Automatic Software Verification

Instructor: Mooly Sagiv

TA: Oded Padon

Slides from Eran Yahav and the Noun Project, Wikipedia

Course Requirements

- Summarize one lecture
- 10% one lecture notes
- 45% homework assignment
- 45% exam/project

Inspired by Other Courses

- CS395T: Automated Logical Reasoning (UT Austin) Isil Dillig
- SAT/SMT Solver and Applications Graduate Seminar W2013 University of Waterloo (Vijay Ganesh)

Software is Everywhere






Exploitable

Software is Everywhere

Exploitable Software is Everywhere

Sony PlayStation

RSA hacked, information leaks

Stuxnet Worm Still Out of Control at Iran's Nuclear Sit

Security Advisory for Adobe Flash Player, Adobe Read

RSA tokens may be behind major network security problems at

Buffer Overrun

```
void foo (char *x) {
  char buf[2];
  strcpy(buf, x);
}
int main (int argc, char *argv[]) {
  foo(argv[1]);
}
```

```
> ./a.out
abracadabra
```

source code

```
Segmentation
fault
```

terminal

memory

Buffer Overrun Exploits

```
int check_authentication(char *password) {
 int auth_flag = 0;
 char password_buffer[16];

 strcpy(password_buffer, password);
 if(strcmp(password_buffer, "brillig") == 0) auth_flag = 1;
 if(strcmp(password_buffer, "outgrabe") == 0) auth_flag = 1;
  return auth_flag;
}
int main(int argc, char *argv[]) {
   if(check_authentication(argv[1])) {
      printf("\n-=-=-=-=-=-=-=-=-=-=-=-=-=-\n");
      printf("      Access Granted.\n");
      printf("-=-=-=-=-=-=-=-=-=-=-=-=-=-\n");          }
  else
      printf("\nAccess Denied.\n");
}
```

(source: “hacking – the art of exploitation, 2nd Ed”)

Attack

evil input

Application

AAAAAAAAAAAAA

-=-=-=-=-=-=-=-=-=-=-=-=-=-

Access Granted. 65

-=-=-=-=-=-=-=-=-=-=-=-=-=-

Automatic Program Verification

Program
P

Desired
Properties φ

Solver

Is there a behavior
of P that violates φ?

Counterexample

Proof

Example

```
    int check_authentication(char *password) {
     int auth_flag = 0;
     char password_buffer[16];

     strcpy(password_buffer, password);
     if(strcmp(password_buffer, "brillig") == 0) auth_flag = 1;
     if(strcmp(password_buffer, "outgrabe") == 0) auth_flag = 1;
      return auth_flag;
}
int main(int argc, char *argv[]) {
    if(check_authentication(argv[1])) {
        printf("\n-=-=-=-=-=-=-=-=-=-=-=-=-=-=-\n");
        printf("      Access Granted.\n");
        printf("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-\n");           }
   else
        printf("\nAccess Denied.\n");
}
```

Undecidability

- The Halting Problem
  - Does the program P terminate on input I
- Rice's Theorem
  - Any non-trivial property of partial functions, there is no general and effective method to decide if program computes a partial function with that property

Handling Undecidability

- Permits occasional divergence
- Limited programs (not Turing Complete)
- Unsound Verification
  - Explore limited program executions
- Incomplete Verification
  - Explore superset of program executions
- Programmer Assistance
  - Inductive loop invariants

Limited Programs

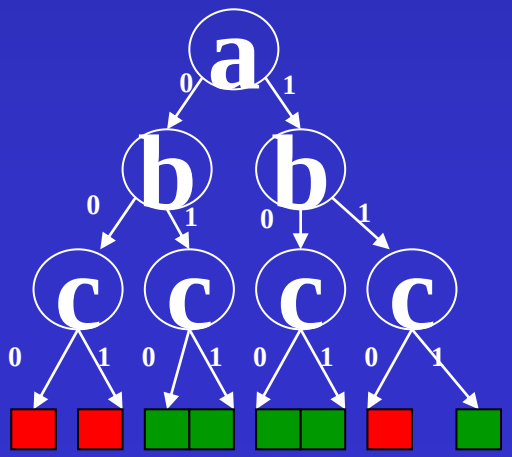
- Finite state programs
  - Finite state model checking
    - Explicit state SPIN, CHESS
    - Symbolic model checking SMV
- Loop free programs
  - Configuration files

Unsound Verification

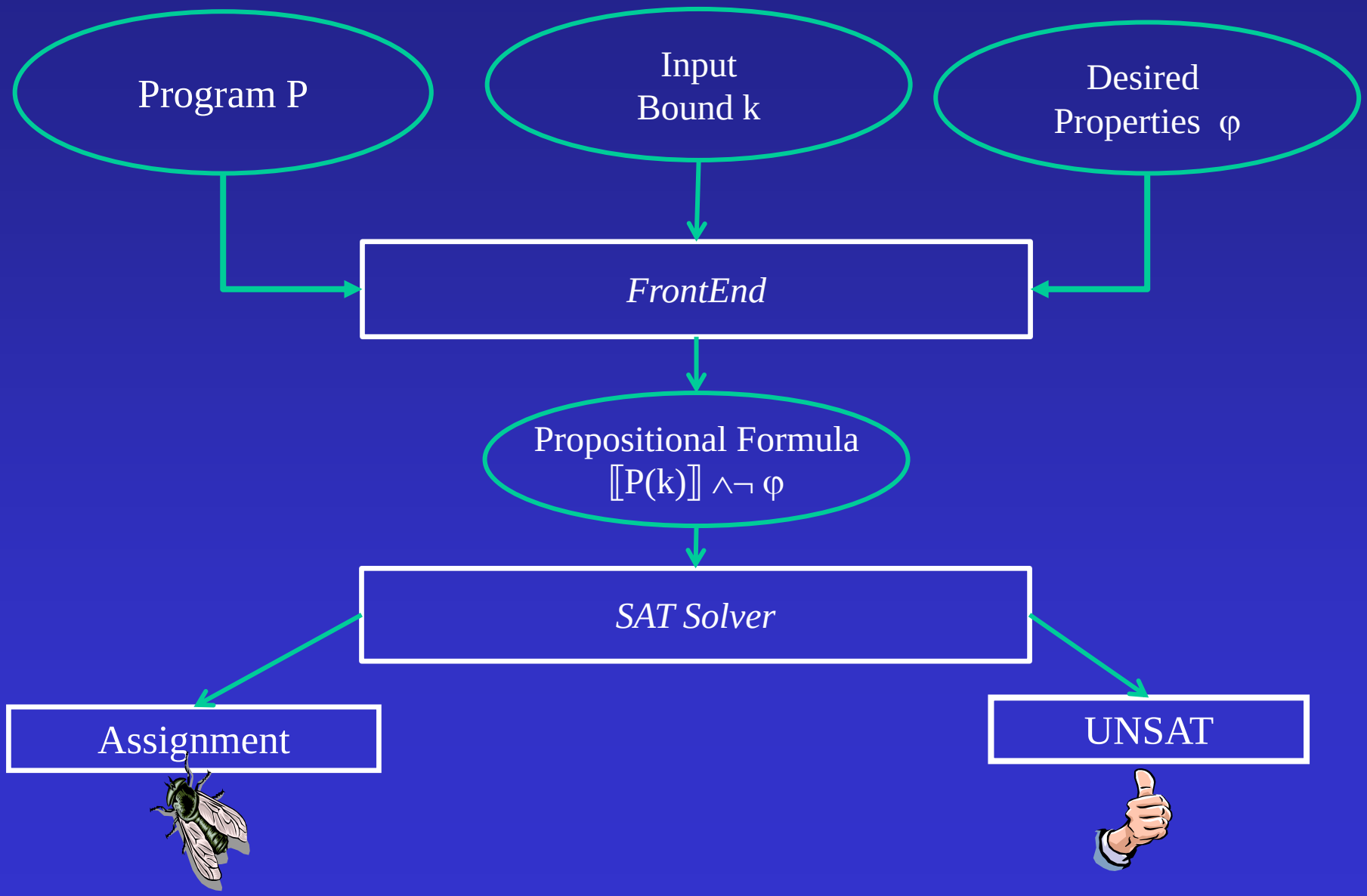
- Dynamic checking
  - Valgrind, Parasoft Insure, Purify, Eraser
- Bounded Model Checking
- Concolic Executions

