Abstract Interpretation

Lecture (2)

Sriram Rajamani
Microsoft Research

Recall from yesterday….

Lattice

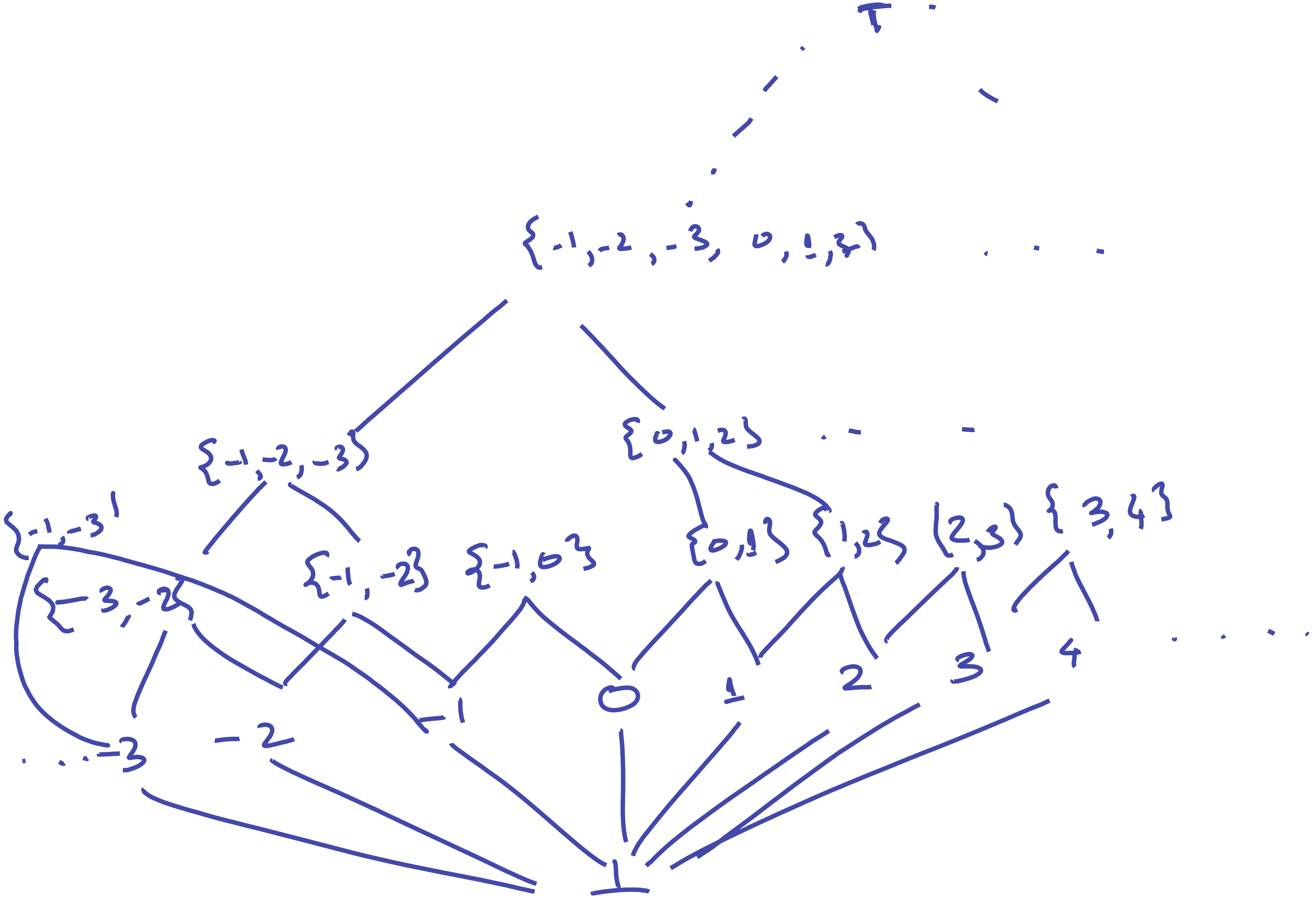
Let $\langle S, \leq \rangle$ be a po-set

$\langle S, \leq \rangle$ is a lattice if every non-empty subset of elements in S has a GLB and LUB

