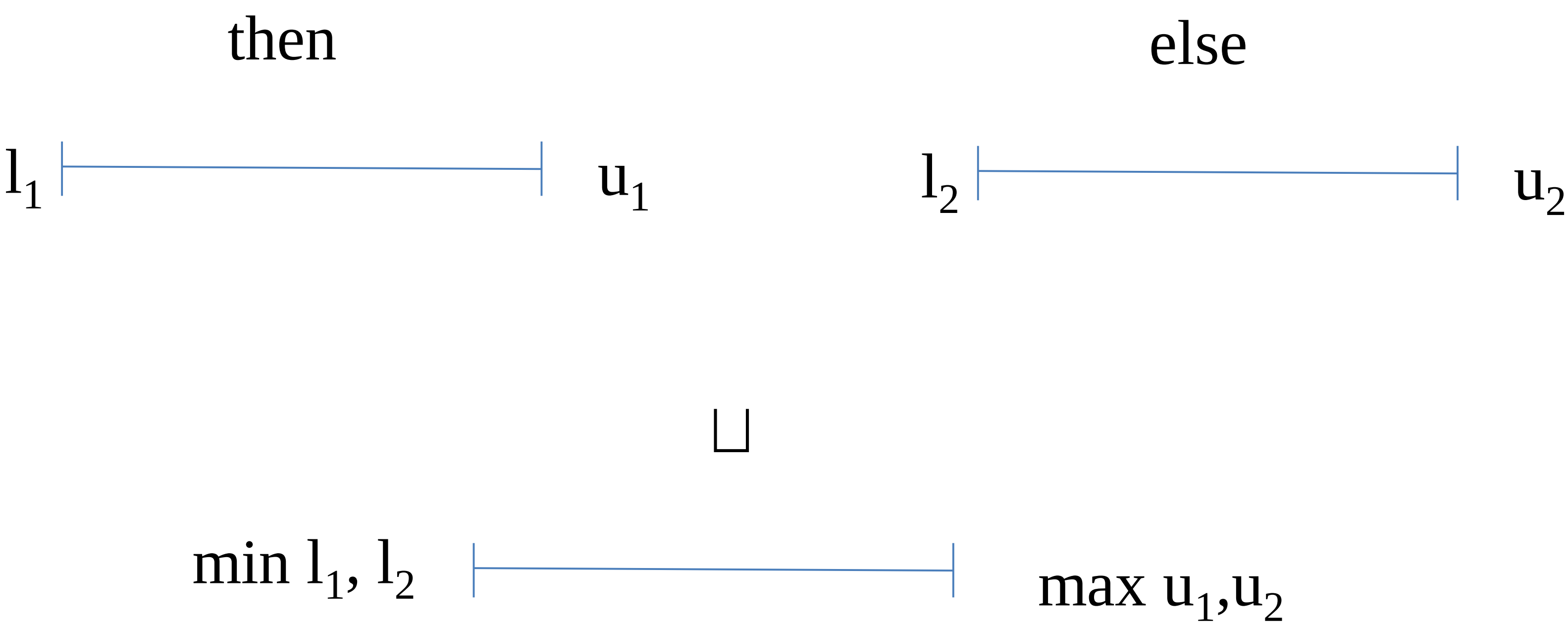
En(1) = [minint,maxint]
Ex(1) = [1,1]

In(2) = Ex(1) join Ex(3)
Ex(2) = In(2)

En(3) = Ex[2] meet [minint, 1000]
Ex(3) = In(3)+[1,1]

En(4) =Ex[2] meet [1001, maxint]
Ex(4) = In(4)