The SAT Problem

- Given a propositional formula (Boolean function)
  - $\varphi = (a \vee b) \wedge (\neg a \vee \neg b \vee c)$
- Determine if φ is valid
- Determine if φ is satisfiable
  - Find a satisfying assignment or report that such does not exit
- For n variables, there are 2^n possible truth assignments to be checked
- Effective heuristics exit



Bounded Model Checking



A Simple Example

Program

```
int x;
int y=8,z=0,w=0;
if (x)
  z = y - 1;
else
  w = y + 1;
assert (z == 5 ||
        w == 9)
```

Constraints

y = 8,
z = x ? y – 1 : 0,
w = x ? 0 :y + 1,
z != 5,
w != 9

SAT
counterexample found!

y = 8, x = 1, w = 0, z = 7

A Simple Example

Program

```
int x;
int y=8,z=0,w=0;
if (x)
  z = y - 1;
else
  w = y + 1;
assert (z == 7 ||
        w == 9)
```

Constraints

$y = 8,$
$z = x \ ? \ y - 1 : 0,$
$w = x \ ? \ 0 : y + 1,$
$z \mathrel{!=} 7,$
$w \mathrel{!=} 9$

UNSAT
Assertion always holds!

Summary Bounded Model Checking

- Excellent tools exist (CBMC, Alloy)
- Many bugs occur on small inputs
- Useful for designs too
- Scalability is an issue
- Challenging features
  - Bounded arithmetic
  - Pointers and Heap
  - Procedures
  - Concurrency

Concolic Testing

- Combine runtime testing and symbolic execution
- Runtime testing
  - Effectiveness depends on input test
- Symbolic Execution

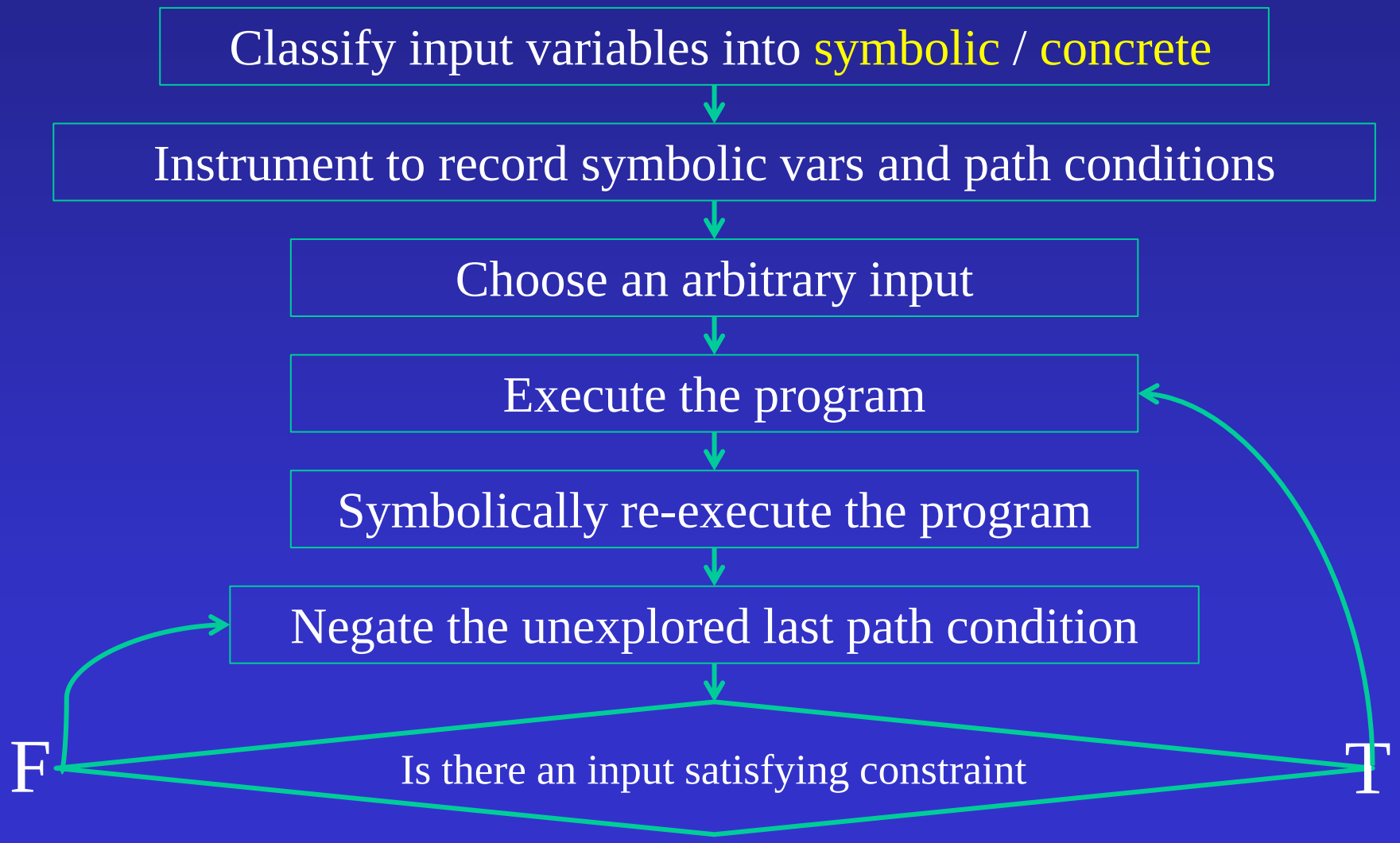
```
read(x);
y = 2 * x ;
assert y != 12;
```

  - Need constraint solver
  - Can be complex
- Concolic testing aims to improve both