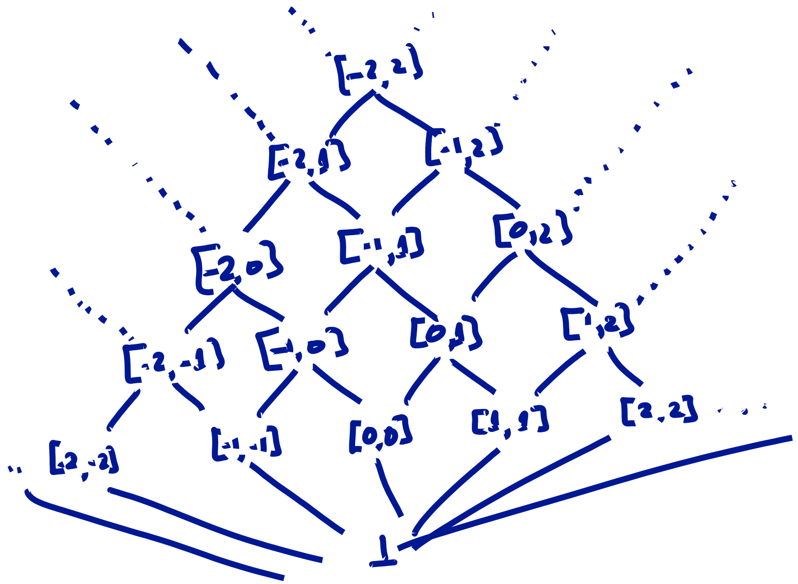
Join Semi-Lattice

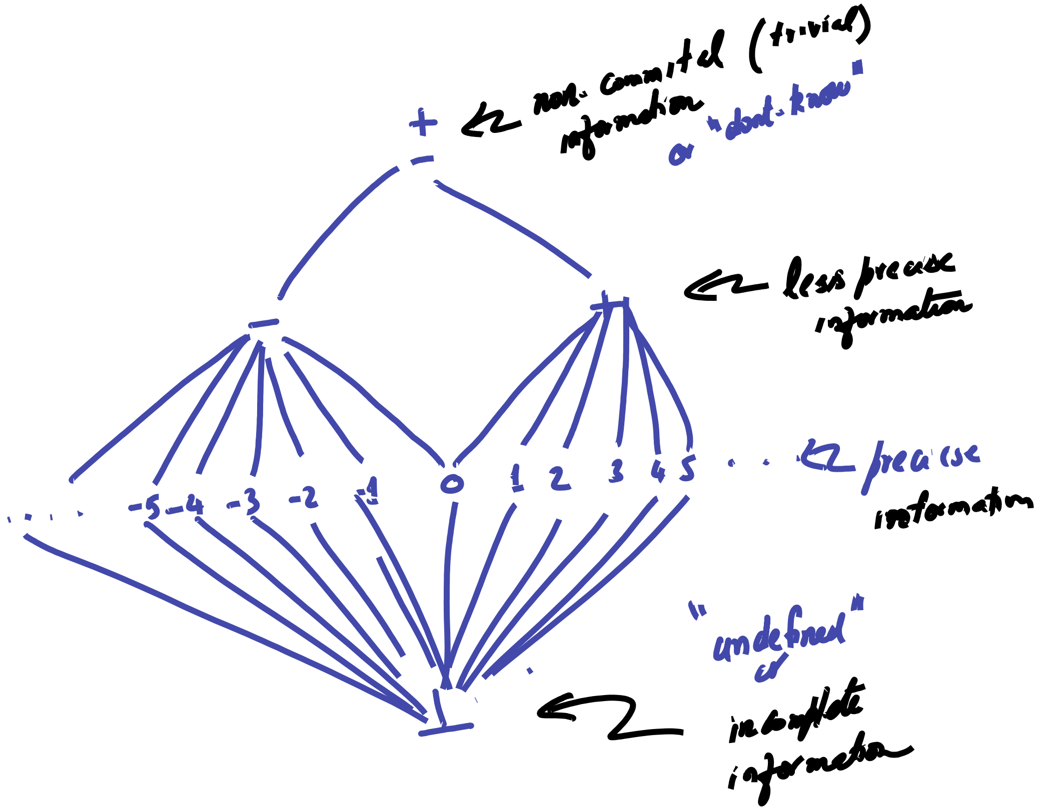
Let $\langle S, \leq \rangle$ be a po-set

$\langle S, \leq \rangle$ is a join semi-lattice if every non-empty subset of elements in S has a LUB in S



⊤
{-1,-2,-3, 0, 1, 2}
{-1,-2,-3}
{0,1,2}
{-1,-3}
{-3,-2}
{-1,-2}
{-1,0}
{0,1}
{1,2}
{2,3}
{3,4}
-1
0
1
2
3
4
-3
-2
⊥


[-2,2]
[-2,1]
[-1,2]
[-2,0]
[-1,1]
[0,2]
[-2,-1]
[-1,0]
[0,1]
[1,2]
[-2,-2]
[-1,-1]
[0,0]
[1,1]
[2,2]
⊥


⊤
non-committed (trivial) information
or "don't know"
less precise information
-5 -4 -3 -2 -1 0 1 2 3 4 5
precise information
⊥
"undefined" or incomplete information

Specifying an abstract interpretation

$$\langle D, \circ_D, \leq_D, \top_D, \bot_D, I_D \rangle \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} \langle A, \circ_A, \leq_A, \top_A, \bot_A, I_A \rangle$$

$$\alpha : D \longrightarrow A$$

$$\gamma : A \longrightarrow D$$

α, γ form a Galois connection iff

1. α, γ are order preserving

   $\forall d_1, d_2 \in D \quad d_1 \leq_D d_2 \Rightarrow \alpha(d_1) \leq_A \alpha(d_2)$

   $\forall a_1, a_2 \in A \quad a_1 \leq_A a_2 \Rightarrow \gamma(a_1) \leq_D \gamma(a_2)$

2. $\forall d \in D .\, d \leq \gamma(\alpha(d))$

3. $\forall a \in A \quad a = \alpha(\gamma(a))$

From Galois connection to abstract state transition fn.

$$\langle D, \circ_D, \leq_D, \top_D, \bot_D, I_D \rangle \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} \langle A, \circ_A, \leq_A, \top_A, \bot_A, I_A \rangle$$

Sp. $\langle \alpha, \gamma \rangle$ form a Galois connection

Can define $\underline{\underline{I_A}}$ in terms of I_D, α, γ.

$$I_A(a) = \alpha(I_D(\gamma(a)))$$

ie.. $I_A = \alpha \circ I_D \circ \gamma$

Yesterday's question: but this seems to defeat the purpose... (in terms of efficiency)

$$\langle D, \circ_D, \leq_D, \top_D, \bot_D, I_D \rangle \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} \langle A, \circ_A, \leq_A, \top_A, \bot_A, I_A \rangle$$

$$I_A = \alpha \circ I_D \circ \gamma$$

$$\boxed{\underline{\text{Theorem:}} \quad Reach(I_D) \leq_D \gamma(Reach(I_A))}$$

Thus, any property proved on I_A carries over to I_D !!

Yesterday's question: what does "Reach" mean, precisely?

Recipe for analysis:

Program's concrete interpretation : $\mathcal{C} = \langle D, \circ_D, \leq_D, \top_D, \bot_D, I_D \rangle$

Concrete semantics : Least Fix Point (I_D)

Difficulty : Least Fix Point (I_D) may be expensive to compute, or may not converge.

Solution: Come up with an abstract domain A and a Galois connection $D \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} A$

Immediately get : $\mathcal{A} = \langle A, \circ_A, \leq_A, \top_A, \bot_A, I_A \rangle$

$I_A = \gamma \circ I_D \circ \alpha$

Abstract Semantics : Least Fix Point (I_A)

Hopefully, easier to compute!

Today….