Abstract Interpretation of Joins



$$[l_1, u_1] \sqcup [l_2, u_2] = [\min(l_1, l_2), \max(u_1, u_2)]$$

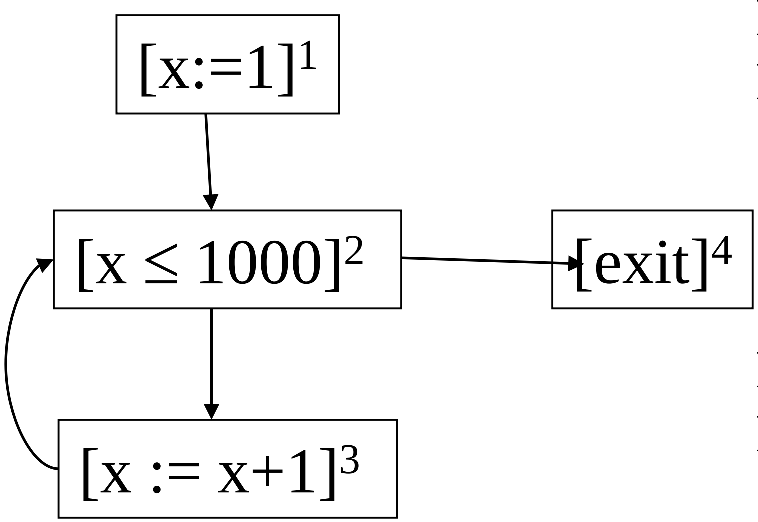
Equations Interval Analysis

$[x := 1]^1$;

