tball@microsoft.com

- Falcons, Apple][,1981
- B.A. Cornell, 1987
- Ph.D. Univ. Wisc., 1993
- AT&T Bell Labs, 1993-96
- Lucent Technologies Bell Labs, 1996-99
- Microsoft Research, 1999-present
- Research interests
  - software reliability
  - programming languages, program analysis, model checking, automated theorem proving

Microsoft

~700

worldwide

computer

scien

Redmond

Software Development

Software Productivity

SLAM → Static Driver

Testing

Zap

Verification

theorem prover

→ an

Bartok & Phoenix backends

Testing, Verification and Measurement

- Tom Ball
- Madan Musuvathi (Stanford)
- Shuvendu Lahiri (CMU)
- Nachi Nagappan (NCSU)

- Visitors
  - Orna Kupferman (Hebrew Univ.), Mooly Sagiv (Tel-Aviv Univ.), Andrei Voronkov (Univ. Manchester), Andreas Zeller (Univ. Saarland)
  - Domagoj Babic, Sumit Gulwani, Krishna Mehra, Roman Manevich, Carlos Pacheco, Greta Yorsh

Microsoft Research: University Relations

- Hiring Ph.D.s
- Fellowships
- Summer internships
- New faculty awards
- Research grants in selected areas
- Sabattical
- Faculty Summit

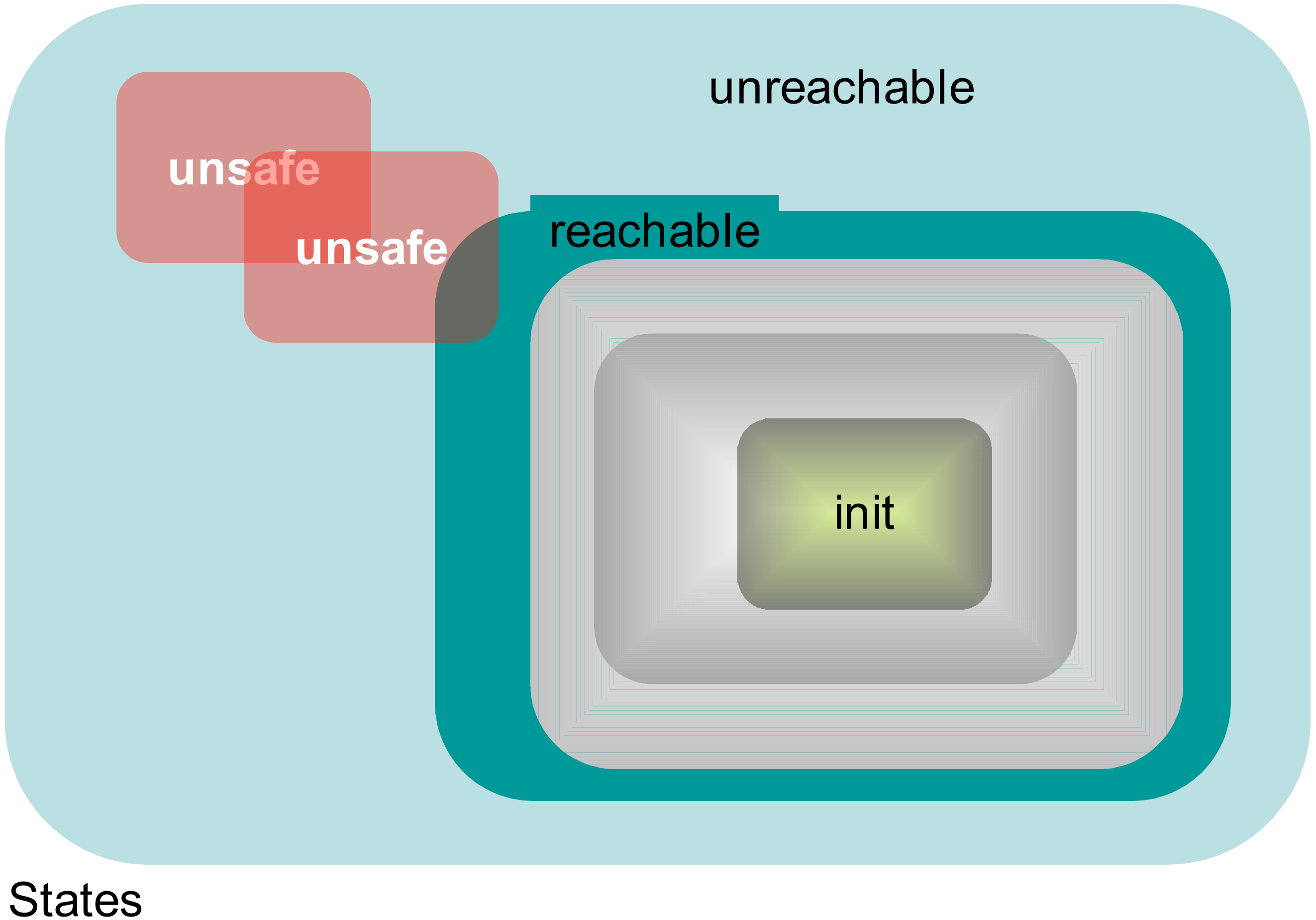
Automatic

Abstract

Automating Verification of Software

- Remains a “grand challenge” of computer science
- Behavioral abstraction is central to this effort
  - abstractions simplify our view of program behavior
  - proofs over the abstractions carry over to proofs over the program

Reachability



Aside

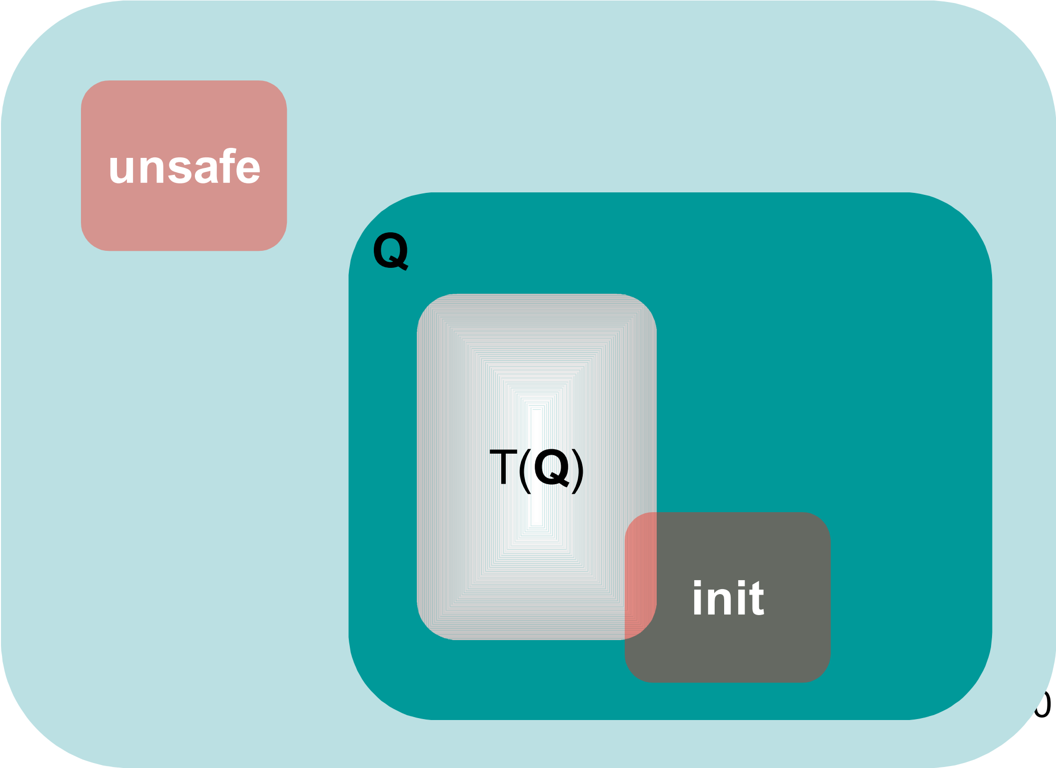
Reac hability

Assertion ≃

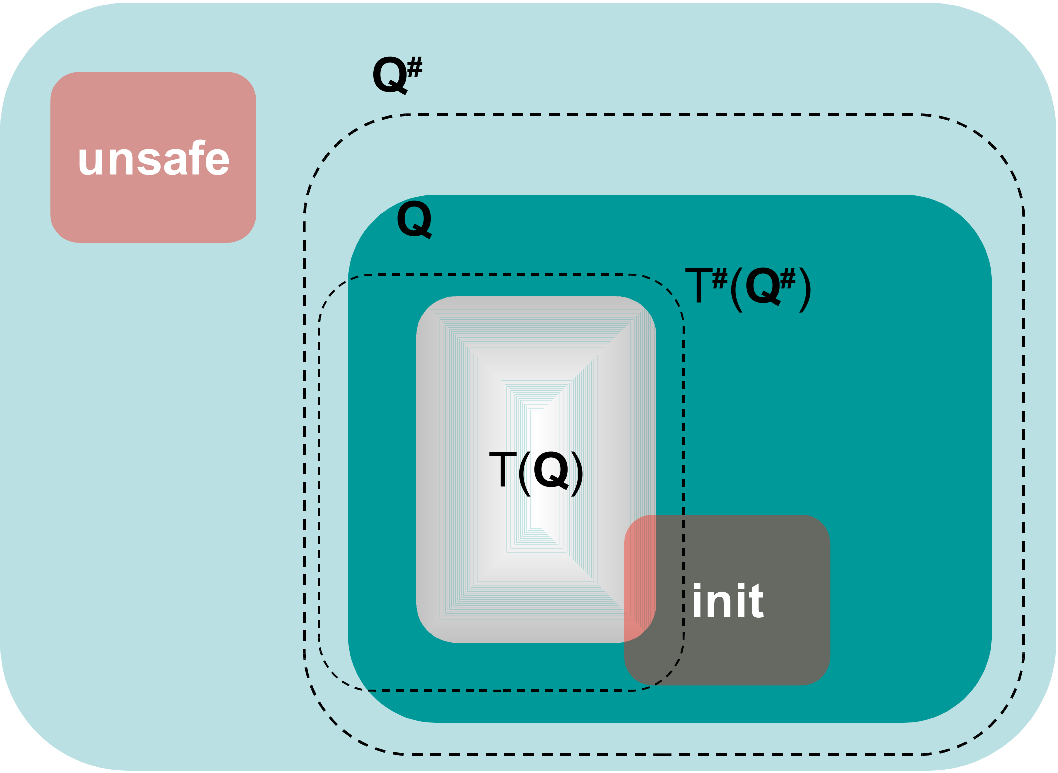
/Invariant ~

Safe Invariants

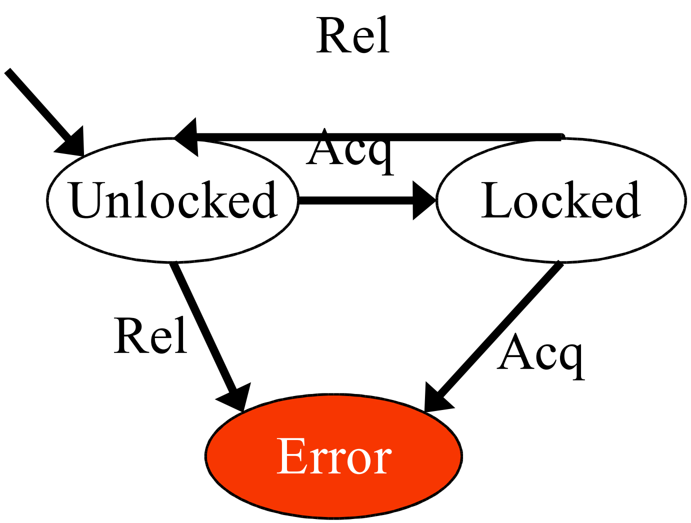
- Q is a safe invariant if
  - init $\subseteq$ Q
  - T(Q) $\subseteq$ Q
  - Q $\subseteq$ safe