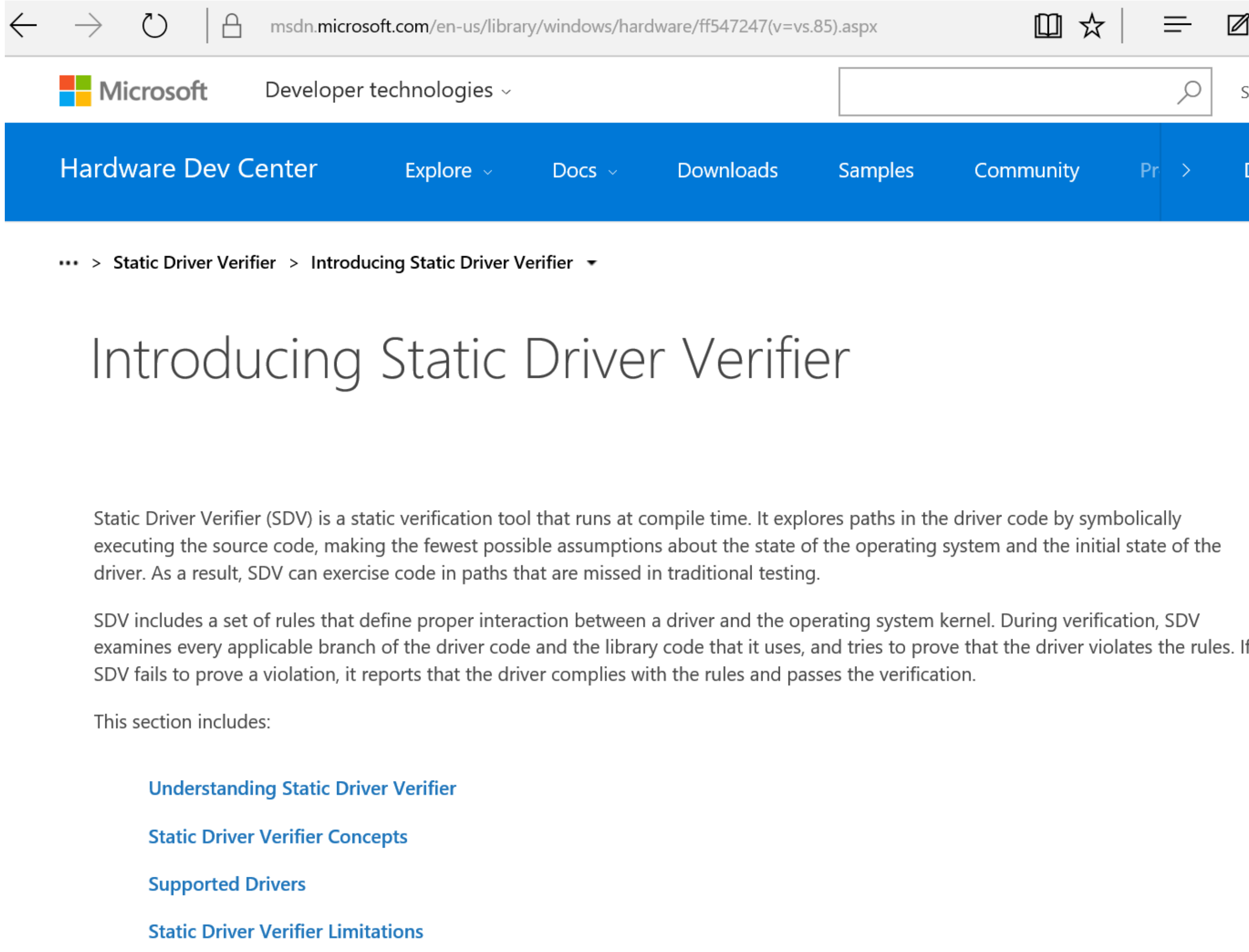
let me first show you a practical system based on abstract interpretation..

What is SLAM?

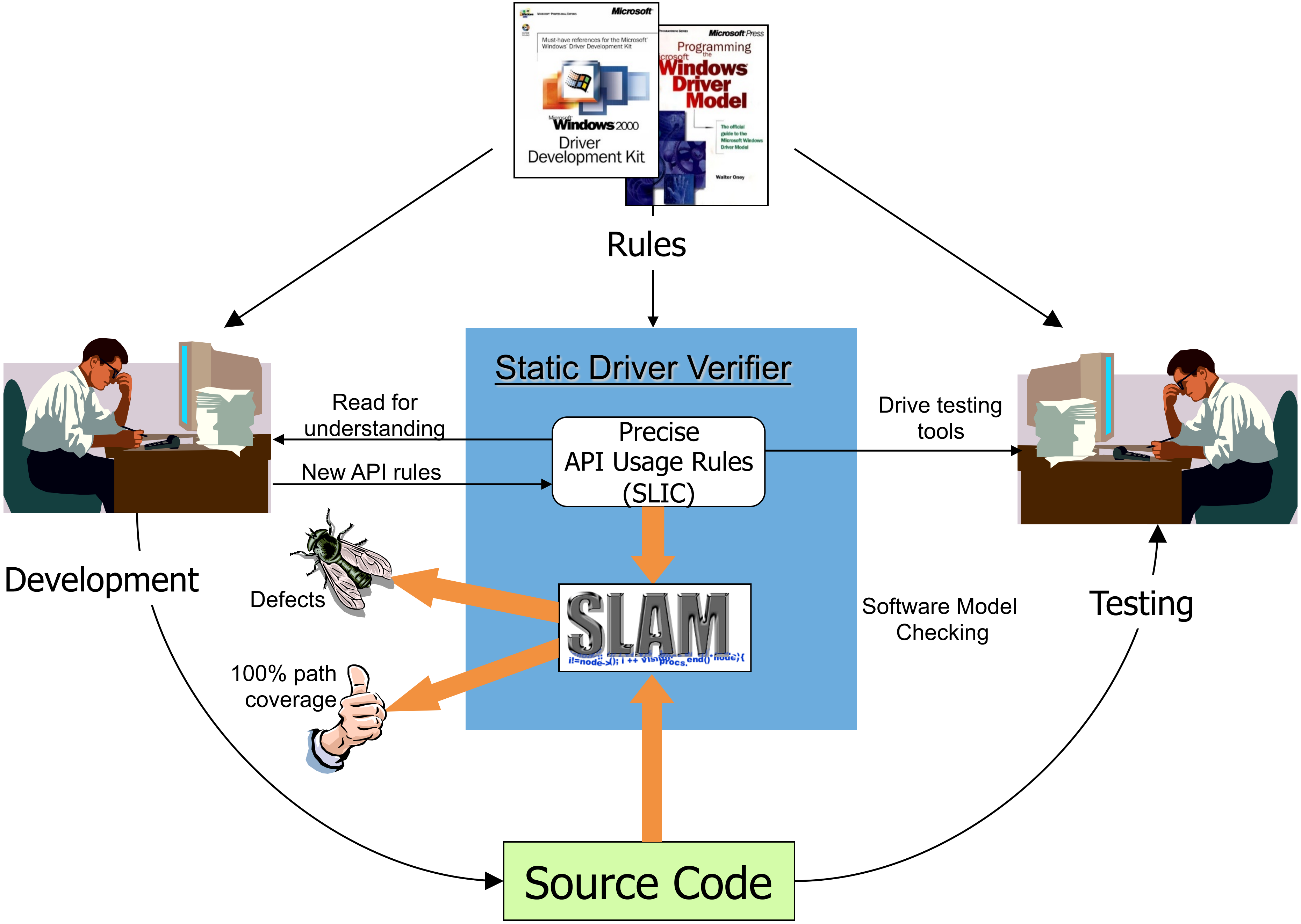
SLAM is a software model checking project at Microsoft Research.