while $[x \leq 1000]^2$

do

$[x := x + 1;]^3$



$En(1) = [minint, maxint]$

$Ex(1) = [1,1]$

$En(2) = En(1) \sqcup En(3)$

$Ex(2) = En(2)$

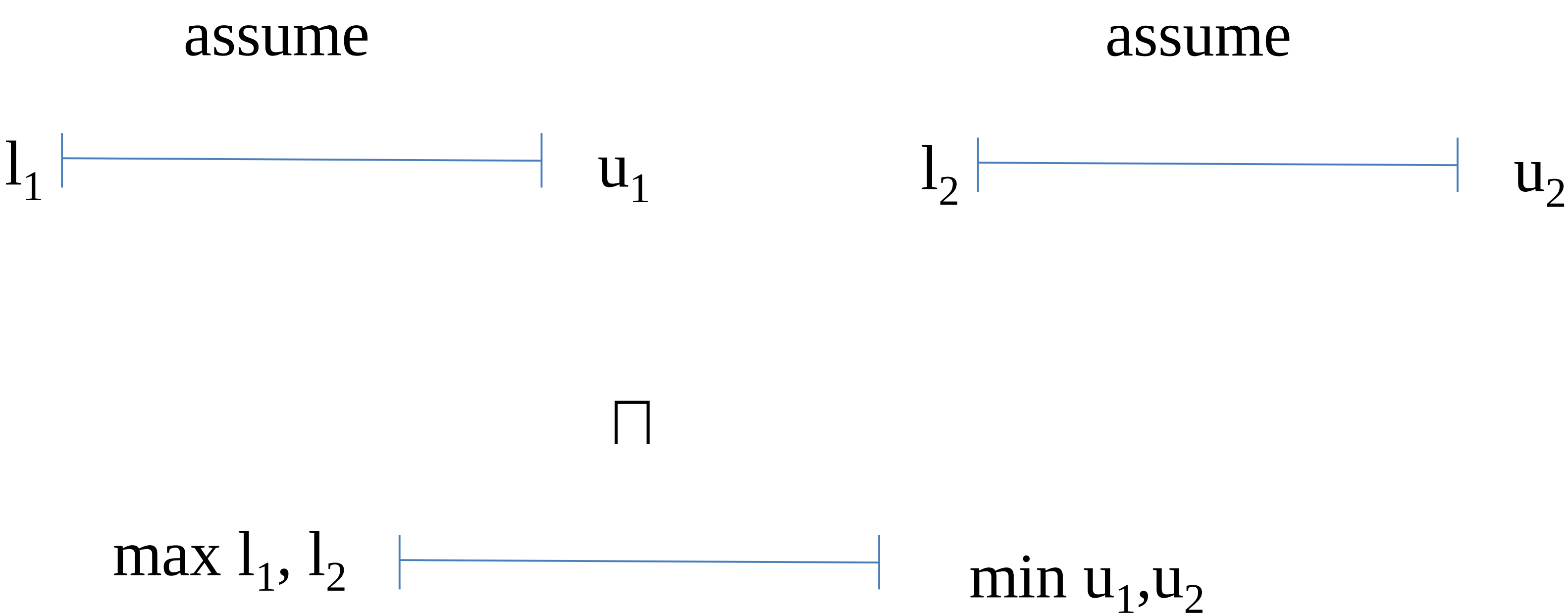
$En(3) =$

$Ex(3) = En(3)+[1,1]$

$En(4) =$

$Ex(4) = En(4)$

Abstract Interpretation of Meets



$[l_1, u_1] \sqcap [l_2, u_2] = [\max(l_1, l_2), \min(u_1, u_2)]$

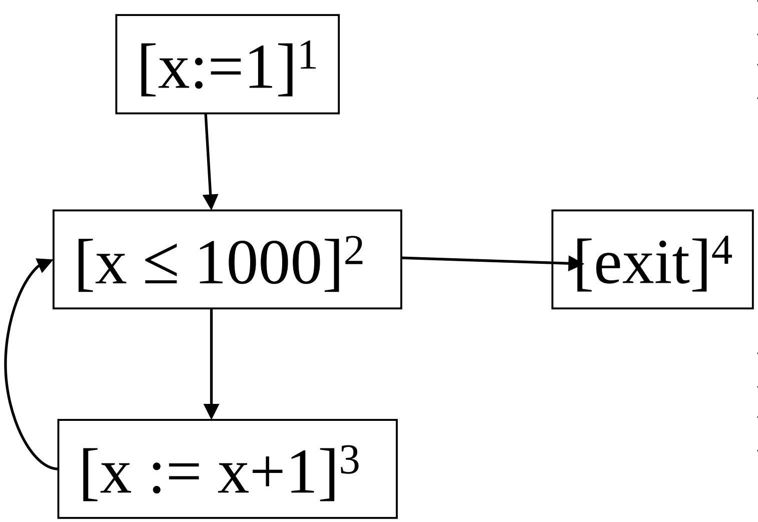
Equations Interval Analysis

[x := 1]^1 ;

while [x $\leq$ 1000]^2

do

    [x := x + 1;]^3



En(1) = [minint, maxint]
Ex(1) = [1,1]

En(2) = Ex(1) $\sqcup$ Ex(3)
Ex(2) = En(2)

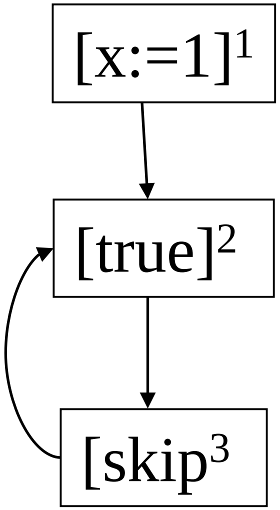
En(3) = Ex(2) $\sqcap$ [minint,1000]
Ex(3) = En(3)+[1,1]

En(4) = Ex(2) $\sqcap$ [1001,maxint]
Ex(4) = En(4)

Solving the Equations

- For programs with loops the equations have many solutions
- Every solution is sound
- Compute a minimal solution

An Example with Multiple Solutions



En(1) = [minint,maxint]
Ex(1) = [1,1]

In(2) = Ex(1) ⊔ Ex(3)
Ex(2) = In(2)

IntEntry(3) = IntExit(2)
IntExit(3) = IntEntry(3)