A Motivating Example

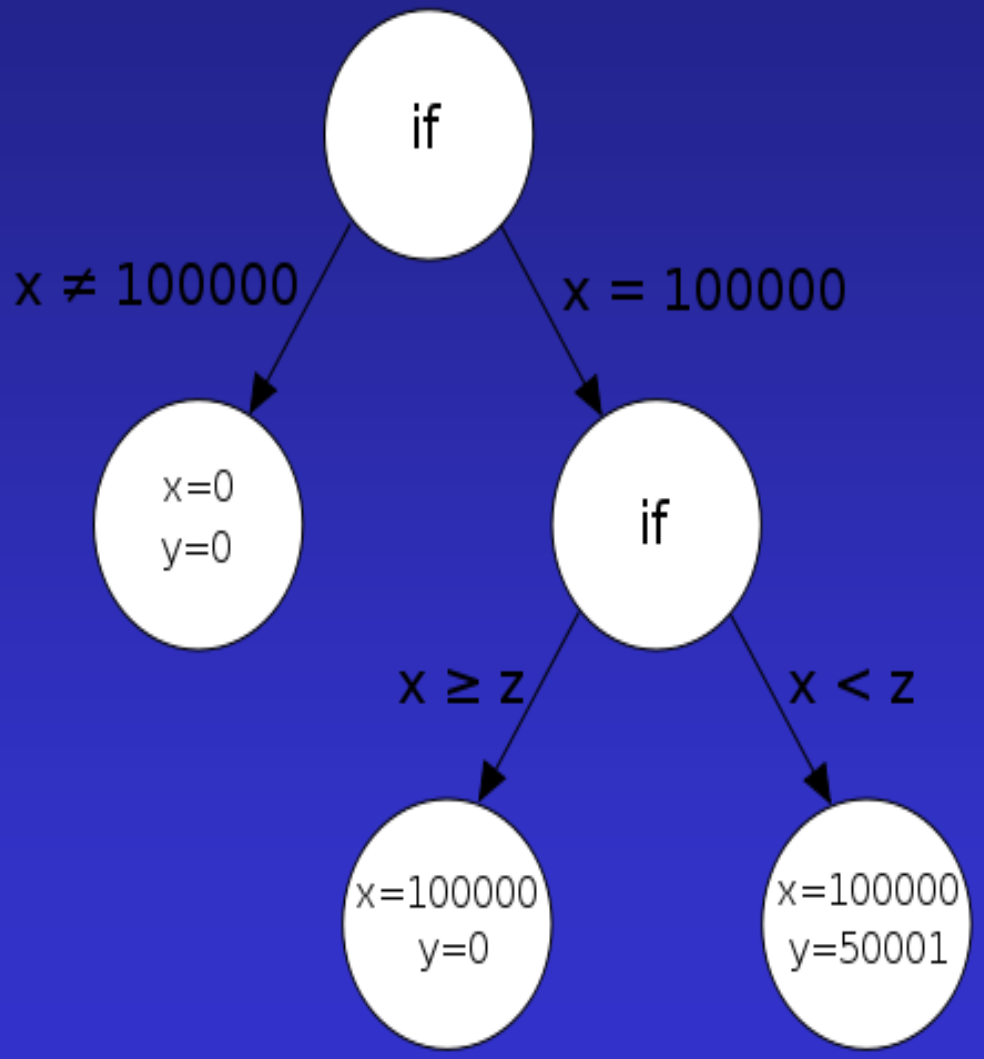
```
void f(int x, int y) {
    int z = 2*y;
    if (x == 100000) {
        if (x < z) {
            assert(0); /* error */
        }
    }
}
```

The Concolic Testing Algorithm



Example Concolic Testing

```
void f(int x, int y) {
    int z = 2*y;
    if (x == 100000) {
        if (x < z) {
            assert(0); /* error */
        }
    }
}
```



Summary Concolic Testing

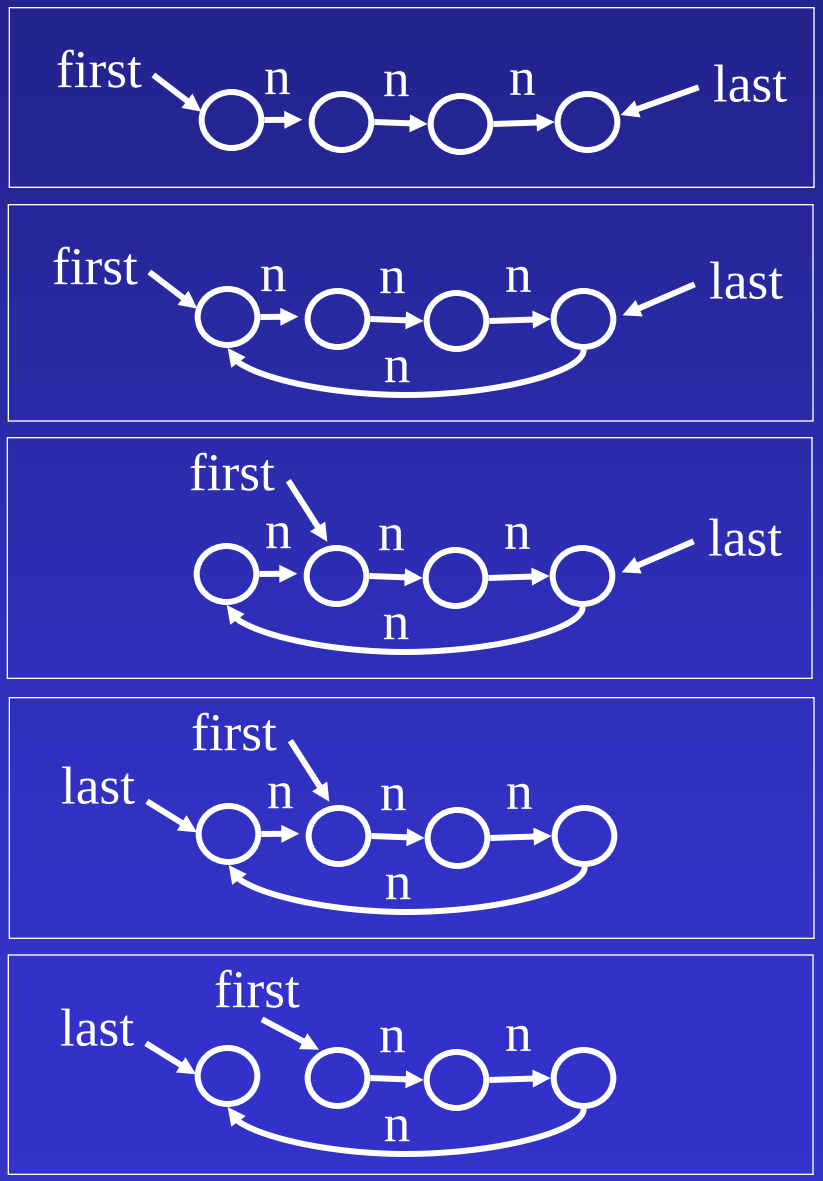
- Quite effective:
  - SAGE (Microsoft Research)
  - Datarace detection (Candea, EPFL)
- Instrumentation can be tricky
- Scalability is an issue
- Coverage is an issue
- Limitations of theorem provers
- Data structures

Invariant

- An assertion I is an invariant at program location if I holds whenever the execution reaches this location
- An invariant is inductive at a loop "while B do C" if whenever C is executed on a state which satisfies B and I it can only produce states satisfying I

Temporary Cycle Creation

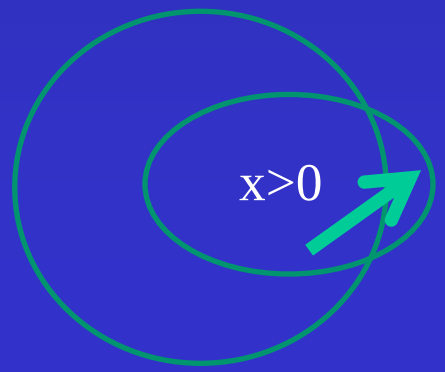
```
rotate(List first, List last) {
    if ( first != NULL) {
        last → next = first;
        first = first → next;
        last = last → next;
        last → next = NULL;
    }
}
```