- Goal: Check C programs (system software) against safety properties using model checking
- Application domain: device drivers

Shipped as part of “Static Driver Verifer” in the Windows Driver Development Kit for several releases



[A Decade of Software Model Checking with SLAM](), T. Ball, V. Levin, S. K. Rajamani, Communications of the ACM, Vol. 54. No. 7, 2011, Pages 68-76



Rules
Static Driver Verifier
Read for understanding
New API rules
Precise API Usage Rules (SLIC)
Drive testing tools
Development
Defects
SLAM
100% path coverage
Software Model Checking
Testing
Source Code

SLAM – Software Model Checking

- SLAM innovations
  - boolean programs: a new model for software
  - model creation (c2bp)
  - model checking (bebop)
  - model refinement (newton)

- SLAM toolkit
  - built on MSR program analysis infrastructure

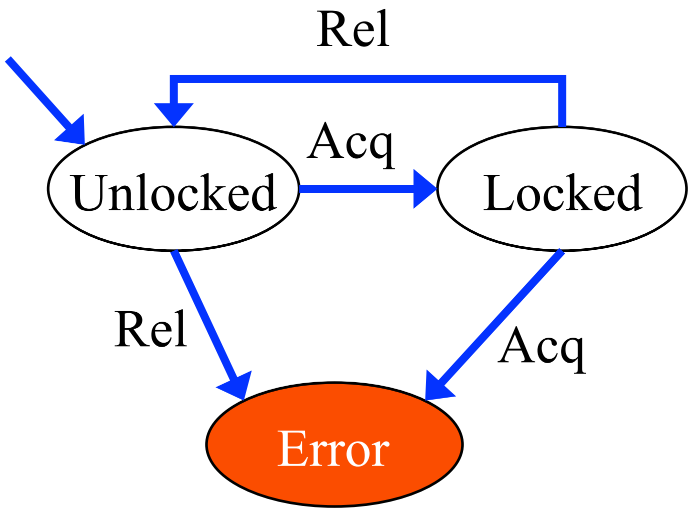
The model can be thought of as an "abstract interpretation"

SLIC

- Finite state language for stating rules
  - monitors behavior of C code
  - temporal safety properties (security automata)
  - familiar C syntax

- Suitable for expressing control-dominated properties
  - e.g. proper sequence of events
  - can encode data values inside state

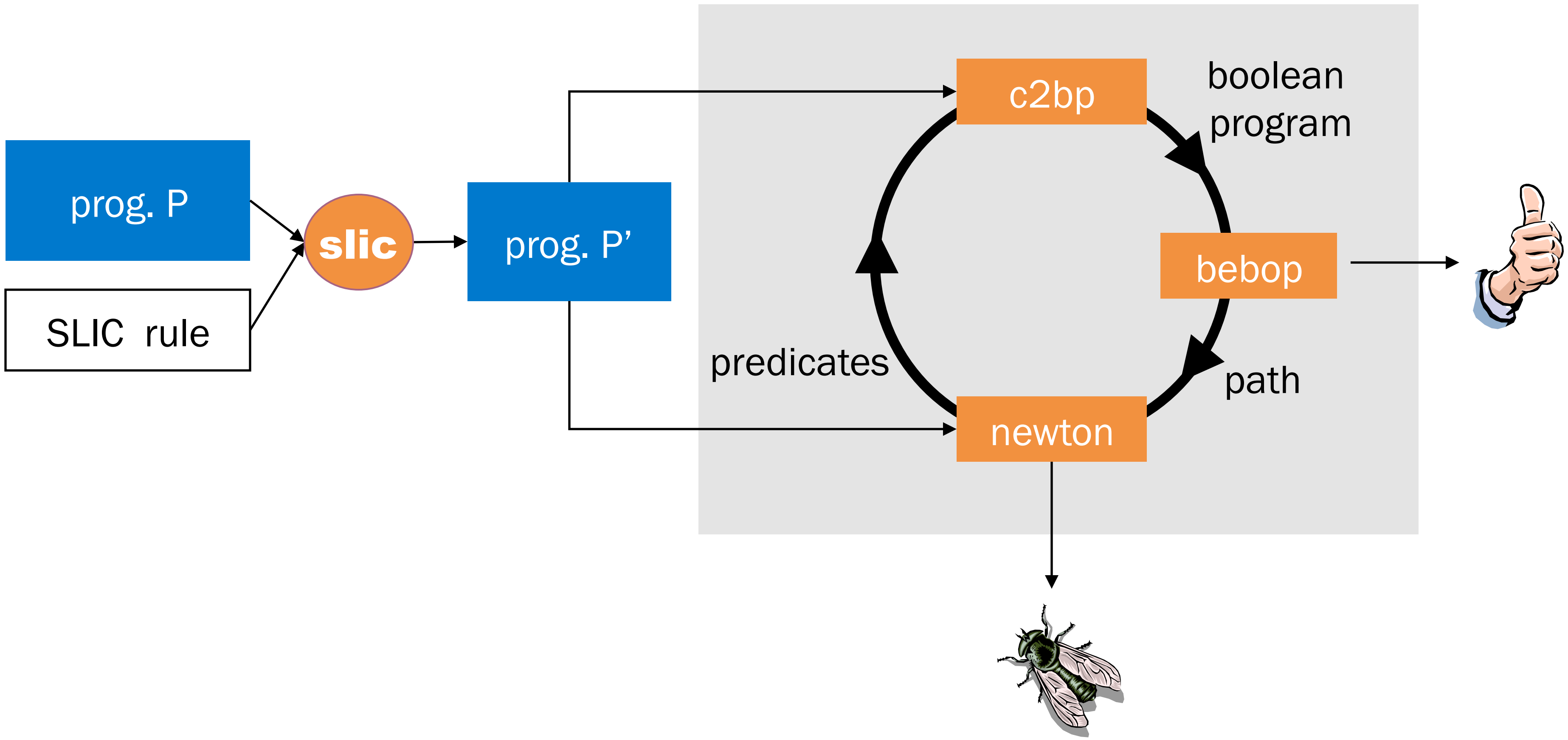
State Machine for Locking



Locking Rule in SLIC

```
state {
  enum {Locked,Unlocked}
    s = Unlocked;
}

KeAcquireSpinLock.entry {
  if (s==Locked) abort;
  else s = Locked;
}

KeReleaseSpinLock.entry {
  if (s==Unlocked) abort;
  else s = Unlocked;
}
```