En[1]	Ex[1]	En[2]	Ex[2]	En[3]	Ex[3]	Comments
[-∞, ∞]	[1, 1]	[-∞, ∞]	[-∞, ∞]	[-∞, ∞]	[-∞, ∞]	Maximal
[-∞, ∞]	[1, 1]	[1, 1]	[1, 1]	[1, 1]	[1, 1]	Minimal
[-∞, ∞]	[1, 2]	[1, 2]	[1, 2]	[1, 2]	[1, 2]	Solution
[-∞, ∞]	⊥	[1, 1]	[1, 1]	[1, 2]	[1, 2]	Not a solution

Computing Minimal Solution

- Initialize the interval at the entry according to program semantics
- Initialize the rest of the intervals to empty
- Iterate until no more changes

Iterations Interval Analysis

IntEntry(1) = [minint,maxint]
IntExit(1) = [1,1]

IntEntry(2) = IntExit(1) $\sqcup$ IntExit(3)
IntExit(2) = IntEntry(2)

IntEntry(3) = IntExit(2) $\sqcap$ [minint,1000]
IntExit(3) = IntEntry(3)+[1,1]

IntEntry(4) = IntExit(2) $\sqcap$ [1001,maxint]
IntExit(4) = IntEntry(4)

En[1]	Ex[1]	En[2]	Ex[2]	En[3]	Ex[3]	In[4]	Ex[4]
[-∞, ∞]	⊥	⊥	⊥	⊥	⊥	⊥	⊥
	[1, 1]						
		[1,1]					
			[1,1]				
				[1, 1]			
					[2, 2]		
		[1, 2]					

Widening

- Accelerate the convergence of the iterative procedure by jumping to a more conservative solution
- Heuristic in nature
- But simple to implement

Widening for Interval Analysis

- $\bot \nabla [c, d] = [c, d]$
- $[a, b] \nabla [c, d] = [$

    if $a \leq c$

        then a

        else $-\infty$,

    if $b \geq d$

        then b

        else ∞

    ]

Iterations with widening

IntEntry(1) = [minint,maxint]
IntExit(1) = [1,1]

IntEntry(2) = IntEntry(2) ∇(IntExit(1) $\sqcup$ IntExit(3))
IntExit(2) = IntEntry(2)

IntEntry(3) = IntExit(2) $\sqcap$ [minint,1000]
IntExit(3) = IntEntry(3)+[1,1]

IntEntry(4) = IntExit(2) $\sqcap$ [1001,maxint]
IntExit(4) = IntEntry(4)