Abstraction = Overapproximation of Behavior



More Concretely



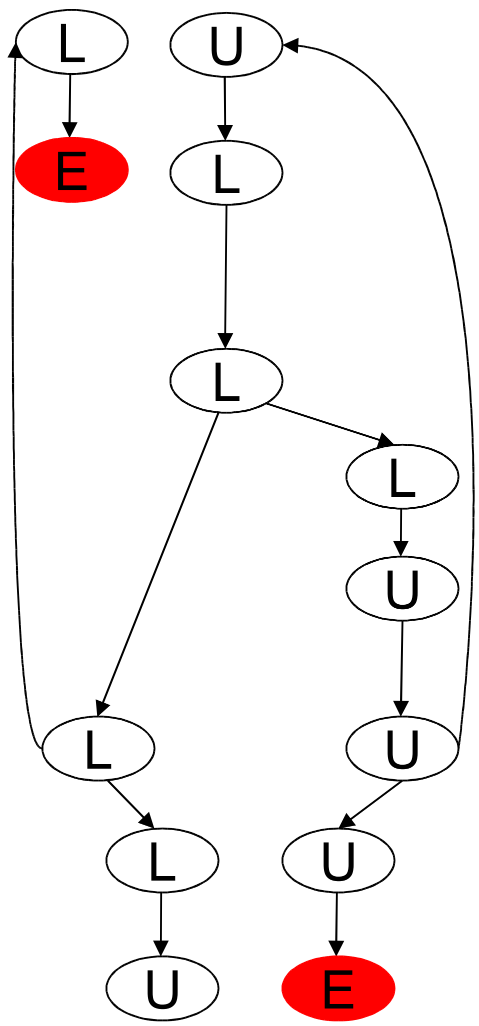

```
do {
  KeAcquireSpinLock();

  nPacketsOld = nPackets;

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

KeReleaseSpinLock();
```

Abstraction (via Boolean program)

```
do {
  KeAcquireSpinLock();

  nPacketsOld = nPackets;

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

KeReleaseSpinLock();
```

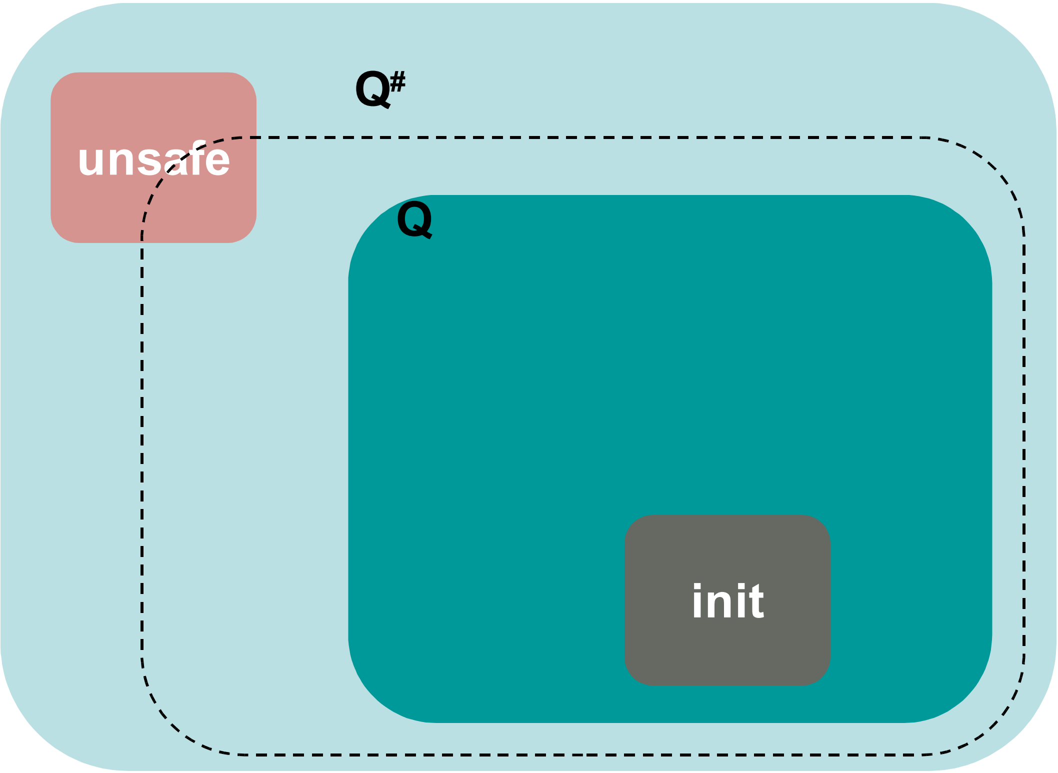
```
s:=U;
do {
  assert(s=U); s:=L;



  if(*){

     assert(s=L); s:=U;

  }
} while (*);

assert(s=L); s:=U;
```

State Space Exploration




```
s:=U;
do {
   assert(s=U); s:=L;

   if(*){
       assert(s=L); s:=U;
   }
} while (*);

assert(s=L); s:=U;
```

Overapproximation Too Large!



Refined Boolean Abstraction

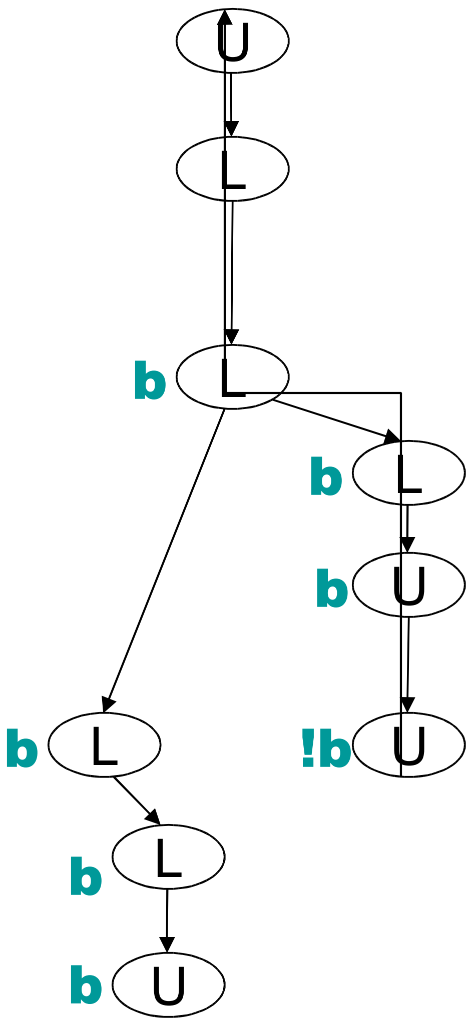
b : (nPacketsOld == nPackets)

```
do {
  KeAcquireSpinLock();

  nPacketsOld = nPackets;

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

KeReleaseSpinLock();
```

```
s:=U;
do {
  assert(s=U); s:=L;

  b := true;

  if(*){

     assert(s=L); s:=U;
     b := b ? false : *;
  }
} while ( !b );

assert(s=L); s:=U;
```

Refined Boolean Abstraction

b : (nPacketsOld == nPackets)




```
s:=U;
do {
   assert(s=U); s:=L;

   b := true;

   if(*){

      assert(s=L); s:=U;
      b := b ? false : *;
   }
} while ( !b );

assert(s=L); s:=U;
```

Invariant

"The lock is hel

of the loop if

Software Verification: A Search for Abstractions

- A complex search space with a fitness function (false errors)
  - search for right abstraction
  - search within state space of abstraction

- Can a machine beat a human at search for the right abstractions?

Overview

- Part I: Abstract Interpretation
  - [Cousot & Cousot, POPL'77]
  - *Manual abstraction and refinement*
  - ASTRÉE Analyzer
- Part II: Predicate Abstraction
  - [Graf & Saïdi, CAV '97]
  - *Automated abstraction and refinement*
  - SLAM and Static Driver Verifier
- Part III: Comparing Approaches