The SLAM Process



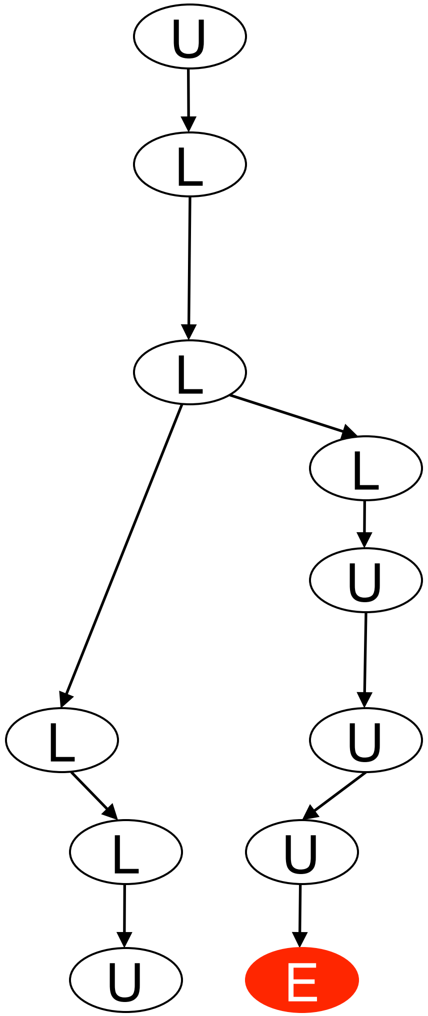
Example

Does this code obey the locking rule?

```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;

   if(request){
       request = request->Next;
       KeReleaseSpinLock();
       nPackets++;
   }
} while (nPackets != nPacketsOld);

KeReleaseSpinLock();
```

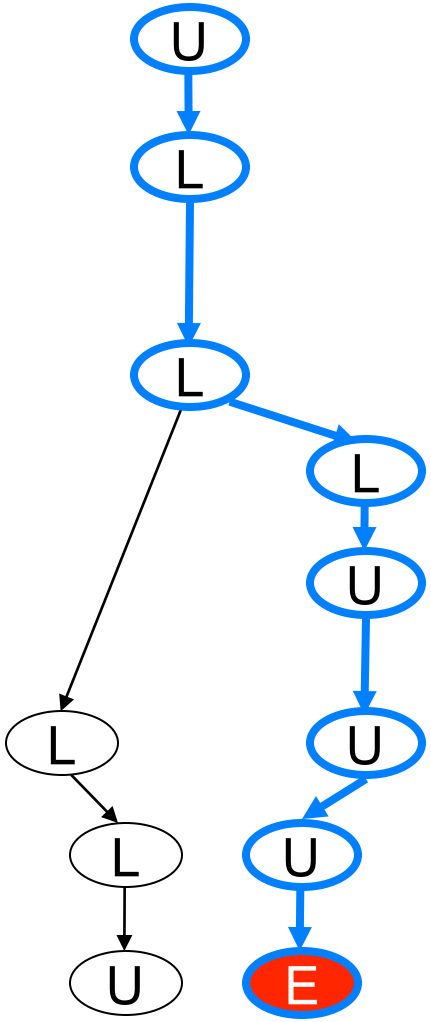
Example

Model checking
boolean program
(bebop)




```
do {
   KeAcquireSpinLock();

   if(*){

      KeReleaseSpinLock();

   }
} while (*);

KeReleaseSpinLock();
```

Example



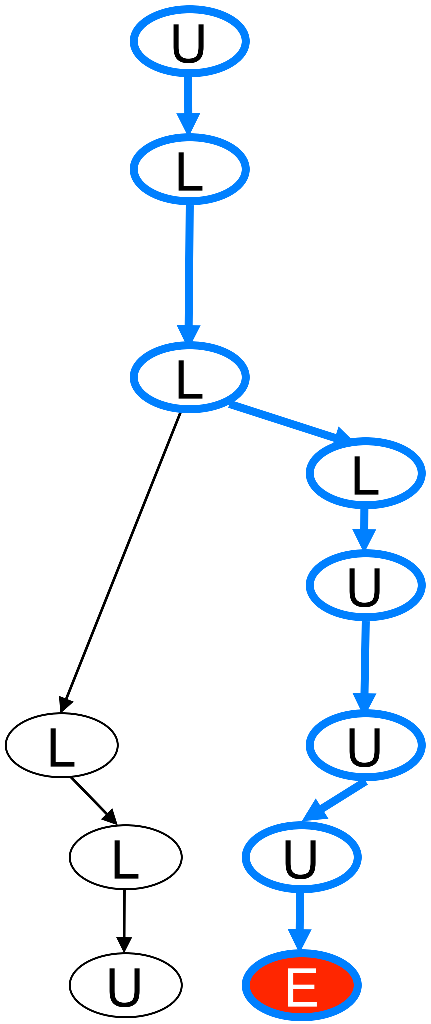

```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;

   if(request){
       request = request->Next;
       KeReleaseSpinLock();
       nPackets++;
   }
} while (nPackets != nPacketsOld);

KeReleaseSpinLock();
```

Is error path feasible in C program? (newton)

Example

b : (nPacketsOld == nPackets)

Add new predicate to boolean program (c2bp)