Inductive Invariants

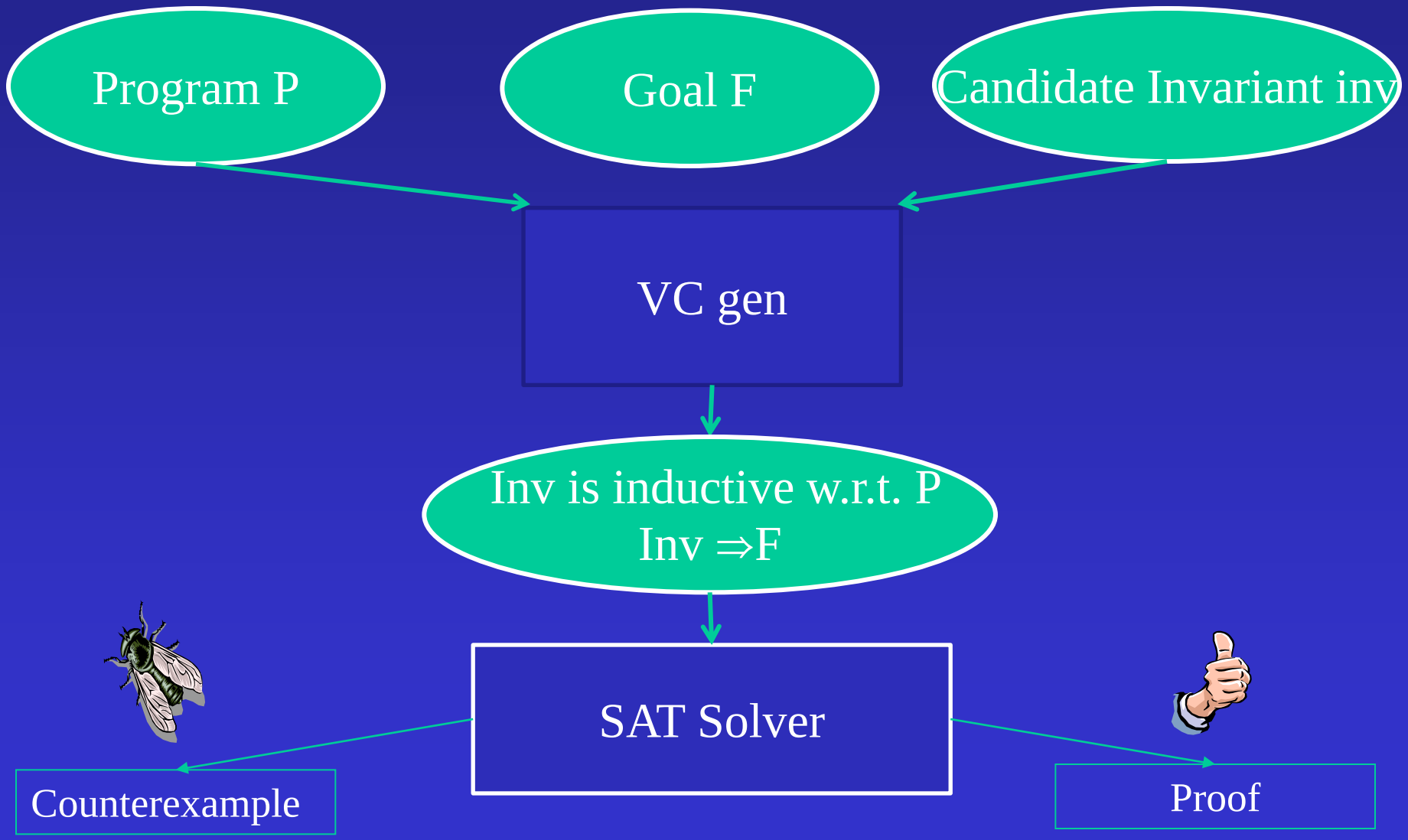
```
x = 2;
while true do {x >0}
        x = 2* x - 1
```

Non-inductive



Inductive

Deductive Verification



Summary Deductive Verification

- Existing Tools
  - ESCJava, Dafny, CAVEAT
- Hard to write inductive invariants
  - Need to consider all corner cases
  - Small program change can lead to huge change in the invariant
  - The lack of specification languages
- Deduction can be hard

Deduction

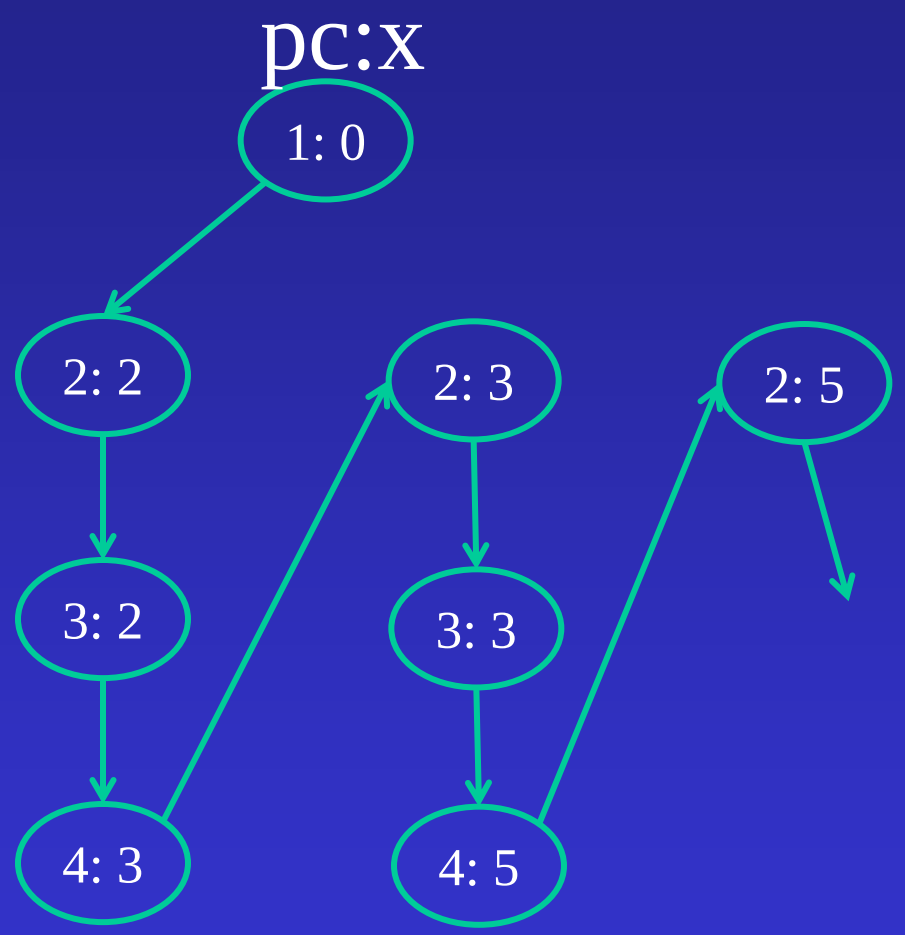
```
x = 2;
while true do {x >1}
        x = (2*x*x + x – 1) / (x + 1)
```

Transition Systems

- The program semantics can be described as (potentially infinite) graph of reachable states
  - Values of program variables
- Program statements and conditions are relations between states
- Proving a safety property usually means showing that certain state cannot be reached
  - A bad reachable state indicate a bug
- Bounded model checking and concolic testing explore subsets of reachable states

Example Transition System

1: x = 2;
2: while true do
    3: x = 2* x – 1
    4:



Abstract Interpretation

- Automatically prove that the program is correct by also considering infeasible executions
- Abstract interpretation of program statements/conditions
- Conceptually explore a superset of reachable states
- Sound but incomplete reasoning
- Automatically infer sound inductive invariants

Automatic Program Verification

Program
P

Desired
Properties φ

Solver

Is there a behavior of P that violates φ?