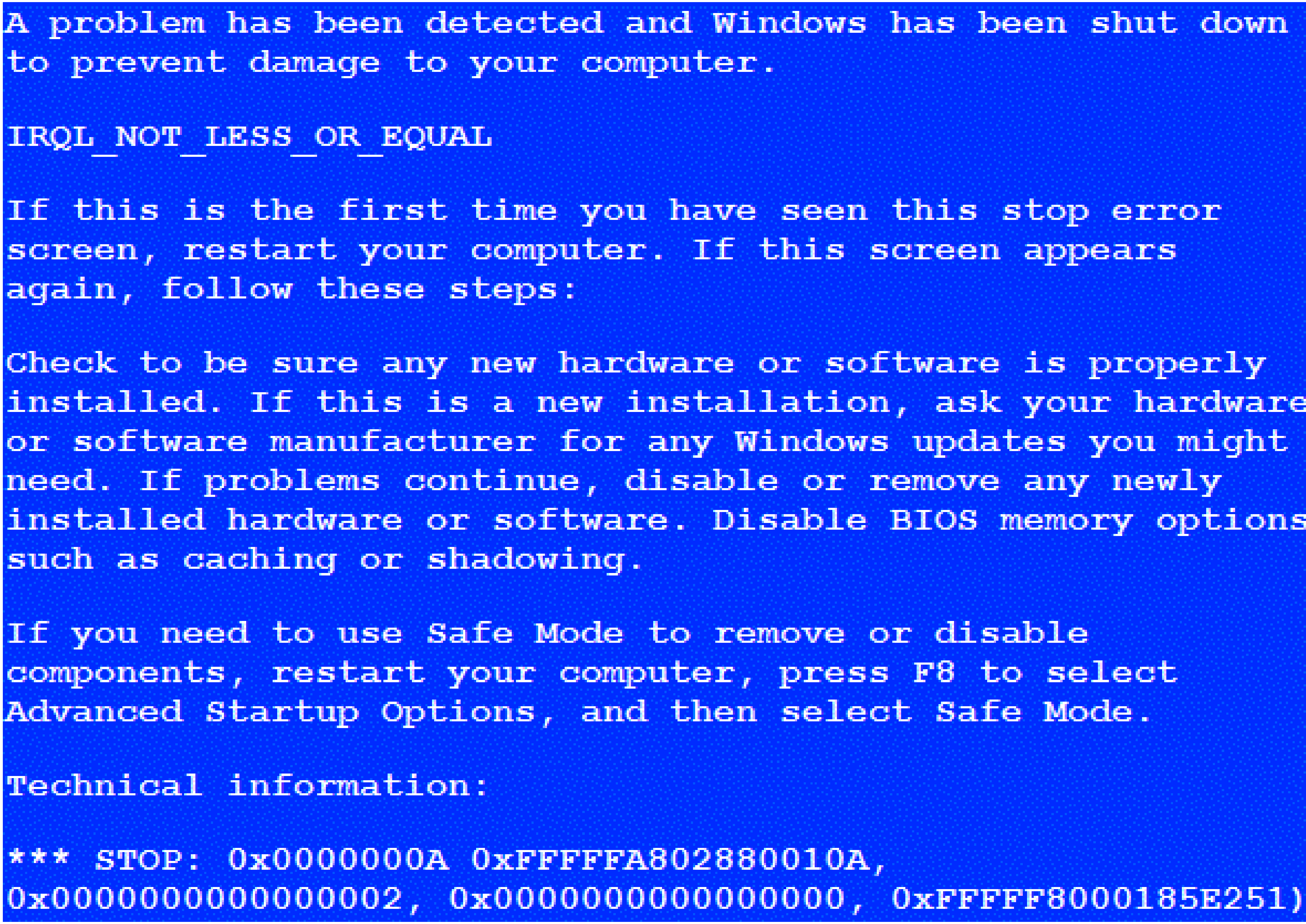
En[1]	Ex[1]	En[2]	Ex[2]	En[3]	Ex[3]	In[4]	Ex[4]
[-∞, ∞]	⊥	⊥	⊥	⊥	⊥	⊥	⊥
	[1, 1]						
		[1,1]					
			[1,1]				
				[1, 1]			
					[2, 2]		
		[1, ∞]					
			[1, ∞]				
				[1, 1000]			

More Static Analysis

- Liveness analysis
- Initialized variables
- Resolving virtual functions
- Pointer analysis
- Array bound checking

Some Success Stories

- The SLAM Static Driver Verification (MSR)
- Polyspace (INRIA, Mathworks)
- aiT (AbsInt)
- **The *Astrée* Static Analyzer**
- **LLVM Static Analysis**


A problem has been detected and Windows has been shut down to prevent damage to your computer.

IRQL_NOT_LESS_OR_EQUAL

If this is the first time you have seen this stop error screen, restart your computer. If this screen appears again, follow these steps:

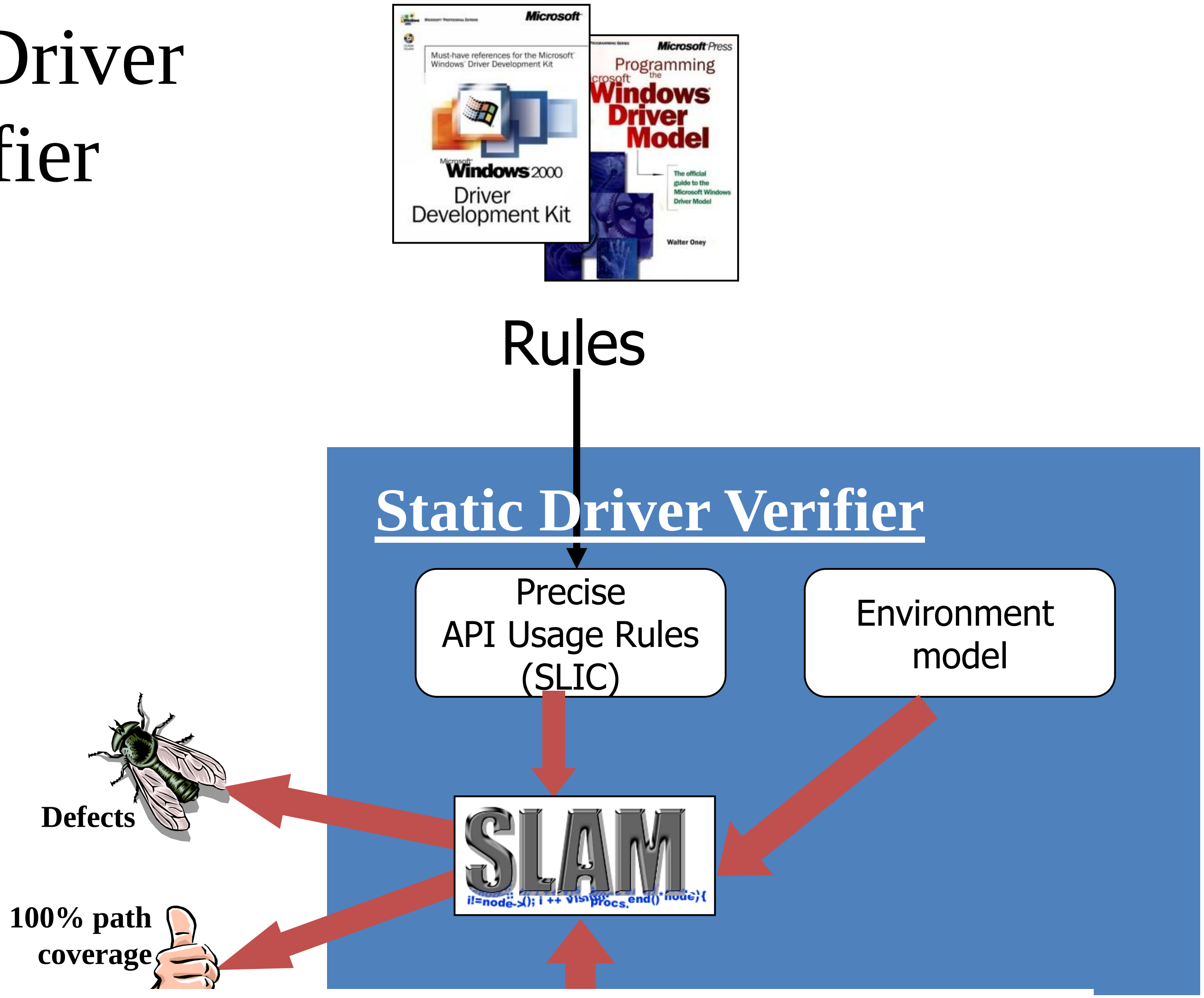
Check to be sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need. If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing.