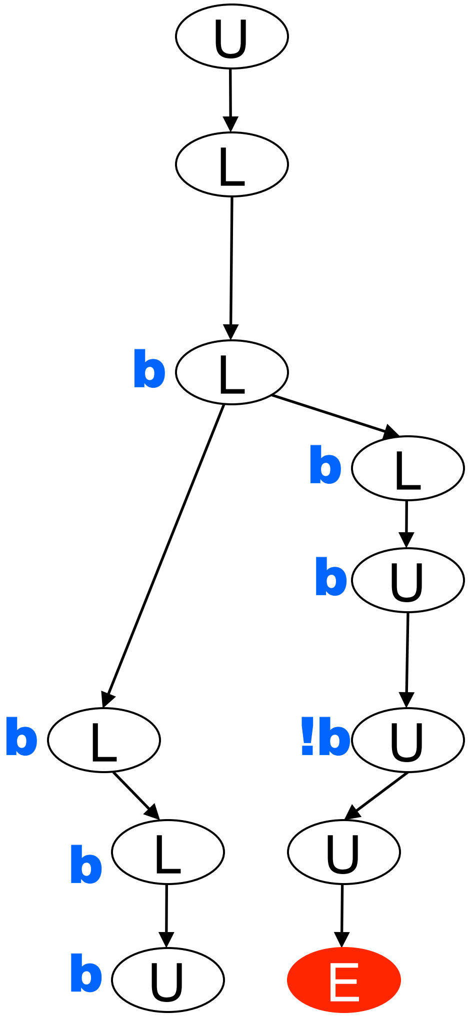



```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets; b = true;

   if(request){
       request = request->Next;
       KeReleaseSpinLock();
       nPackets++; b = b ? false : *;
   }
} while (nPackets != nPacketsOld);  !b

KeReleaseSpinLock();
```

Example

b : (nPacketsOld == nPackets)

Model checking refined boolean program (bebop)

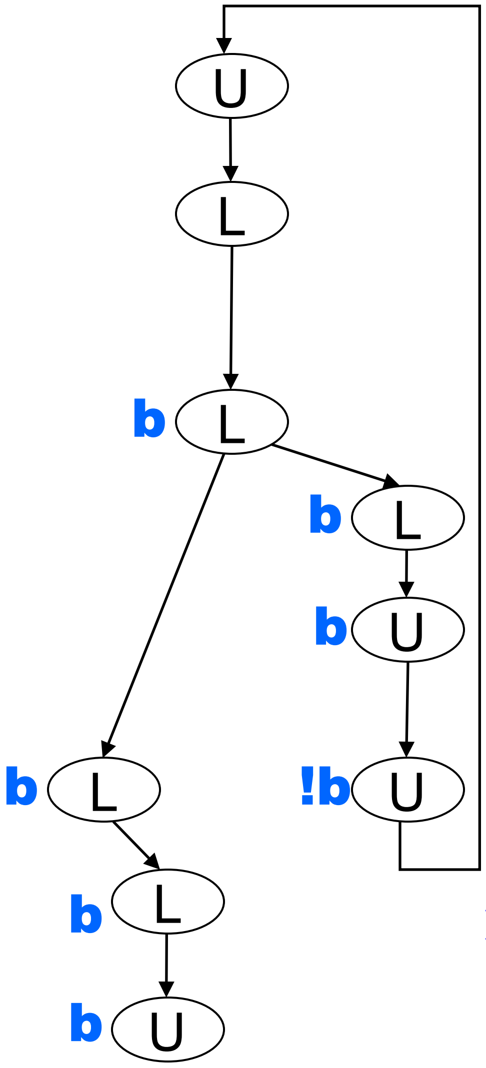



```
do {
   KeAcquireSpinLock();

   b = true;

   if(*){

      KeReleaseSpinLock();
      b = b ? false : *;
   }
} while ( !b );

KeReleaseSpinLock();
```

Example

b : (nPacketsOld == nPackets)

Model checking refined boolean program (bebop)




```
do {
  KeAcquireSpinLock();

  b = true;

  if(*){

      KeReleaseSpinLock();
      b = b ? false : *;
  }
} while ( !b );

KeReleaseSpinLock();
```

What is the loop invariant for this loop?

Questions:

- What is α for SLAM?
- What is γ for SLAM?
- When is the "fixpoint" or "reachability" in the abstract domain guaranteed to terminate?

$$\langle D, \circ_D, \leq_D, \top_D, \bot_D, I_D \rangle \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} \langle A, \circ_A, \leq_A, \top_A, \bot_A, I_A \rangle$$

$$I_A = \alpha \circ I_D \circ \gamma$$

$$\boxed{\underline{\text{Theorem:}} \quad Reach(I_D) \leq_D \gamma(Reach(I_A))}$$

Yesterday's question: what does "Reach" mean, precisely?

Convergence of fixpoints

Consider $\mathcal{A} = \langle A, \circ_A, \leq_A, \top_A, \bot_A, \mathcal{I}_A \rangle$

If A has finite height then $LFP(\mathcal{I}_A)$ will converge even if A is infinite

If A has infinite height (or large finite height) then $LFP(\mathcal{I}_A)$ may not converge (or may be too expensive to compute)

Two techniques:

1) Widening: to accelerate and ensure convergence of fixpoints (even if A has infinite height)

2) Narrowing: to improve precision of fixpoint computed by widening

Consider $\mathcal{A} = \langle A, \cup, \leq, \top, \bot, I \rangle$

<u>Fixpoint computation:</u>

$$A_0 = \bot$$

$$A_1 = A_0 \cup I(A_0)$$

$$A_2 = A_1 \cup I(A_1)$$

$$\vdots$$

$$A_n = A_{n-1} \cup I(A_{n-1})$$

$$\vdots$$

(may not converge)

Widening ∇ is a binary operator

$$\nabla : A \times A \xrightarrow[\text{partial}]{} A$$

$S_1 \nabla S_2$ <u>defined</u> only if $S_1 \leq S_2$

Two properties of ∇:

1. Sp. $S_1 \nabla S_2 = S_3$, then $S_1 \leq S_2 \leq S_3$
2. For any infinite sequence

   $S_0 \leq S_1 \leq S_2 \leq S_3 \ldots\ldots\ldots$

   Sp. $R_0 = S_0$

   $R_i = R_{i-1} \nabla (R_{i-1} \cup S_i), \quad i > 0$

   <u>Then</u>

   $R_0 \leq R_1 \leq R_2 \leq R_3 \ldots\ldots$ Converges!!