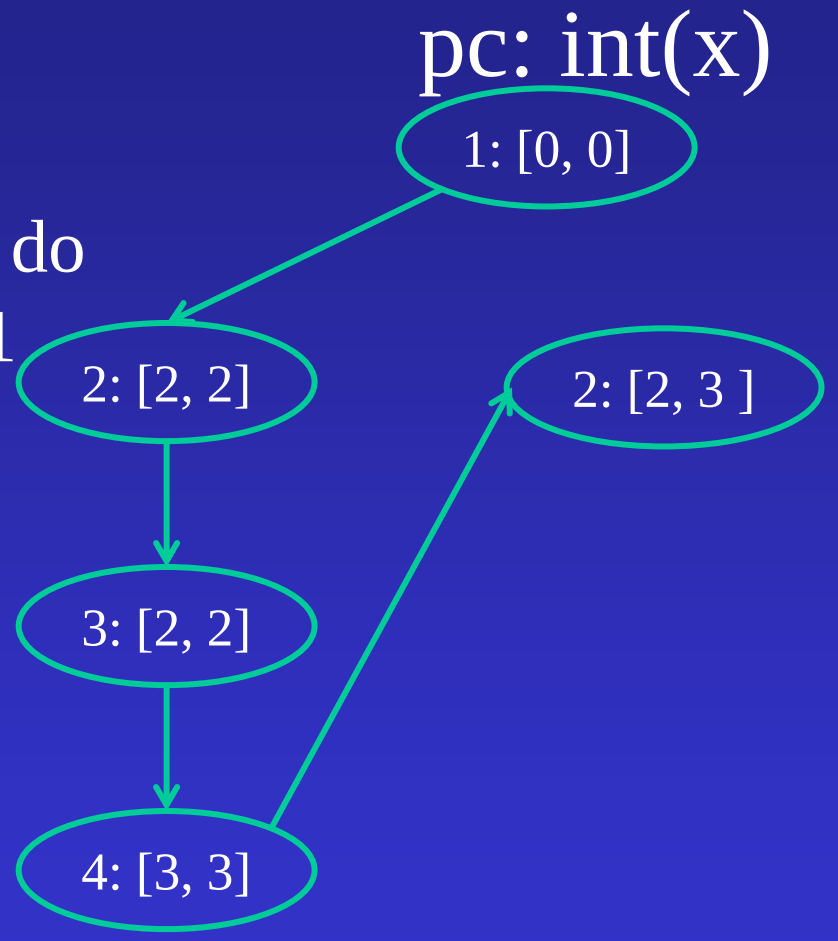
Counterexample

Unknown

Proof

Interval Based Abstract Interpretation

1: x = 2;
2: while true {x > 0} do
3: x = 2* x – 1
4:



Interval Based Abstract Interpretation

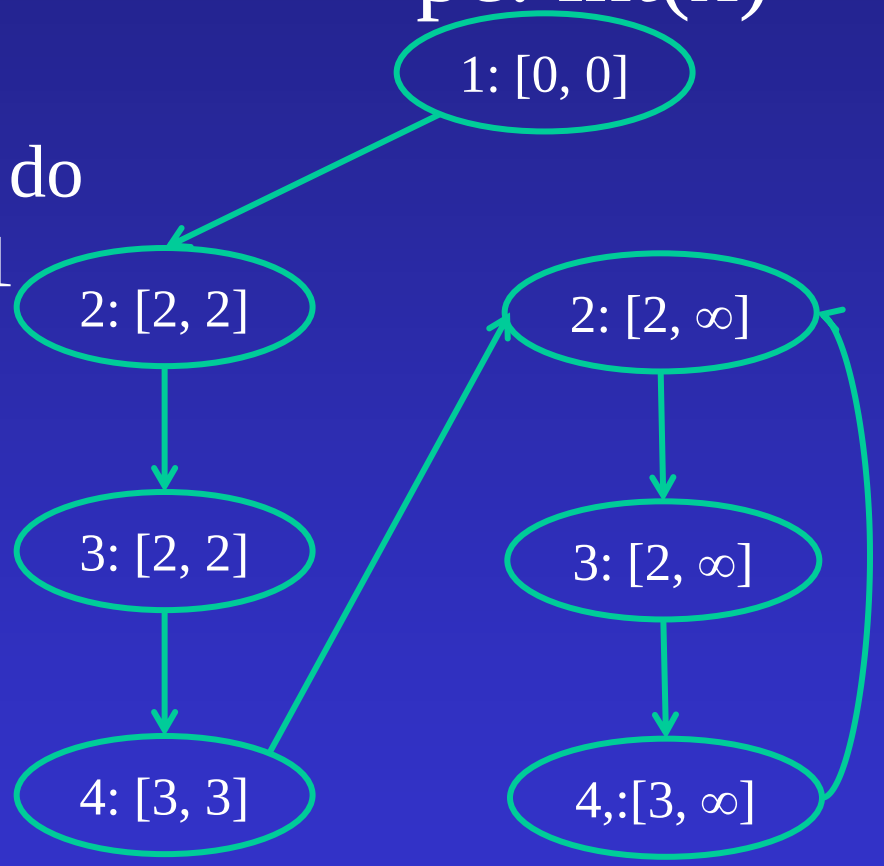
pc: int(x)

1: x = 2;

2: while true {x > 0} do

3: x = 2* x – 1

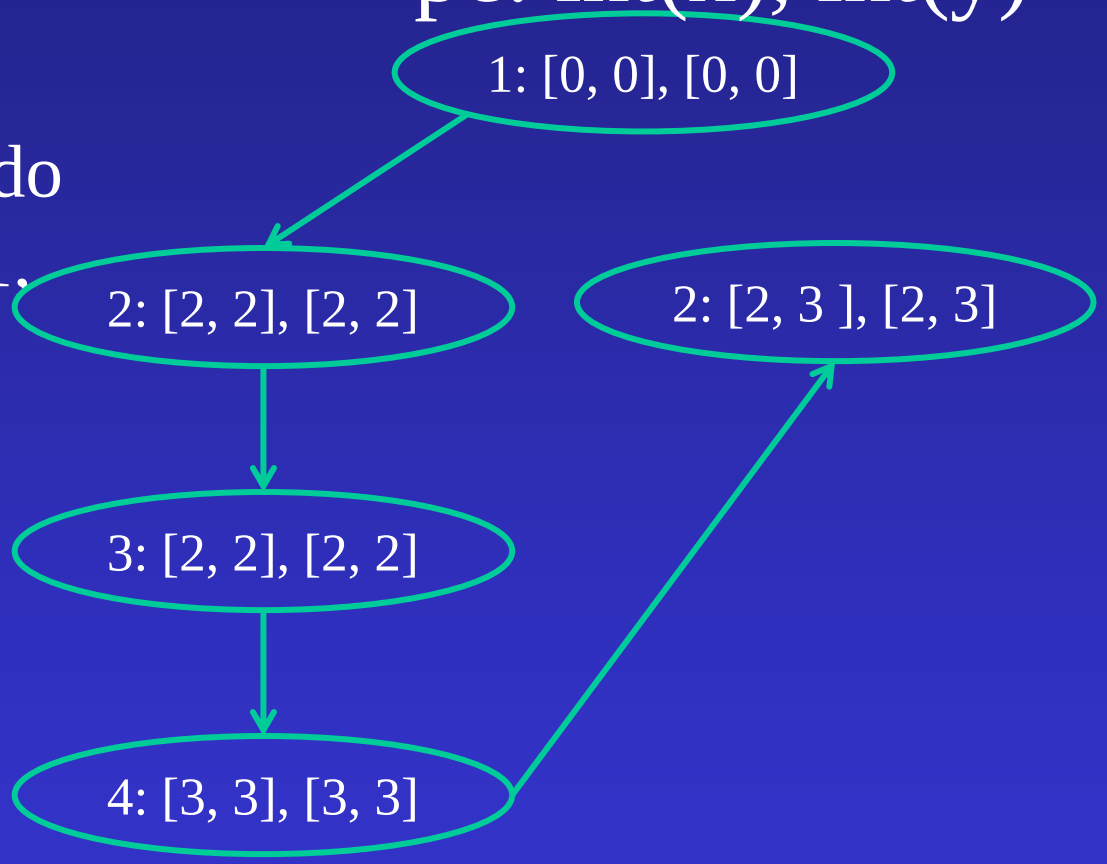
4:



Interval Based Abstract Interpretation

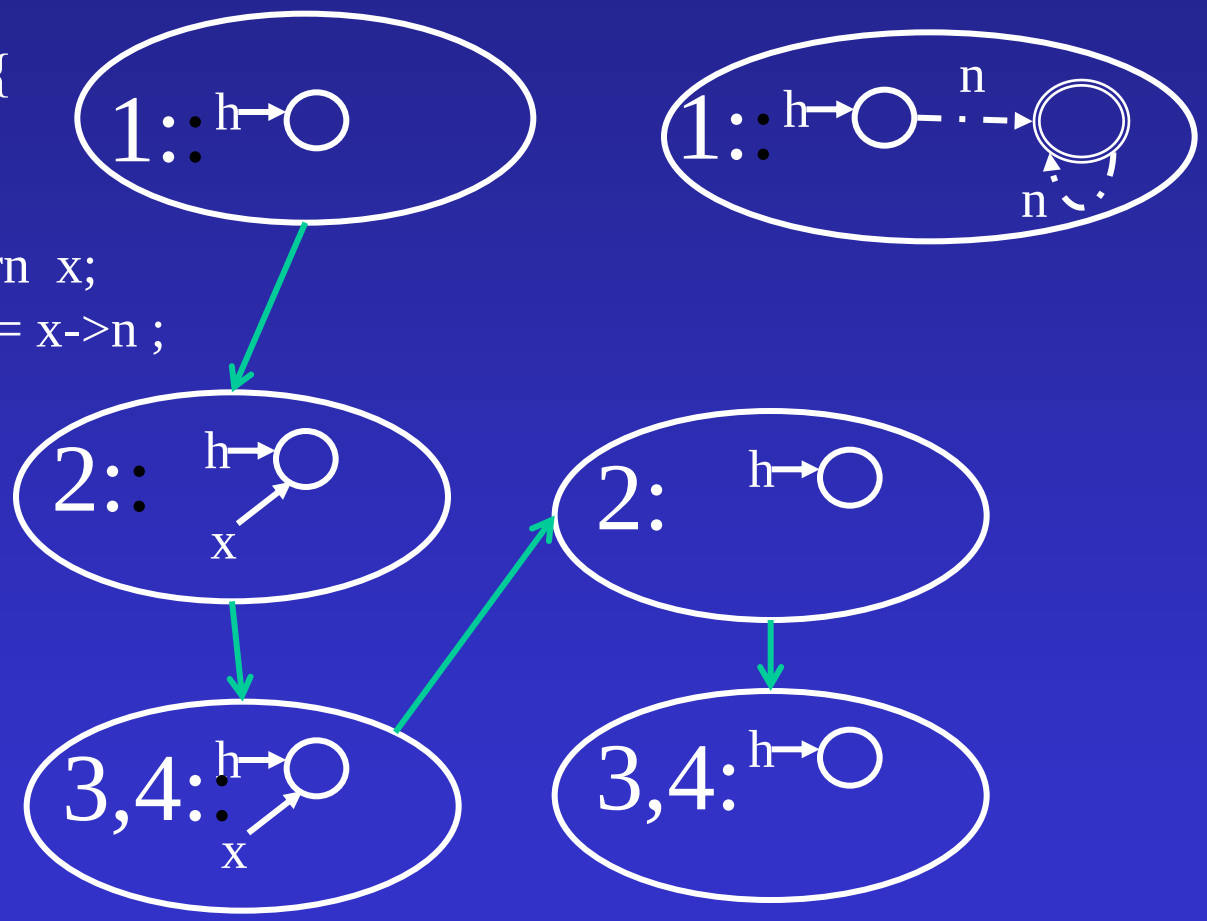
pc: int(x), int(y)

1: x = 2, y = 2
2: while true {x =y} do
3: x = 2* x – 1,
y = 2*y -1
4:



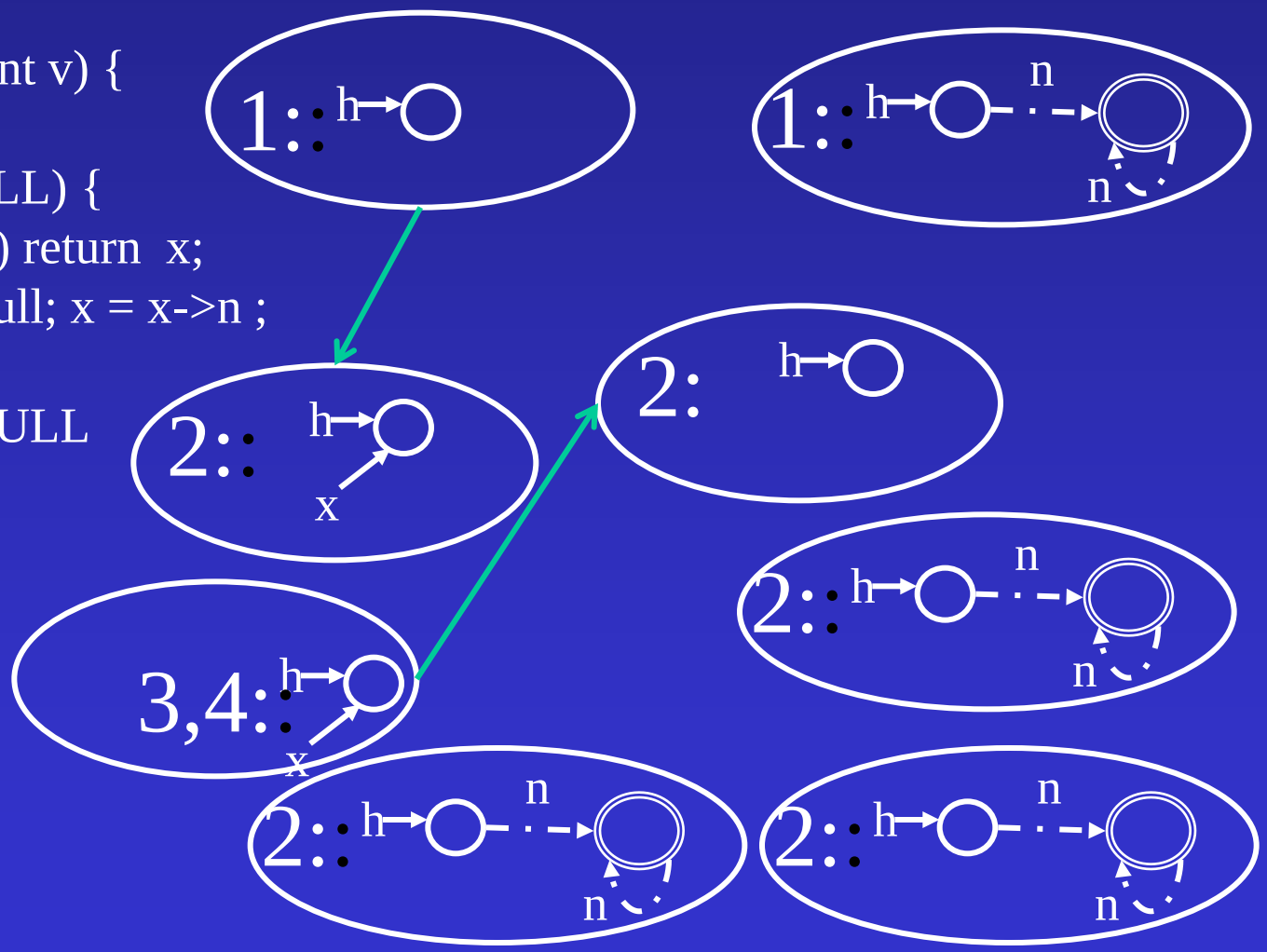
Shape-Based Abstract Interpretation

```
node search(node h, int v) {
   1:  node x = h;
   2:  while (h != NULL) {
       3:  if (x->d == v) return  x;
       4:  assert  x != null; x = x->n ;
   }
   5:   return (node) NULL
```