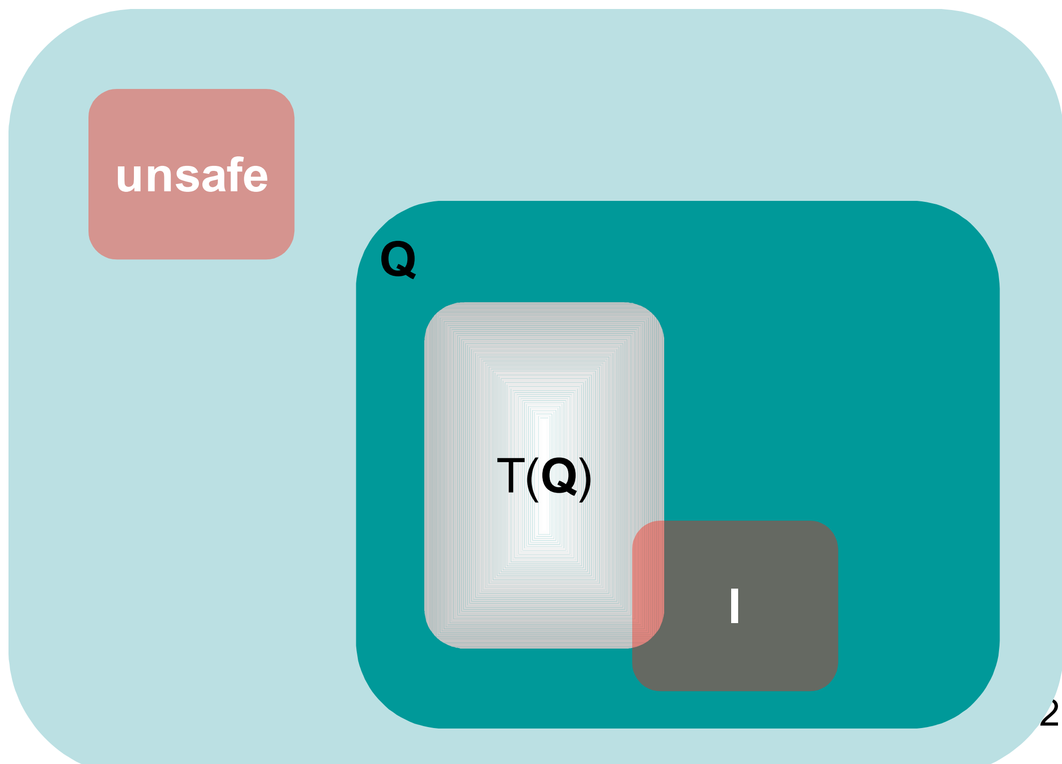
Concrete System

Prog = (C , I , T ,

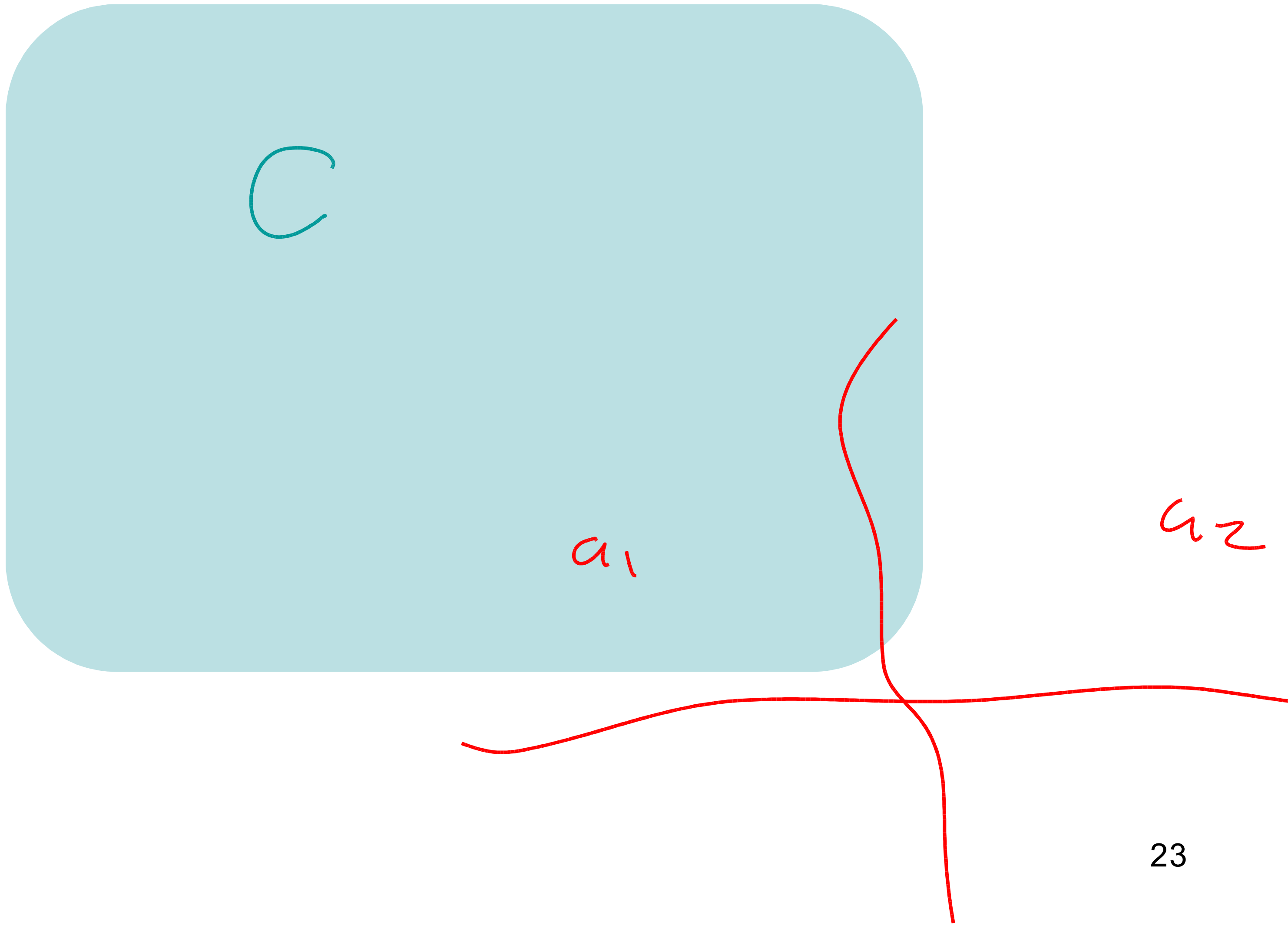
C : infinite

Safe Invariants

- Q is a safe invariant if
  - $I \subseteq Q$
  - $T(Q) \subseteq Q$
  - $Q \subseteq F$



Coping with In



α and γ function

α maps a set of

abstract element

$\alpha : 2^C \rightarrow A$

Abstraction

Sets of states order

Abstraction

Sets of states order

2^C

Abstraction

Sets of states   orde

2^C

Ordering in Abs

A embedded with in lattic

Ordering in Abs

A embedded with in lattic

A=S.

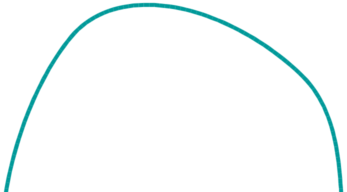
Ordering in Abs

A embedded with in lattic

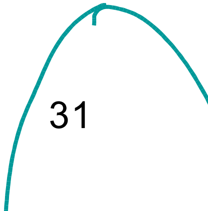
A = S.

Galois Connection

2^C



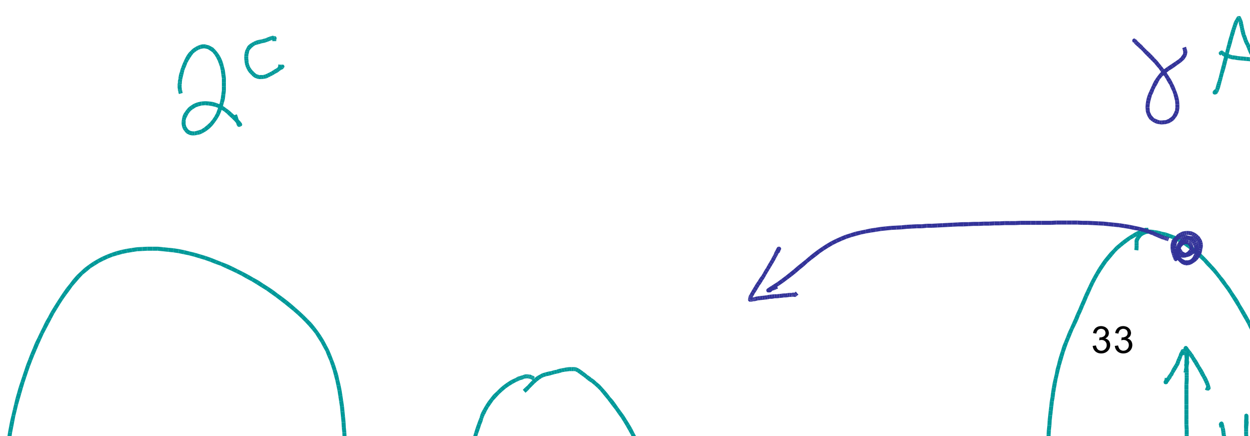
A



Galois Connection

2^C

Galois Connection



Example: Signs

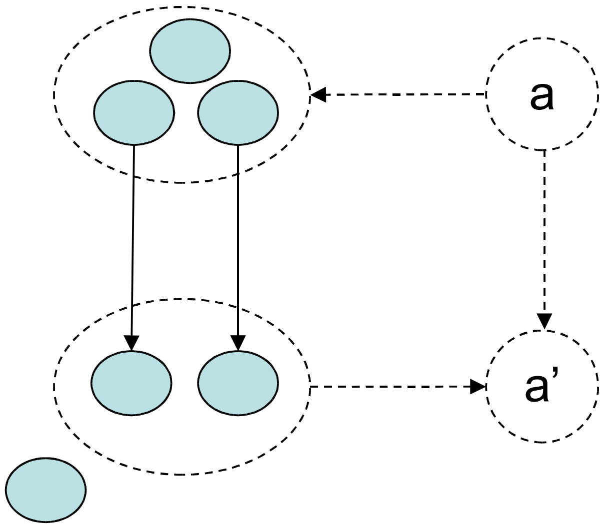
$$D = 2^{int}$$

Example:

Signs

$D = 2^{int}$

Abstract Transition Relation



γ

s

T

α

s'

Signs Transition

X := C;

X

Signs Transition

ass ume (x > 0);

Signs Transition

ass ert (x > 0);

Abstract Fixpoint

$X := \alpha(I);$

while $X \sqsubseteq \alpha(F)$

$X' := X$

Example

$\bot$

x := 0;

$\top$ $\bot$

while x <

Example

$\bot$

$x := 0;$

$\top$

while $x < 10$

$\top$

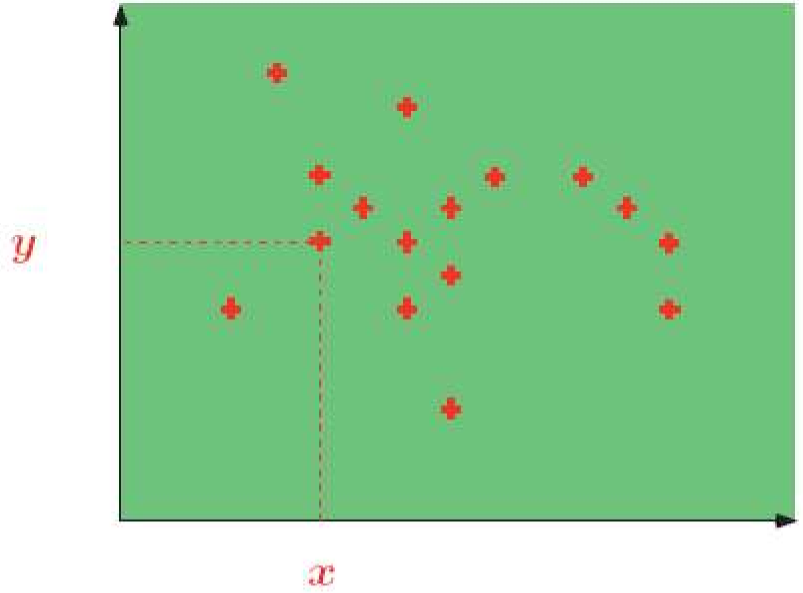
Signs Transition

ass ert (x ≤ 10, 000);

Refinement of

- signs a ∈
- intervals a ∈

Effective computable approximations of an [in]finite set of points; Signs [3]

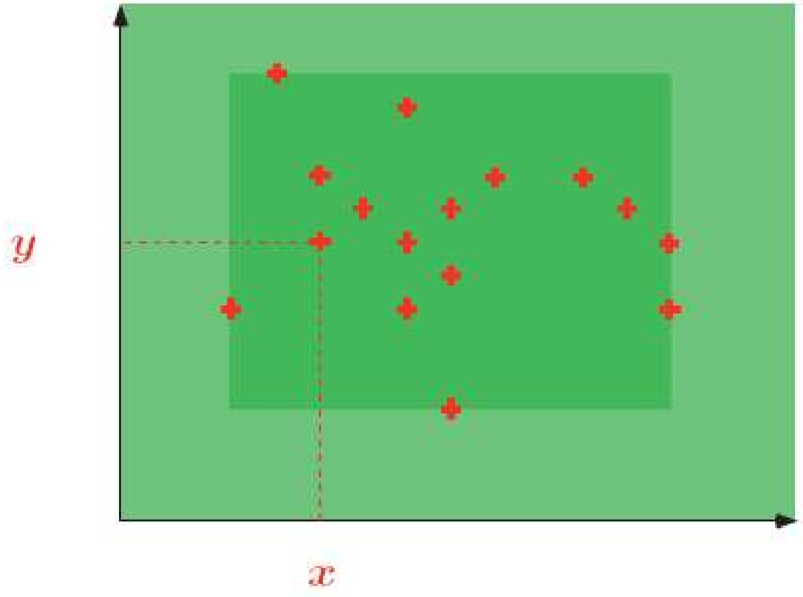


$$\begin{cases} x \geq 0 \\ y \geq 0 \end{cases}$$

3 P. Cousot & R. Cousot. *Systematic design of program analysis frameworks*. ACM POPL'79, pp. 269–282, 1979.

Slide courtesy of Patrick Cousot

Effective computable approximations of an [in]finite set of points; Intervals [4]

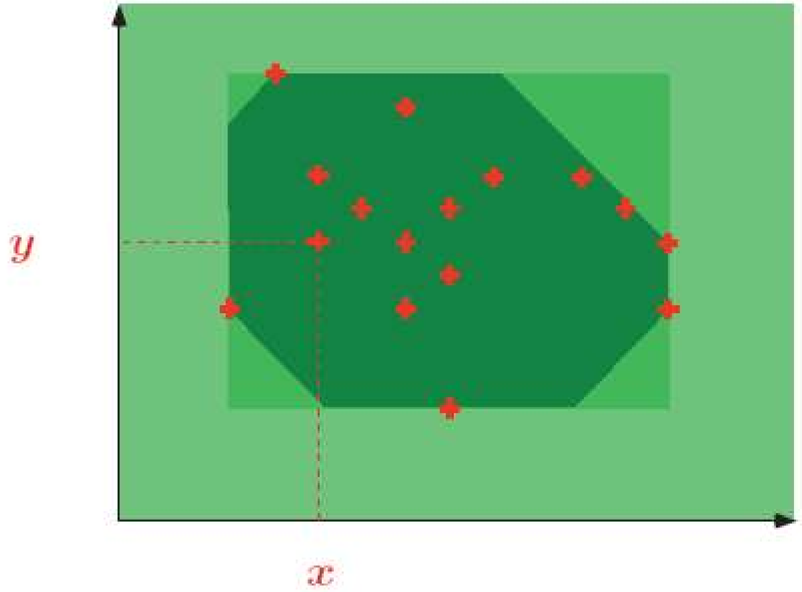


$$\begin{cases} x \in [19, 77] \\ y \in [20, 03] \end{cases}$$

[4] P. Cousot & R. Cousot. *Static determination of dynamic properties of programs.* Proc. 2nd Int. Symp. on Programming, Dunod, 1976.