If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

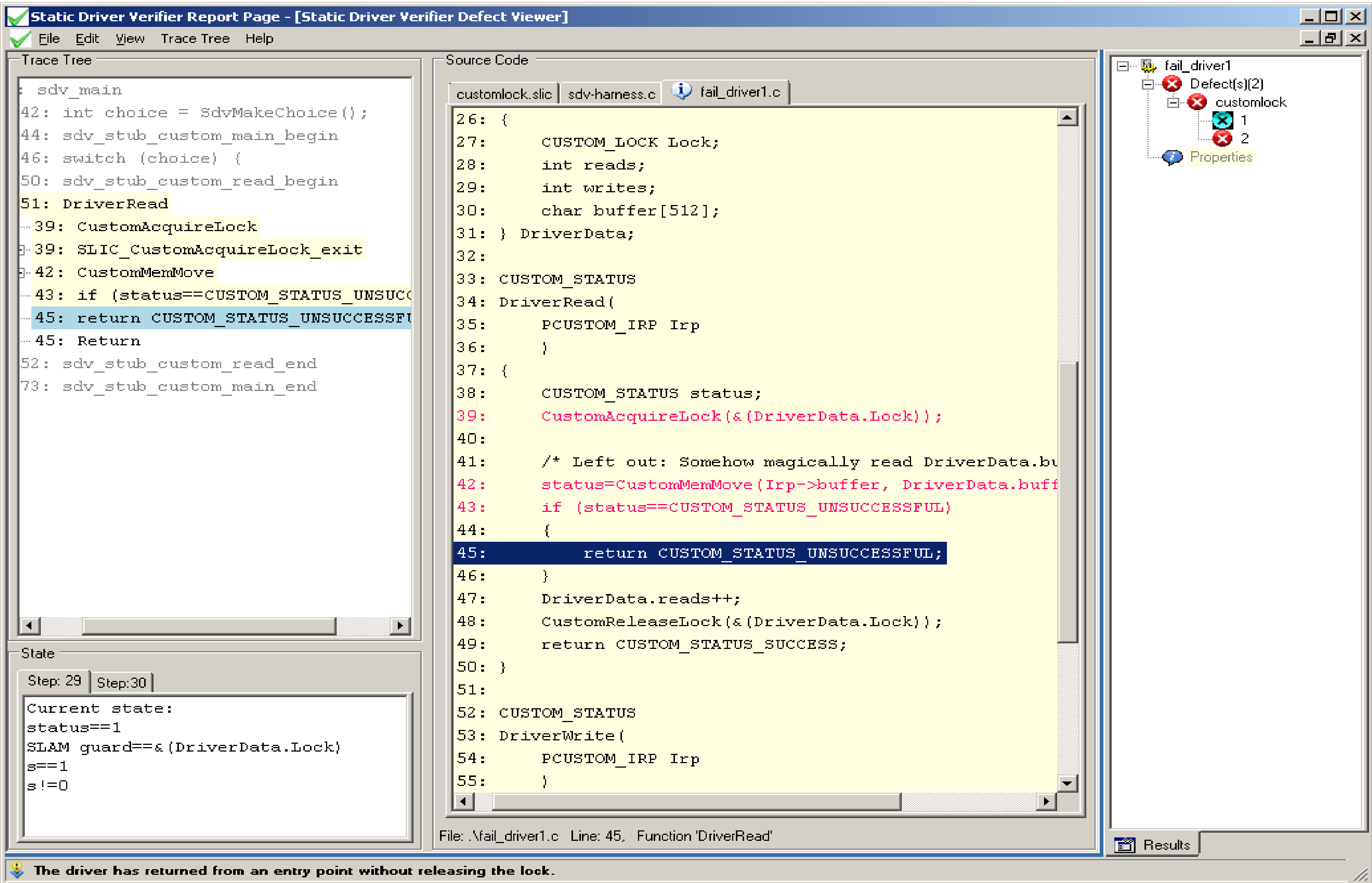
*** STOP: 0x0000000A 0xFFFFFA802880010A,
0x0000000000000002, 0x0000000000000000, 0xFFFFF8000185E251)

Static Driver Verifier



"Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability" Bill Gates

Example SLAM Application



Summary

- Static analysis is powerful
- Can locate rear bugs
- Challenges
  - Soundness
  - Scalability
    - Expensive algorithms
  - False alarms