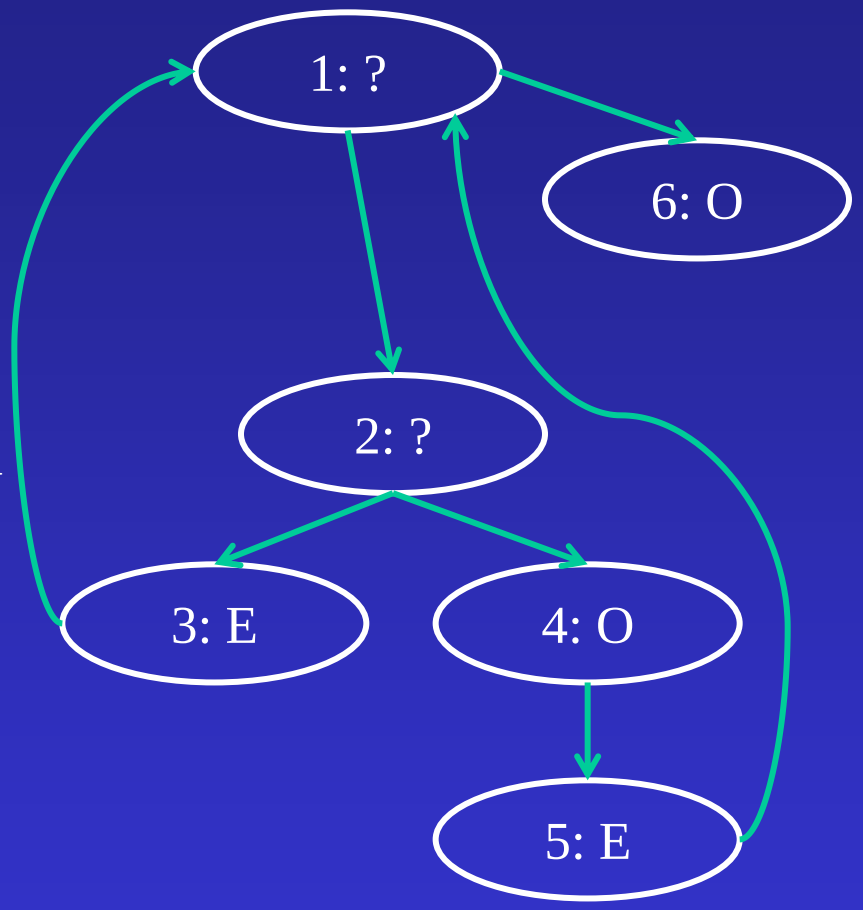
Shape-Based Abstract Interpretation

```
node search(node h, int v) {
   1:  node x = h;
   2:  while (x != NULL) {
       3:  if (x->d == v) return  x;
       4:  assert  x != null; x = x->n ;
   }
   5:   return (node) NULL
```