Slide courtesy of Patrick Cousot

Effective computable approximations of an [in]finite set of points; Octagons [5]



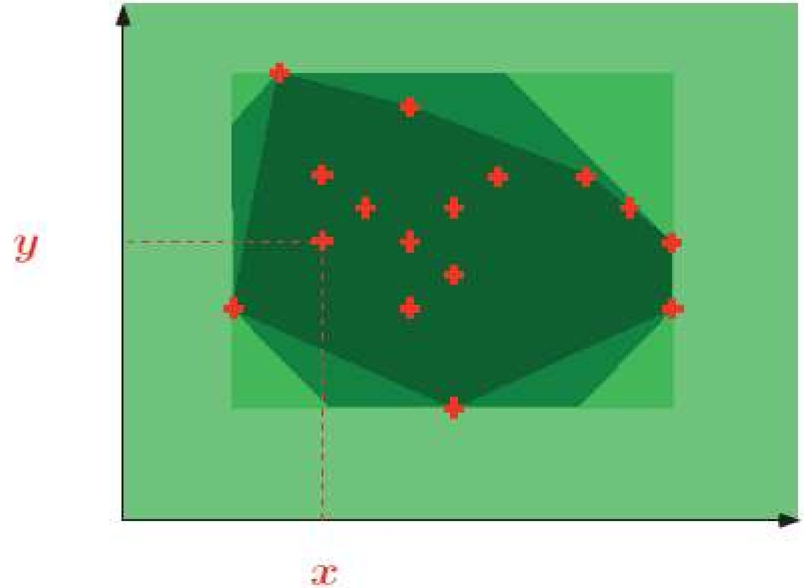
$$\begin{cases} 1 \leq x \leq 9 \\ x + y \leq 77 \\ 1 \leq y \leq 9 \\ x - y \leq 99 \end{cases}$$

---

[5] A. Miné. *A New Numerical Abstract Domain Based on Difference-Bound Matrices*. PADO'2001. LNCS 2053, pp. 155–172. Springer 2001. See the *The Octagon Abstract Domain Library* on `http://www.di.ens.fr/~mine/oct/`

Slide courtesy of Patrick Cousot

Effective computable approximations of an [in]finite set of points; Polyhedra [6]



$$\begin{cases} 19x + 77y \leq 2004 \\ 20x + 03y \geq 0 \end{cases}$$

[6] P. Cousot & N. Halbwachs. *Automatic discovery of linear restraints among variables of a program.* ACM POPL, 1978, pp. 84–97.

Slide courtesy of Patrick Cousot

Overview

- Part I: Abstract Interpretation
  - [Cousot & Cousot, POPL'77]
  - *Manual abstraction and refinement*
  - ASTRÉE Analyzer
- Part II: Predicate Abstraction
  - [Graf & Saïdi, CAV '97]
  - *Automated abstraction and refinement*
  - SLAM and Static Driver Verifier
- Part III: Comparing Approaches

Abstract Interpretation, So Far

- Create abstract domain and supporting algorithms
- Relate domains via α and γ functions
- Prove Galois connection
- Create abstract transformer T# (for $\sqsubseteq$, $\sqcup$
- Show that T# approximates $\alpha \circ T \circ \gamma$
- Refinement to reduce false errors
- Widening to achieve termination

Example

$\bot$

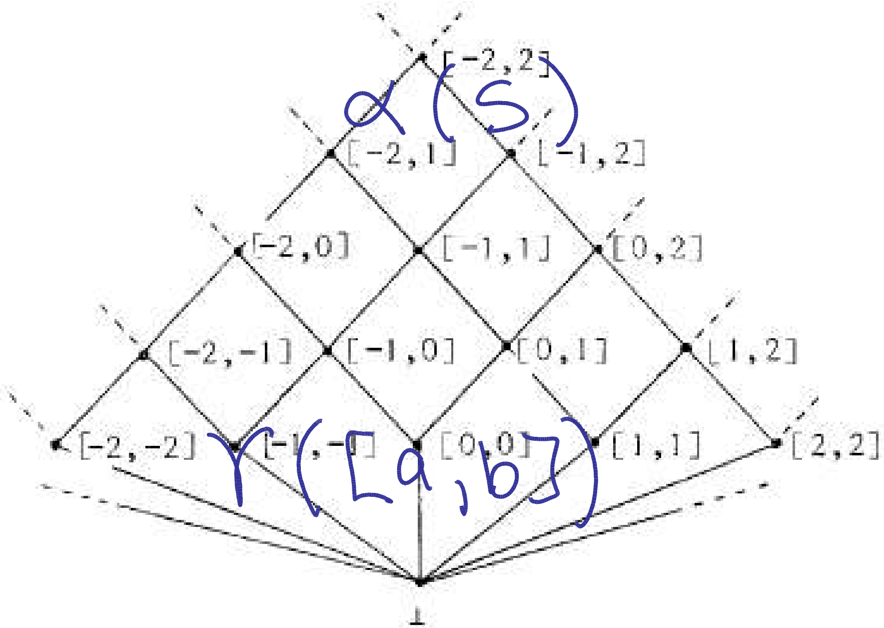
$x := 0;$

$\top$

while $x < 10$

$\top$

Integer Sets $\underset{\gamma}{\overset{\alpha}{\rightleftarrows}}$ Int



$\alpha(S)$ = [min (S),

$\gamma([a,b])$ = {x |

Diagram from Cousot, Cousot, POPL 1977

Interval

Transition

$x := c;$

$x :=$ $x + 1$

[

$x := 0;$

X_1 ___

while $x < 10{,}0$

X_2 ___

$X_1 = [0, 0]$

X_3 ___

$x := x + 1$

X_4 ___

Symbo lic Upper

n

x := 0 ;

while x <

Interval Widening

old new

$$[\,l_0\,,\,u_0\,]\ \nabla\ [\,l_1\,,\,u_1\,] =$$

$$[\ \text{if}\ l_1 < l_0\ \text{then}\ -\infty$$

Abstract Fixpoint

$X := \alpha(I);$

while $X \sqsubseteq \alpha(F)$

$X' := X$

$x := 0;$

X_1 ___

while $x < 10,0$

X_2 ___

$x_1 = [0,0]$

X_3 ___

$x := x+1$

X_4 ___

X

ASTRÉE Analyzer

Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne,
Antoine Miné, David Monniaux, Xavier Rival, Bruno Blanchet

ASTRÉE analyzes structured C programs, without dynamic memory allocation and recursion.

In Nov. 2003, ASTRÉE automatically proved the absence of any run-time error in the primary flight control software of the Airbus A340 fly-by-wire system

a program of 132,000 lines of C analyzed in 1^{h}20 on a 2.8 GHz 32-bit PC using 300 Mb of memory

Abstraction Refinement:
PLDI'03 Case Study of Blanchet et al.

- "… the initial design phase is an iterative manual refinement of the analyzer."

- "Each refinement step starts with a static analysis of the program, which yields false alarms. Then a manual backward inspection of the program starting from sample false alarms leads to the understanding of the origin of the imprecision of the analysis."

- "There can be two different reasons for the lack of precision:
  - some local invariants are expressible in the current version of the abstract domain but were missed
  - some local invariants are necessary in the correctness proof but are not expressible in the current version of the abstract domain."

Part I: Summary

- Create abstract domains and supporting algorithms
- Relate domains via α and γ functions
- Prove Galois connection
- Create abstract transformer T#
- Show that T# approximates $\alpha \circ T \circ \gamma$
- Refinement to reduce false errors
- Widening to achieve termination

Overview

- Part I: Abstract Interpretation
  - [Cousot & Cousot, POPL'77]
  - *Manual abstraction and refinement*
  - ASTRÉE Analyzer
- Part II: Predicate Abstraction
  - [Graf & Saïdi, CAV '97]
  - *Automated abstraction and refinement*
  - SLAM and Static Driver Verifier
- Part III: Comparing Approaches