Consider $\mathcal{A} = \langle A, \cup, \leq, \top, \bot, \mathcal{I} \rangle$

Fixpoint computation:

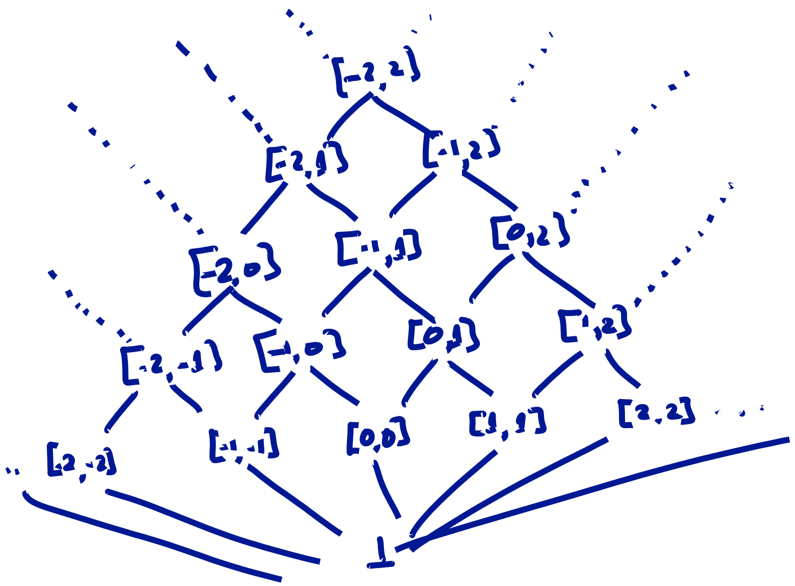
$$A_0 = \bot$$
$$A_1 = A_0 \cup \mathcal{I}(A_0)$$
$$A_2 = A_1 \cup \mathcal{I}(A_1)$$
$$\vdots$$
$$A_n = A_{n-1} \cup \mathcal{I}(A_{n-1})$$
$$\vdots$$

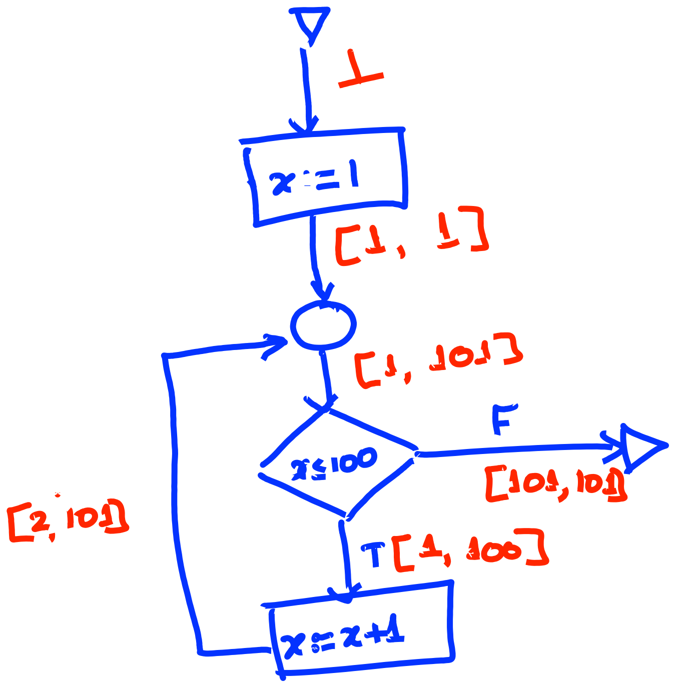
(may not converge)

$$R_0 = \bot$$
$$R_1 = R_0 \nabla (R_0 \cup \mathcal{I}(R_0))$$
$$R_2 = R_1 \nabla (R_1 \cup \mathcal{I}(R_1))$$
$$\vdots$$
$$R_n = R_{n-1} \nabla (R_{n-1} \cup \mathcal{I}(R_{n-1}))$$
$$\vdots$$

Guaranteed to converge!