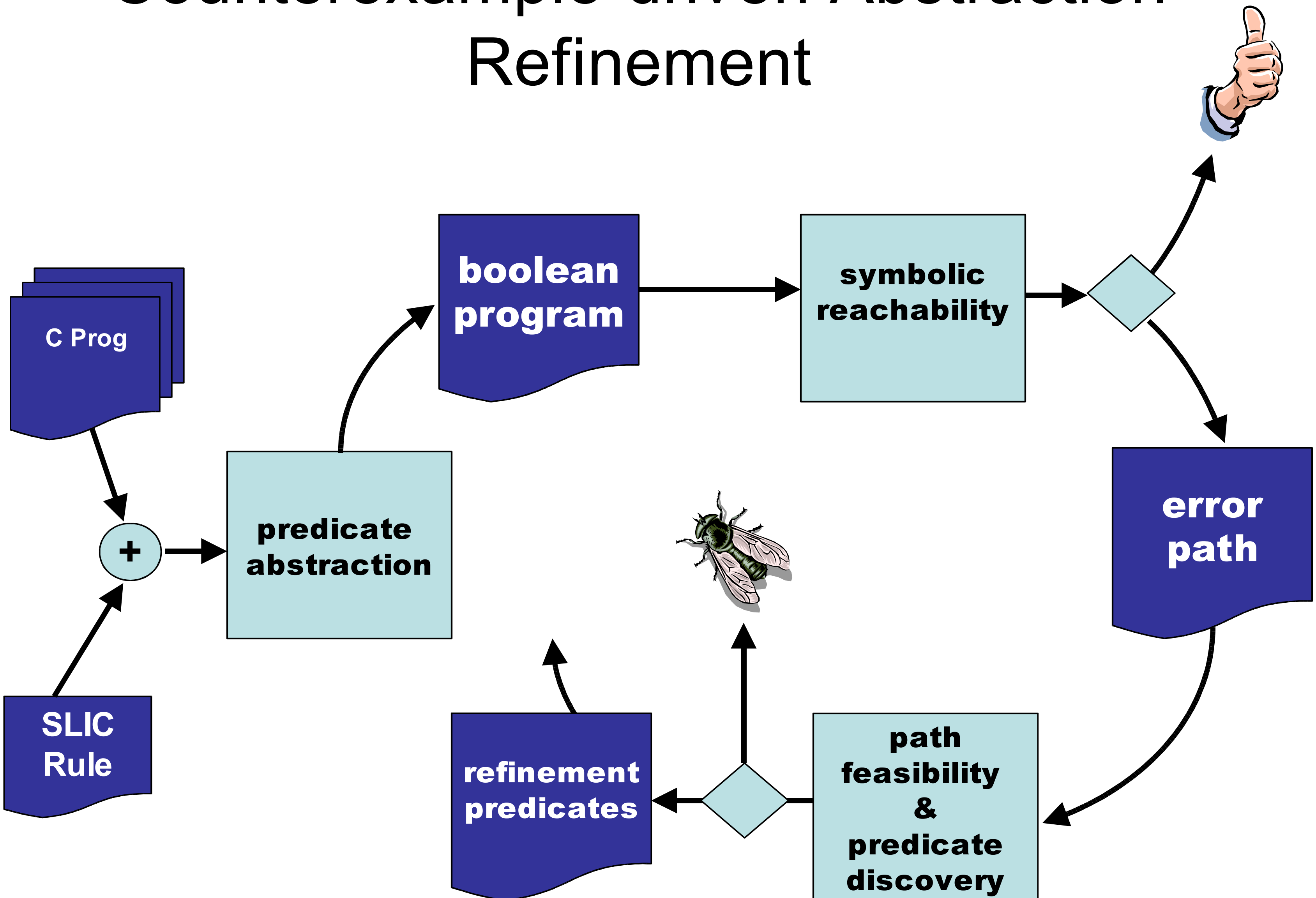
Boolean Abstraction

b : (nPacketsOld == nPackets)

```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;

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

KeReleaseSpinLock();
```

```
s:=U;
do {
   assert(s=U); s:=L;

   b := true;

   if(*){

       assert(s=L); s:=U;
       b := b ? false : *;
   }
} while ( !b );

assert(s=L); s:=U;
```

Counterexample-driven Abstraction Refinement



[Kurshan et al. '93]
[Clarke et al. '00]
[Ball, Rajamani '00]

Part II: Overview

- Predicate Abstraction
- Symbolic Reachability with BDDs
- Predicate Refinement

Predicate Abstraction

- Graf & Saïdi, CAV ’97

- Idea
  - Given set of predicates $P = \{ P_1, \ldots, P_k \}$
    - Formulas describing properties of system state
- Abstract State Space
  - Set of Boolean variables $B = \{ b_1, \ldots, b_k \}$
    - b_i = true $\Leftrightarrow$ Set of states where P_i holds

Approximating concrete states

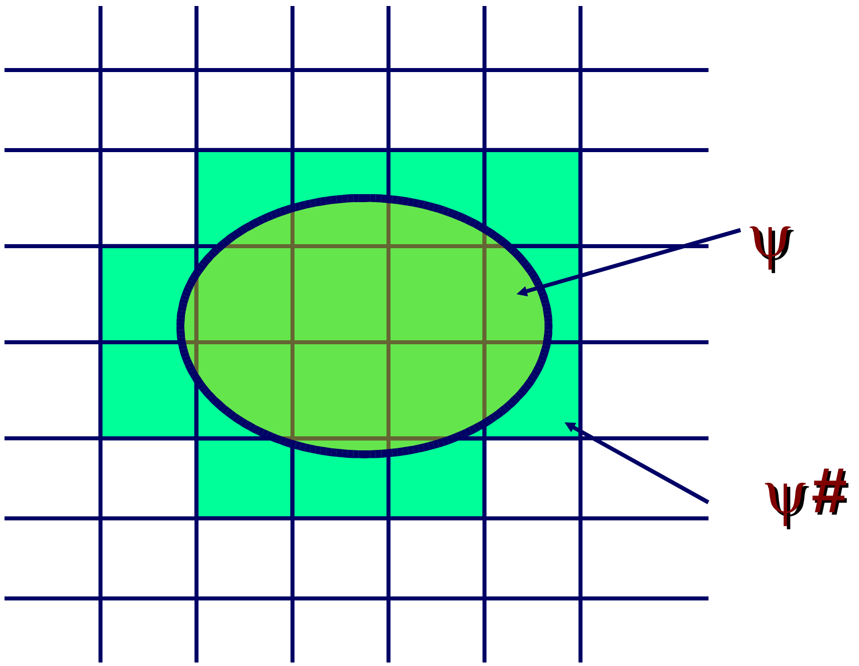
Fundamental Operation

- Approximating a set of concrete states by a set of predicates
- Requires exponential number of theorem prover calls in worst case

Compute Symbolically

- Main Operation

$\exists X.\ [\psi \wedge (\wedge_i\, b_i \Leftrightarrow P_i\ )\ ]$



Partitioning defined by the predicates

Similar to existential abstraction of finite state machines [Clarke, Grumberg, Long]

Abstraction α and Concretization γ Functions

$\alpha : \quad 2^{C} \longrightarrow A$

Abstraction α and Concretization γ Functions

$$\alpha : 2^C \longrightarrow A$$

$$2^C$$

ψ ⇆

Abstraction α and Concretization γ Functions

$\alpha : 2^C \longrightarrow A$

$2^C \cong \Psi$

Example

$$\Psi = (x = 1 \vee$$

Example

$\exists x. \quad (\quad x = \quad 1 \quad \vee$

Example

$\exists x . \quad (\; x = \quad 1 \quad \vee$

Example

$\exists x. \quad (\quad x = \quad 1 \quad \vee$

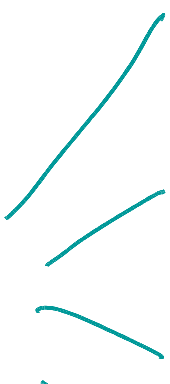
Example

$\exists x. \quad (\; x = \quad 1 \quad \vee$

Alternatively , check

of ψ against

$x < 5$



X

$(x = 1$ [illegible] $x = 6)$

Abstracting Assigns via WP

- WP(x:=e,Q) = Q[x -> e]

- WP(y:=y+1, y<5) =

  (y<5) [y -> y+1] =

  (y+1<5) =

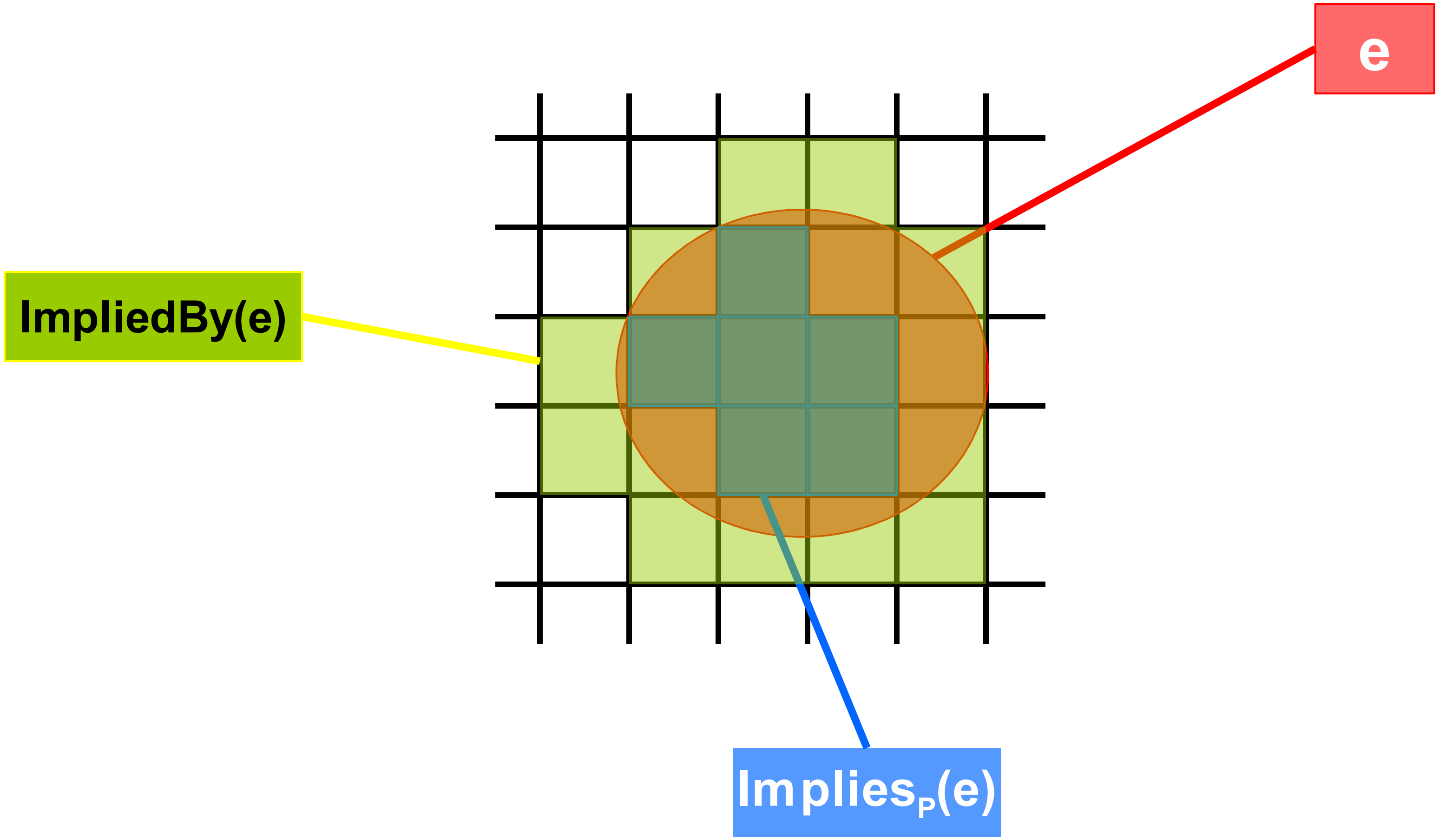
  (y<4)

WP Problem

- WP(s, p_i) not always expressible via P

- Example
  - P = { x=0, x=1, x<5 }
  - WP(x:=x+1 , x<5) = x<4

$\text{Implies}_F(e)$ and $\text{ImpliedBy}_F(e)$



Abstracting Assignments

- if $Implies_P(WP(s, p_i))$ is true before s then
  - p_i is true after s
- if $Implies_P(WP(s, !p_i))$ is true before s then
  - p_i is false after s

$$
\begin{aligned}
b_i \quad := \quad & Implies_P(WP(s, p_i)) \quad ? \quad true : \\
& Implies_F(WP(s, !p_i)) \quad ? \quad false \\
& \quad : \quad *;
\end{aligned}
$$

Assignment Example

Statement:

y := y+1;

Predicates in P:

{x=y}

Weakest Precondition:

WP(y:=y+1, x=y) = x=y+1

$Implies_F$(x=y+1) = ?

$Implies_F$(x!=y+1) = ?

Assignment Example

Statement:

y := y+1;

Predicates in P:

{x=y}

Weakest Precondition:

WP(y:=y+1, x=y) = x=y+1

Implies_F(x=y+1) =

Implies_F(x!=y+1) =

Abstraction of assignment in B:

b = b ? false : *;

Abstracting Assumes

- assume(e) is abstracted to:

  assume(ImpliedBy_P(e))

- Example:

  P = {x=2, x<5}

  assume(x < 2) is abstracted to:

  assume({x<5} && !{x==2})

Assume, Explained

<u>if</u> "assume" evaluates

<u>then</u> it must eval.

Refined Boolean Abstraction

b : (nPacketsOld == nPackets)

```
do {
  KeAcquireSpinLock();

  nPacketsOld = nPackets;

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

KeReleaseSpinLock();
```

```
s:=U;
do {
  assert(s=U); s:=L;

  b := true;

  if(*){

      assert(s=L); s:=U;
      b := b ? false : *;
  }
} while ( !b );

assert(s=L); s:=U;
```

Aside

Predicate abstraction

- procedures

Part II: Overview

- Predicate Abstraction
- Symbolic Reachability with BDDs
- Predicate Refinement

Reachability in Boolean Programs

bool id (bool x , bool z)

decl y ;

L1 : y := ! x ;

Reachability in Boolean Programs

bool id (bool x, bool z)

decl y;

L1: y := !x;

Reachability in Boolean Programs

```
bool id(bool x, bool z)
    decl y;
L1: y := !x;
```

Reachability in Boolean Programs

bool id (bool x, bool z)

decl y;

L1: y := !x;

Reachability in Boolean Programs

bool id (bool x , bool z)

decl y ;

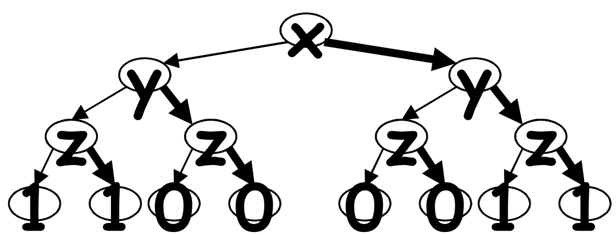
L1 : y := ! x ;

Binary Decision Diagrams

- Acyclic graph data structure for representing a boolean function (equivalently, a set of bit vectors)
- $F(x,y,z) = (x=y)$

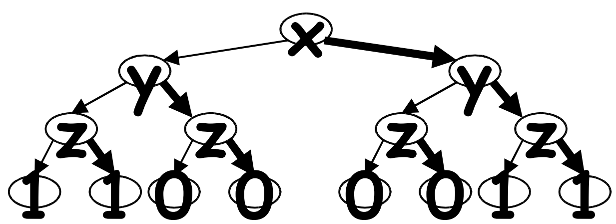
Binary Decision Diagrams

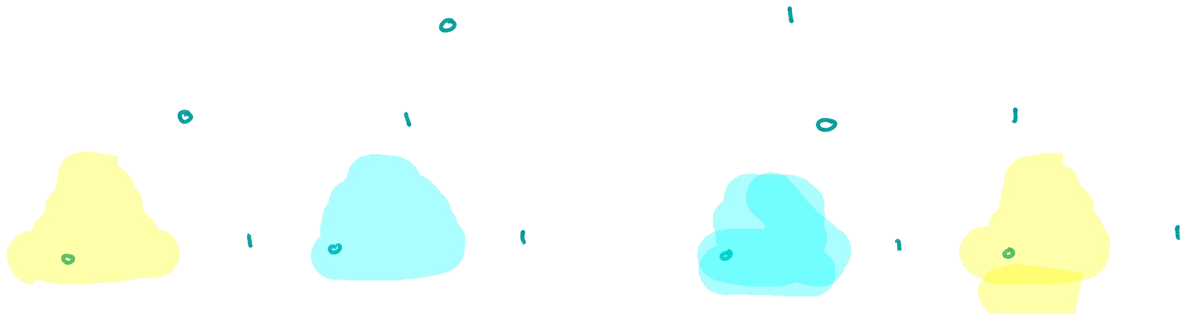
- Acyclic graph data structure for representing a boolean function (equivalently, a set of bit vectors)
- F(x,y,z) = (x=y)



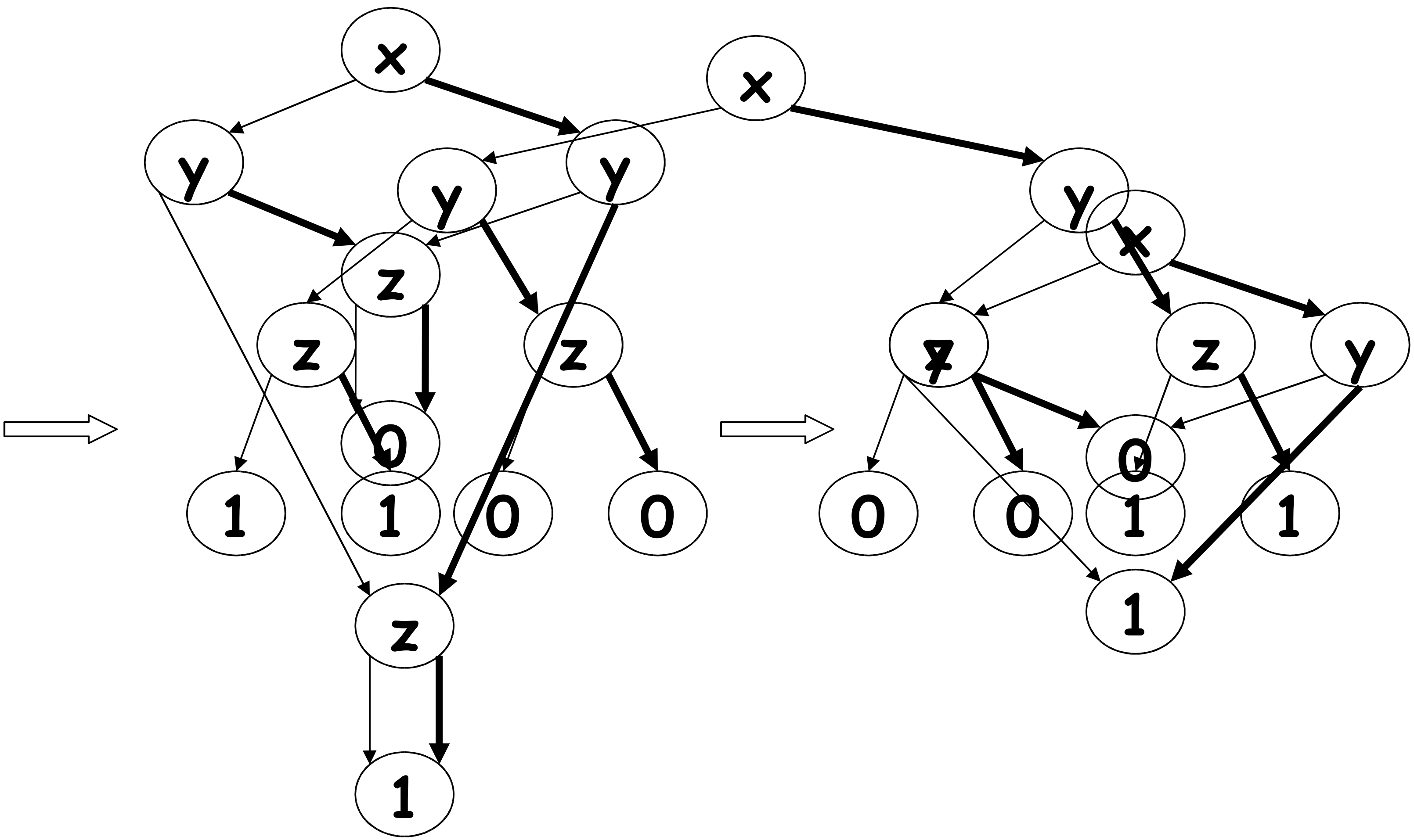
Binary Decision Diagrams

- Acyclic graph data structure for representing a boolean function (equivalently, a set of bit vectors)
- F(x,y,z) = (x=y)





Hash Consing + Variable Elimination



Aside

How to deal with

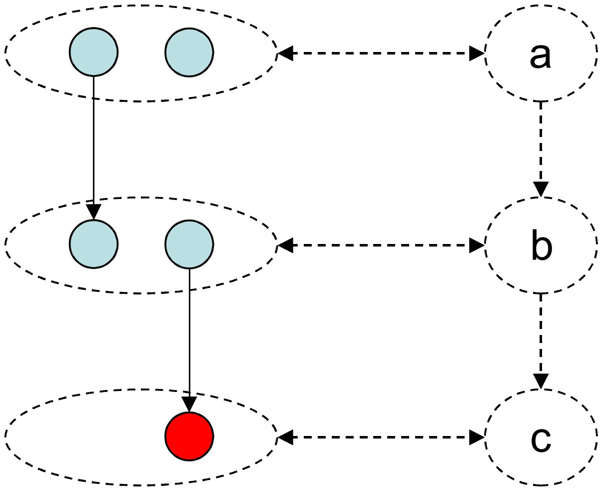
procedure calls i…

Prog…

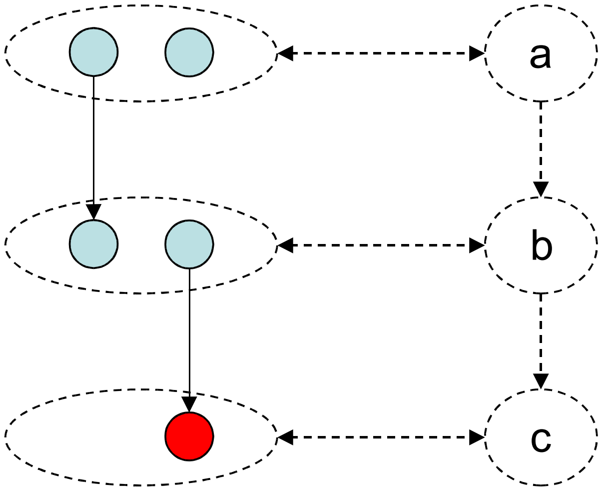
Part II: Overview

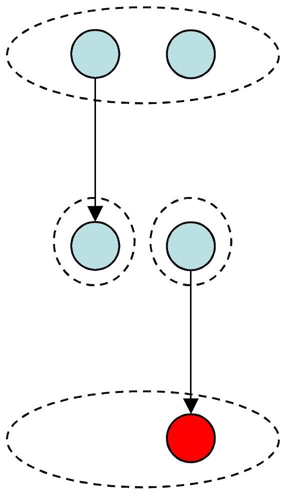
- Predicate Abstraction
- Symbolic Reachability with BDDs
- Predicate Refinement

Refinement



Refinement





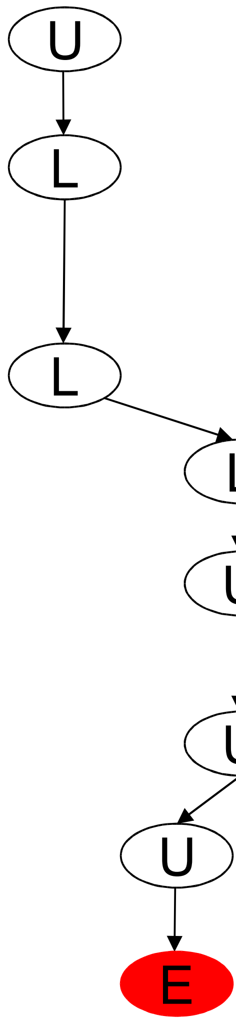
Algorithm for R

path = So . . . s

e = error state

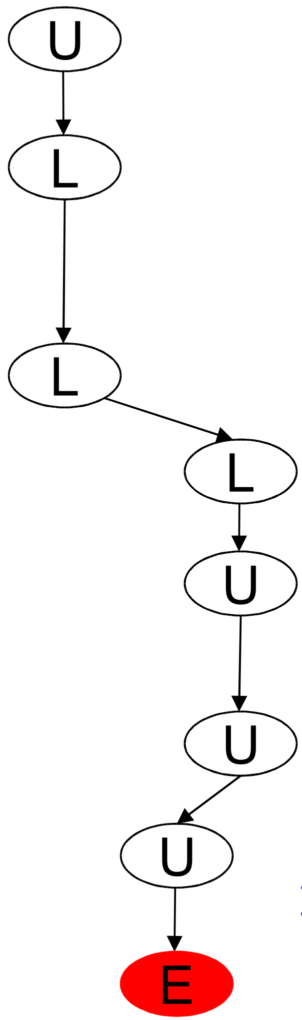
for i=k downto

Abstraction (via Boolean program)