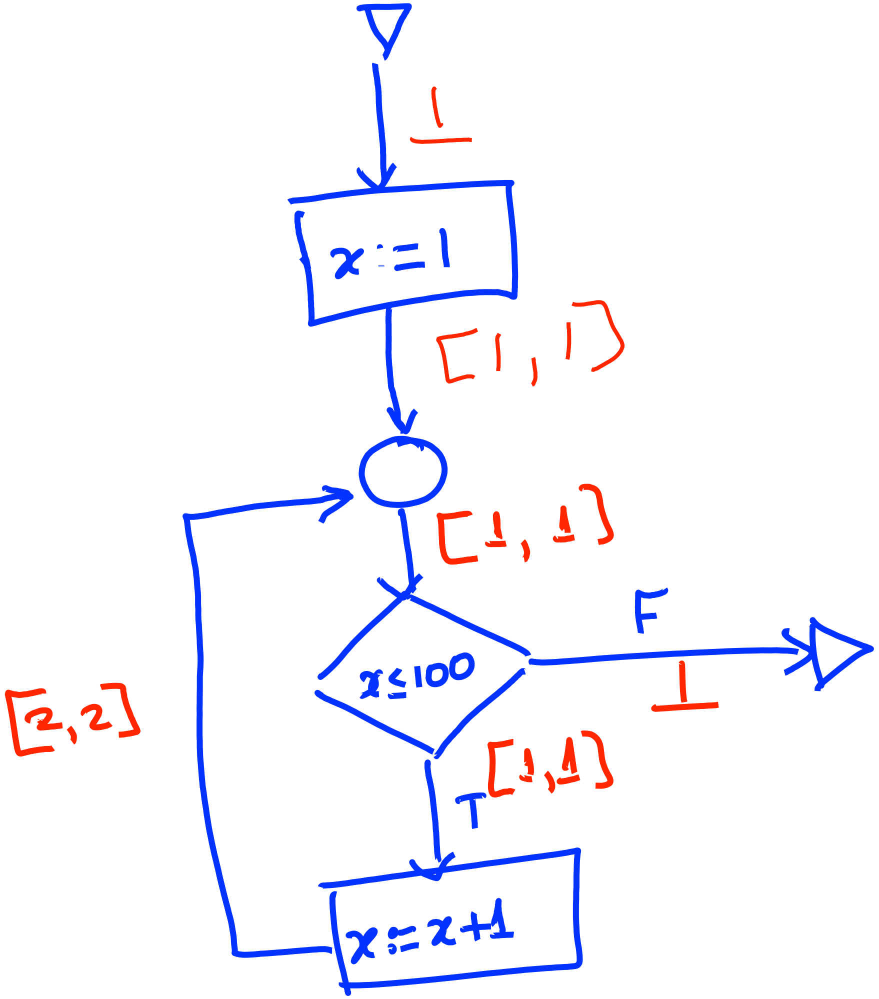
Widening over intervals



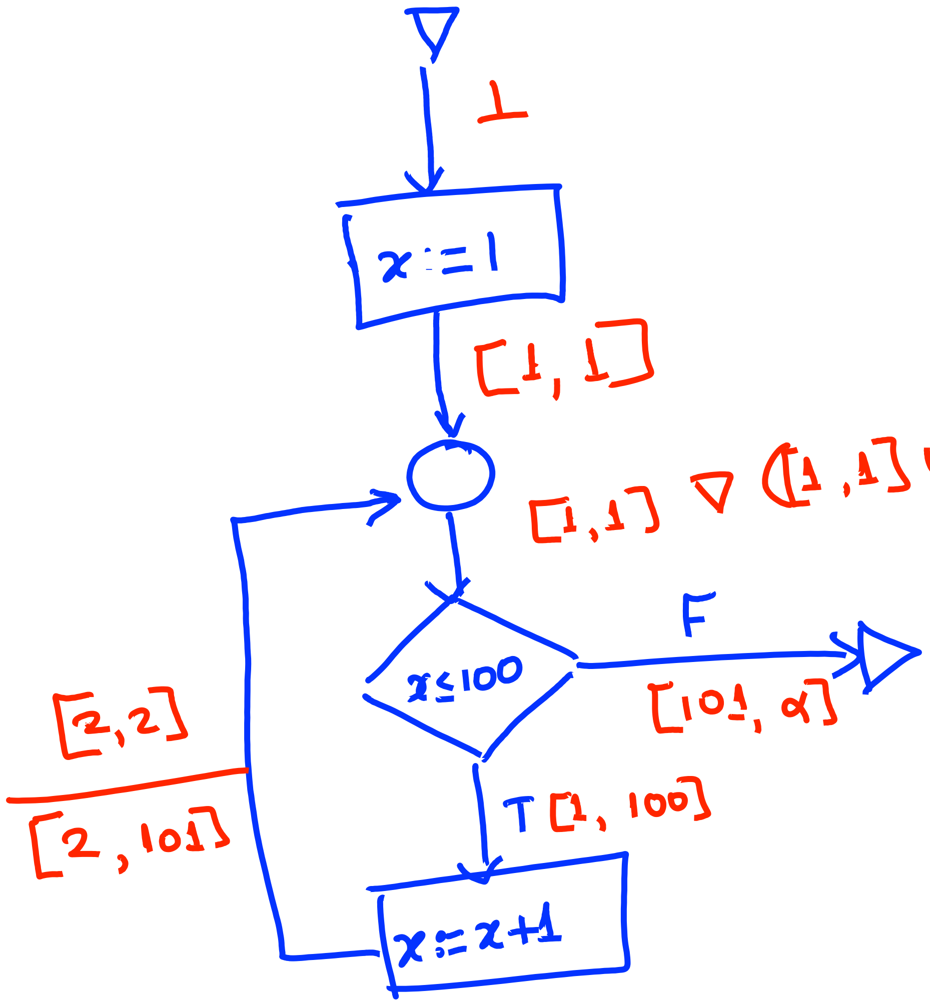


$$[i, j] \nabla [k, l] = [\text{if } k < i \text{ then } -\infty \text{ else } i,\ \text{if } l > j \text{ then } +\infty \text{ else } j]$$

First iteration

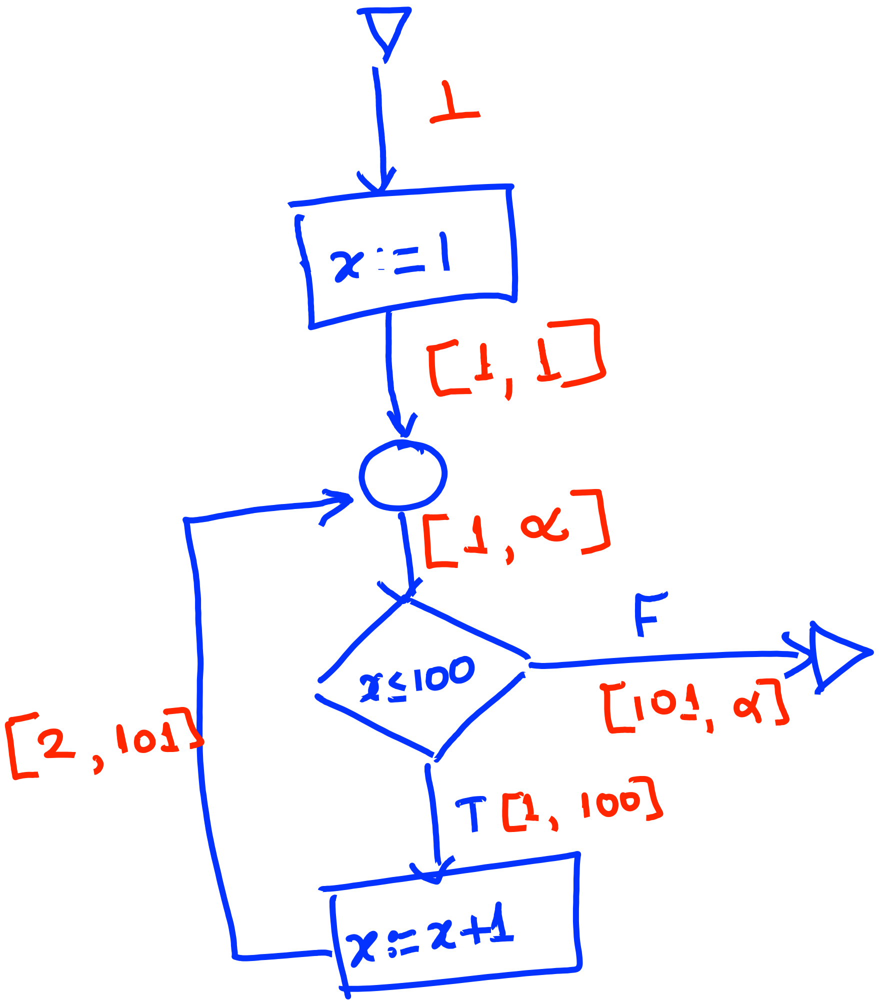


Second iteration



$$[1,1] \,\nabla\, ([1,1] \cup [2,2]) = [1,1] \,\nabla\, [2,2]$$
$$= [1, \infty]$$

Fixpoint



Homework

- Read the SLAM CACM paper: A Decade of Software Model Checking with SLAM, T. Ball, V. Levin, S. K. Rajamani, Communications of the ACM, Vol. 54. No. 7, 2011, Pages 68-76
- Design an abstract domain which tracks if an integer is even or odd. Try and implement an abstract interpreter for such a domain (for any program which works over integers), and infer if the output is (1) always odd, or (2) always even or (3) can't tell.
- Do the same as above but using a slightly richer abstract domain that keeps track of whether the value of integer is 0, 1 or 2 modulo 3.