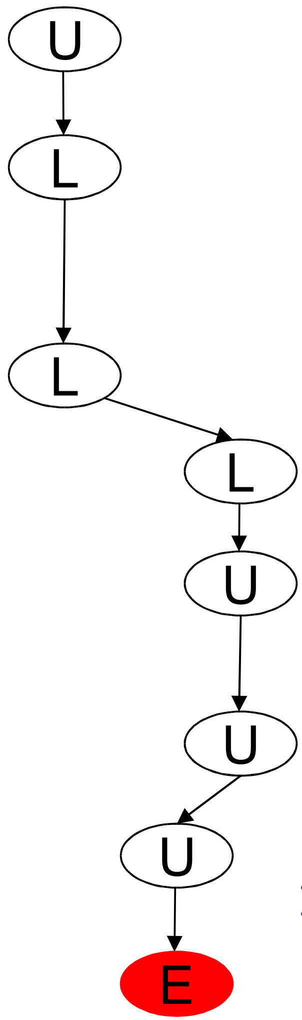

```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;

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

KeReleaseSpinLock();
```

Abstraction (via Boolean program)



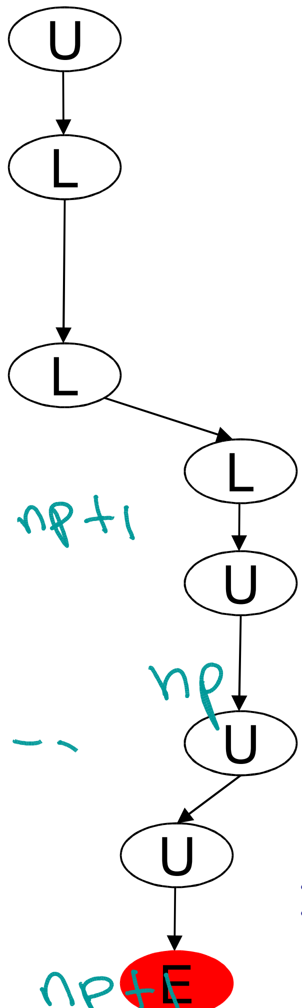

```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;

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

KeReleaseSpinLock();
```

Abstraction (via Boolean program)

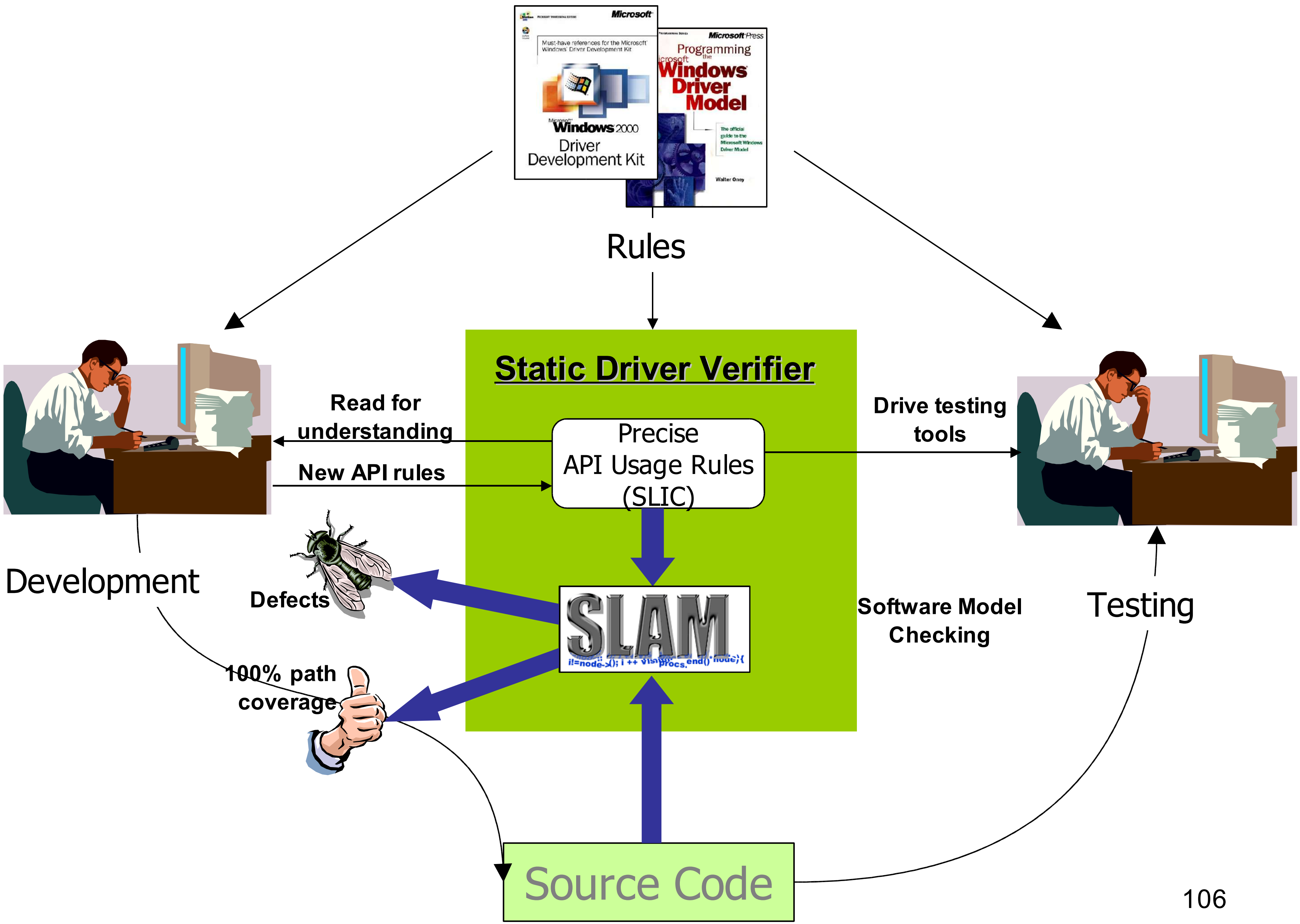



```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;

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

KeReleaseSpinLock();
```

Abstraction (via Boolean program)




```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;

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

KeReleaseSpinLock();
```


Microsoft
Windows 2000
Driver
Development Kit
Microsoft Press
Programming
Windows
Driver
Model
Rules
Static Driver Verifier
Read for
understanding
New API rules
Precise
API Usage Rules
(SLIC)
Drive testing
tools
Development
Defects
SLAM
Software Model
Checking
Testing
100% path
coverage
Source Code

Development Tools and Testing > Tools for Testing and Tuning

Static Driver Verifier - Finding Driver Bugs at Compile-Time

Static Driver Verifier (SDV) is a compile-time tool that explores code paths in a device driver by symbolically executing the source code. SDV is a unit-testing tool for Microsoft® Windows® device drivers based on Windows Driver Model (WDM) and Windows Driver Foundation (WDF).

SDV places a driver in a hostile environment and systematically tests all code paths by looking for violations of WDM usage rules. The symbolic execution makes very few assumptions about the state of the operating system or the initial state of the driver, so it can exercise situations that are difficult to exercise by traditional testing.

The set of rules packaged with SDV define how device drivers should use the WDM API. The categories of rules tested include the following.

Category	Rules tested for ...
IRP	Functions that use of I/O request packets
IRQL	Functions that use interrupt request levels
PnP	Plug and Play functions
PM	Power management
WMI	Functions using Windows Management Instrumentation
Sync	Synchronization related to spin locks, semaphores, timers, mutexes, and other methods of access control
Other	Functions that are not fully described by any of the other categories

Note: SDV is distributed as part of the WDF Beta program. To sign up for the WDF Beta

Tools and Testing
- Ordering Kits and Tools
- Windows DDK Overview
- Windows DDK FAQ
- Debugging Tools
- Tools for Testing and Tuning
- IFS Kit
- HCT Kit
- DCT Kit

Resources
- Support for Developers
- KB Articles for Drivers
- Which Windows DDK to Use

References
- Logo Requirements: B1.0
- WHQL Test Specs
- HCT Procedures
- DDK Online

Part III: Comparison

- Informal
- Formal

Informal Comparison

Abs tract Interpretati

- domain-specific
- large ma nual

efficient

Informal Comparison

Abs tract Interpretati

- domain-specific
- large ma nual

efficient

Informal Comparison

Predicate Abstraction

- domain-specific

- automati c abst

Formaly Comparing the Two Approaches

- WAIL
  - widening + abstract intepretation over infinite lattice
- FAIR
  - finite abstraction + iterative refinement

Abstraction/Refinement

- [Cousot-Cousot, PLILP'92]
  - widening + abstract interpretation with infinite lattices (WAIL) is more powerful than a (single) finite abstraction
- [Namjoshi/Kurshan, CAV'00]
  - if there is a finite (bi-)simulation quotient then WAIL with no widening will terminate [and therefore so will FAIR]
- [Ball-Podelski-Rajamani, TACAS'02]
  - finite abstractions plus iterative refinement (FAIR) is more powerful than WAIL

Guarded Command Language

- Variables $X = \{x_1, \ldots, x_n\}$
- Guarded command c
  - $g \wedge x_1' = e_1 \wedge \ldots \wedge x_n' = e_n$
- Program is a set of guarded commands
  - each command is deterministic
  - set of commands may be non-deterministic

Symbolic Representation of States

$$\varphi \equiv \bigvee_{i \in I} \bigwedge_{j \in J(i)} \varphi_{ij}$$

φ_{ij} : atomic formula such as $(x<5)$

$$\varphi' \leq \varphi \equiv \varphi' \Rightarrow \varphi$$

pre of

$$c \equiv g \;\wedge\; x_1' = e_1 \;\wedge \ldots \wedge\; x_n' = e_n$$

- $pre_c(\varphi) \;\equiv\; g \;\wedge\; \varphi[e_1, \ldots e_n / x_1, \ldots x_n]$
- $pre(\varphi) \;\equiv\; \bigvee_{c \in C} pre_c(\varphi)$

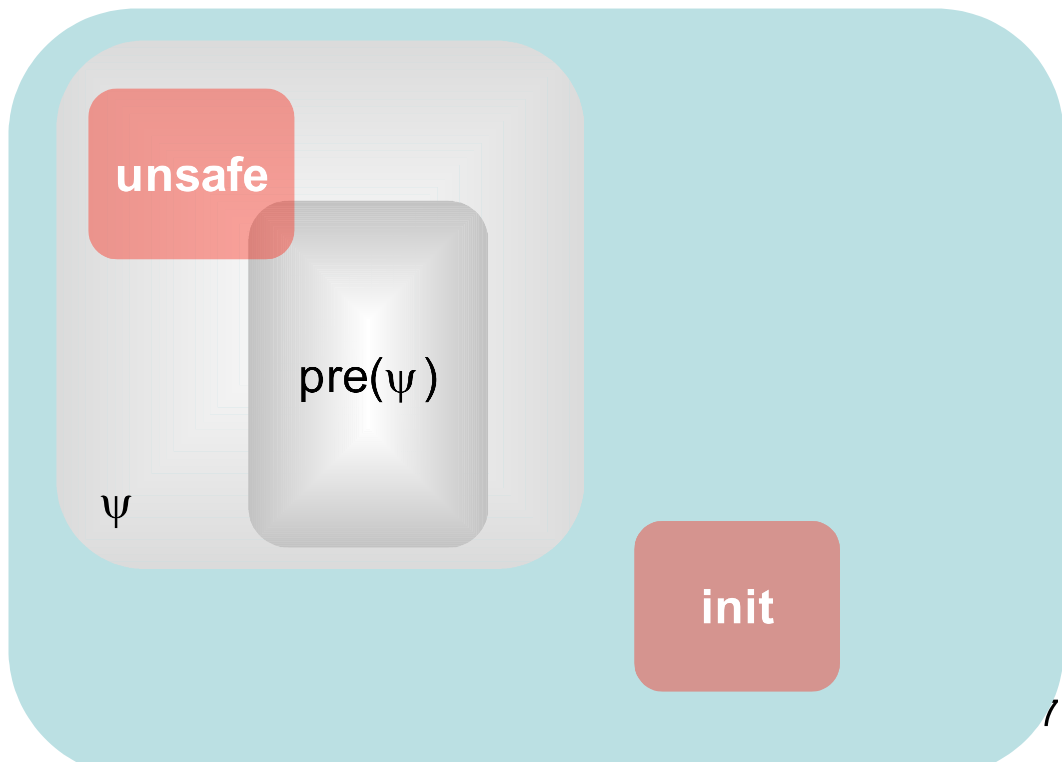
Safe Backward Invariants

∀ ψ is a safe backward invariant if

– unsafe $\Rightarrow \psi$

– pre(ψ) $\Rightarrow \psi$

’ $\psi \Rightarrow$ noninit

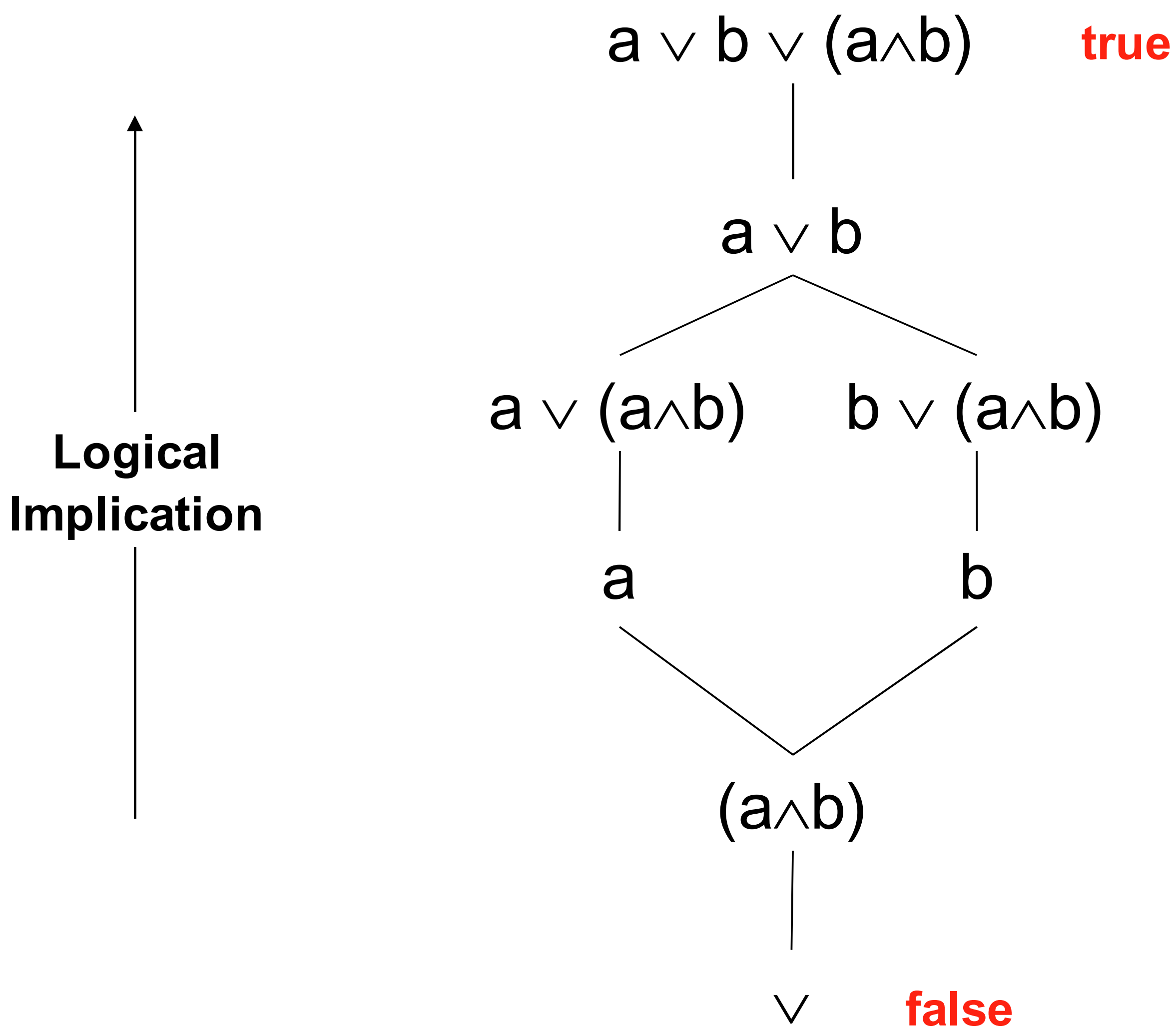


Predicate Abstraction

- A set P of predicates over a program's state space defines an abstraction of the program
  - P = { (a=1), (b=1), (a>0) }
  - Uninterpreted **atoms** [a=1][b=1][a>0]

- If P has n predicates, the abstract domain contains exactly 2^{2^n} elements
  - an abstract state = conjunction ($\wedge$) of atoms
  - a set of abstract states = disjunction ($\vee$) of abstract states

Free Lattice of DNF over {a,b}



$pre^{\#}_P \equiv \alpha_P \; pre \; \gamma$

$\forall \; \gamma \equiv$ the identity function

$\forall \; \alpha_P(\varphi) \equiv$ the least φ' such that $\varphi \leq \gamma \; \varphi'$

- Example:
  - P = { (x<2), (x<3), (x=0) }
  - ’ α_P(x=1) = $(x<2) \wedge (x<3)$

FAIR

$n := 0$; $\varphi :=$ **unsafe**
loop
  $P_n := \text{atoms}(\varphi)$
  construct **pre**$^{\#}_{n}$, as defined by P_n

  $\psi := \text{lfp}($**pre**$^{\#}_{n}$, **unsafe**$)$

  if $(\psi \leq$ **noninit**$)$ **then**
    return "success"

  $\varphi := \varphi \vee$ **pre**(φ);

  $n := n + 1$;
forever

Widening

- widen(φ) = φ' such that $\varphi \leq \varphi$'

- We consider widening that simply drops terms from some conjuncts

$$\text{widen}\left(\bigvee_{i \in I} \bigwedge_{j \in J(i)} \varphi_{ij}\right) =$$

$$\bigvee_{i \in I} \bigwedge_{j \in \mathbf{J'(i)}} \varphi_{ij} \qquad \text{where } \mathbf{J'(i)} \subseteq J(i)$$

- Results can be extended to other classes of widenings

Interval Widening, Revisited

$$[l_0, \quad u_0] \quad \nabla \quad [l_1,$$

$$l_0 \leq x \quad \wedge \quad x \leq u_0$$

WAIL

```
n:= 0;  φ :=  unsafe;  old := false;
loop
   if (φ ≤ old) then
      if (φ ≤ noninit) then
         return "success"
      else
         return "Don't know"
   else
      old := φ
      i     := guess provided by oracle
      φ     := widen(i, φ ∨ pre(φ) )
      n     := n+1
forever
```

FAIR	WAIL

FAIR

```
n := 0; φ := unsafe
loop
   P_n := atoms(φ)
   construct pre#_n, as defined by P_n

   ψ := lfp(pre#_n, unsafe)
   if (ψ ≤ noninit) then
      return "success"

   φ := φ ∨ pre(φ);

   n := n + 1;
forever
```

WAIL

```
n:= 0;  φ :=  unsafe;  old := false;
loop
   if (φ ≤ old) then
      if (φ ≤ noninit) then
         return "success"
      else
         return "Don't know"
   else
      old := φ
      i   := guess provided by oracle
      φ   := widen(i, φ ∨ pre(φ) )
      n   := n+1;
forever
```

Theorem. For any program P, if WAIL terminates with success for some sequence of widening choices, then FAIR will terminate with success as well.

- Lemma 1: If a safe invariant ψ can be expressed in terms of predicates in P then $\mathrm{lfp}(\mathrm{pre}^{\#}_{P}, \mathrm{unsafe})$ is a safe invariant

- Lemma 2: For any guarded command c,

  $\mathrm{pre}_c(\varphi \vee \varphi') = \mathrm{pre}_c(\varphi) \vee \mathrm{pre}_c(\varphi')$

  $\mathrm{pre}_c(\varphi \wedge \varphi') = \mathrm{pre}_c(\varphi) \wedge \mathrm{pre}_c(\varphi')$

- Corollary: For any guarded command c,

  $\mathrm{atoms}(\mathrm{pre}_c(\varphi \vee \varphi')) = \mathrm{atoms}(\mathrm{pre}_c(\varphi)) \cup \mathrm{atoms}(\mathrm{pre}_c(\varphi'))$

  $\mathrm{atoms}(\mathrm{pre}_c(\varphi \wedge \varphi')) = \mathrm{atoms}(\mathrm{pre}_c(\varphi)) \cup \mathrm{atoms}(\mathrm{pre}_c(\varphi'))$

Proof of Theorem

$\varphi_0 = \text{unsafe}$ $\qquad$ $\varphi'_0 = \text{unsafe}$

$\varphi_{n+1} = \varphi_n \vee$ **pre**(φ_n) $\qquad$ $\varphi'_{n+1} = \text{widen}(\varphi'_n \vee$ **pre**$(\varphi'_n))$

for all i, $\text{atoms}(\varphi_i) \supseteq \text{atoms}(\varphi'_i)$

by induction on i and Lemma 2

if φ'_i is a safe inv. then

by Lemma 1 and above result

$\text{lfp}(F^{\#}_{\text{atoms}(\varphi_i)}, \text{start})$ is a safe inv.

Summary

- Predicate abstraction + refinement and widening can be formally related to each other

- Predicate abstraction + refinement = widening with “optimal” guidance

What We Did

- Part I: Abstract Interpretation
  - [Cousot & Cousot, POPL'77]
  - *Manual abstraction and refinement*
  - ASTRÉE Analyzer
- Part II: Predicate Abstraction
  - [Graf & Saïdi, CAV '97]
  - *Automated abstraction and refinement*
  - SLAM and Static Driver Verifier
- Part III: Comparing Approaches

Searching for Solutions

- Once upon a time, only a human could play a great game of chess…
  - … but then smart brute force won the day

- Once upon a time, only a human could design a great abstraction…