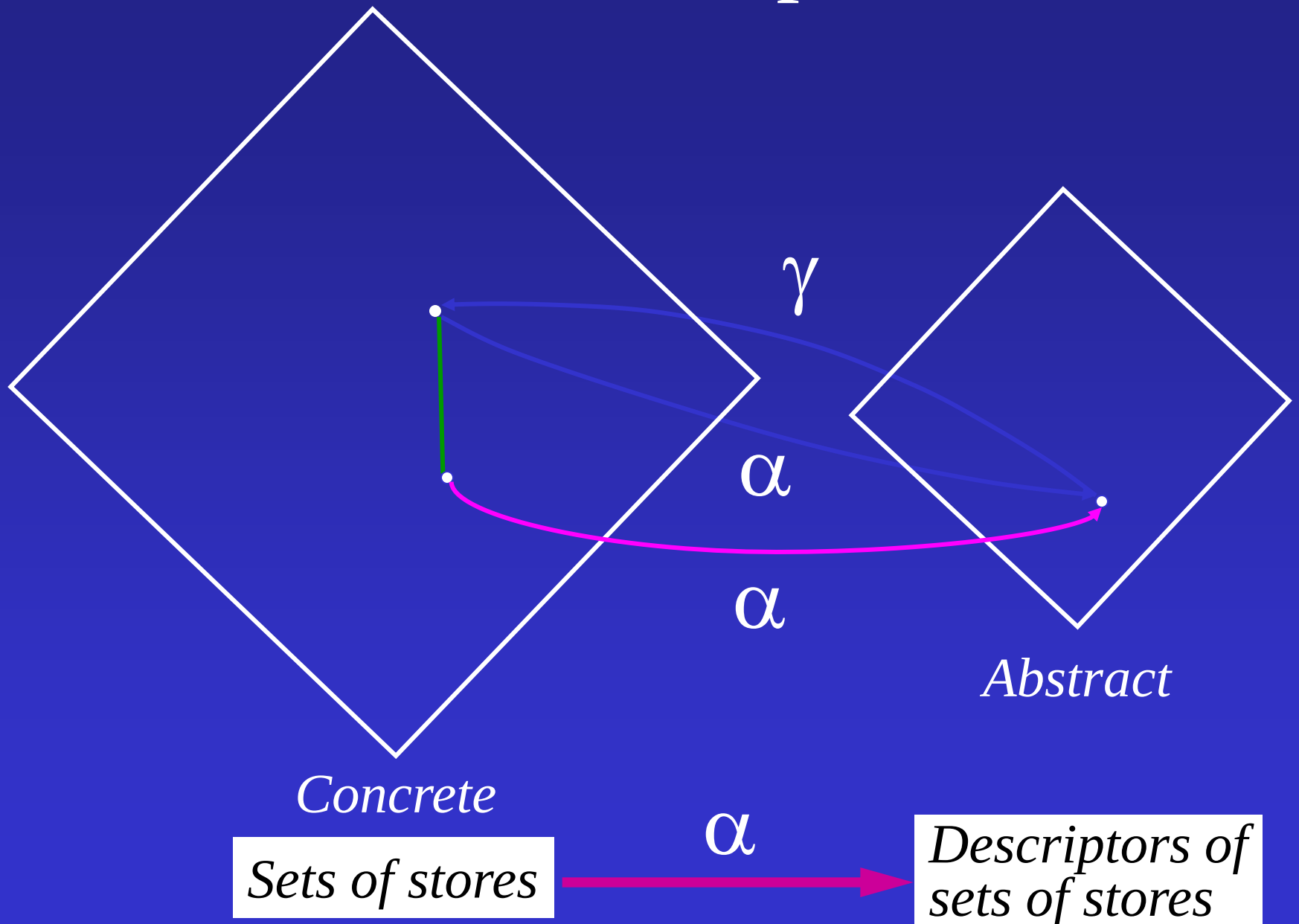


Odd/Even Abstract Interpretation

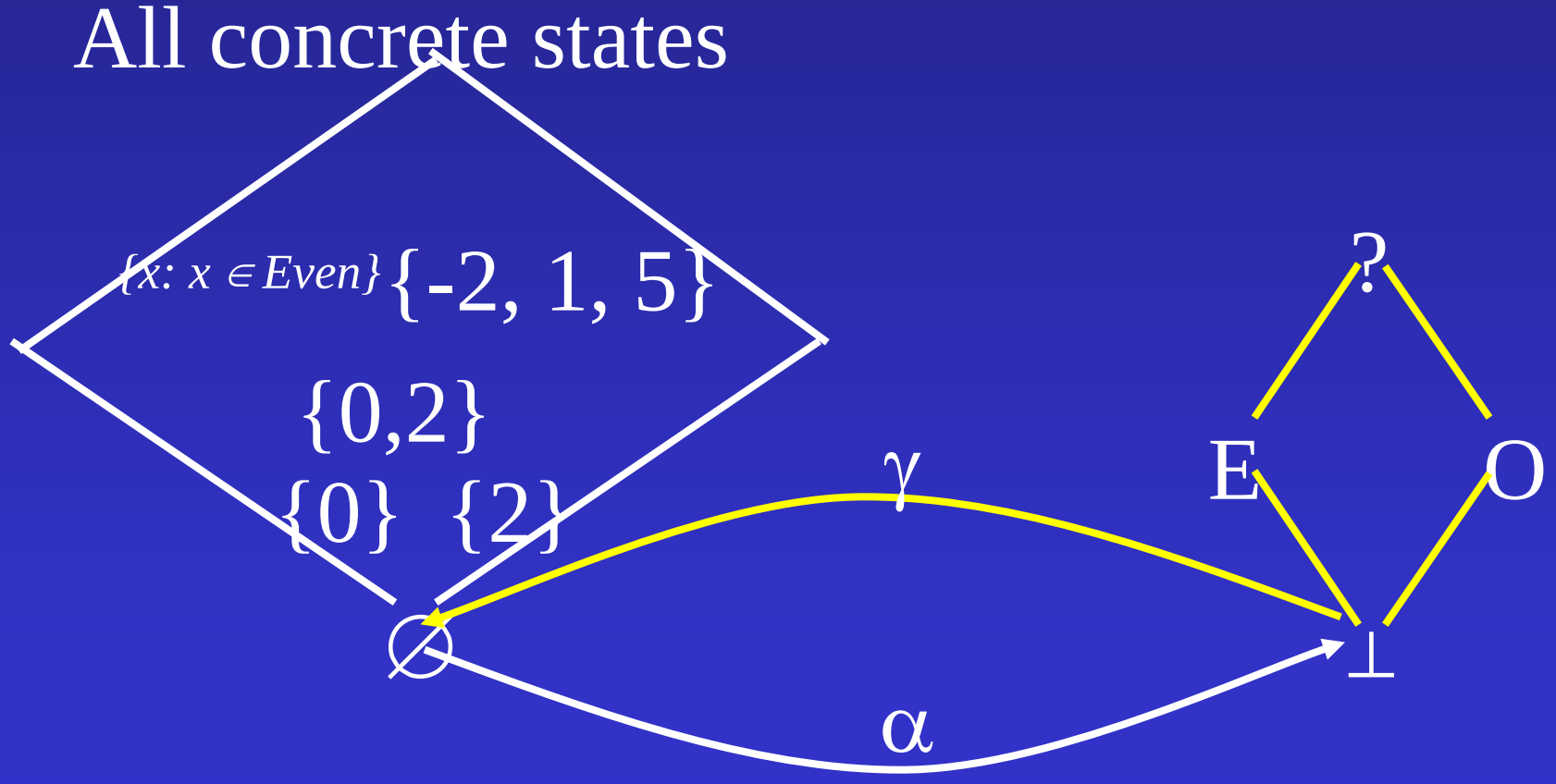
```
1: while  (x !=1)   do {
        2: if   (x %2) == 0
            { 3: x := x / 2; }
             else
            { 4 : x := x * 3 + 1;
           5: assert (x %2 ==0); }
6: }
```



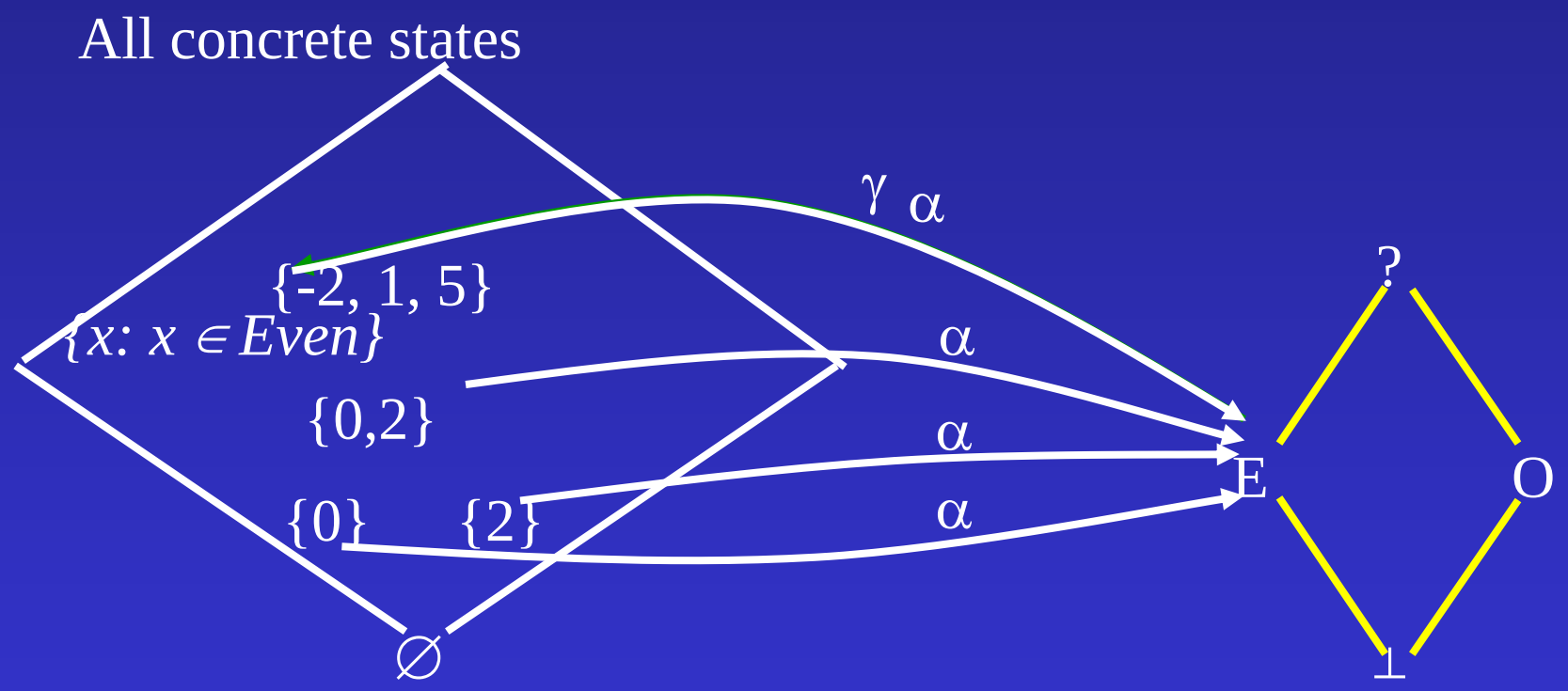
Abstract Interpretation



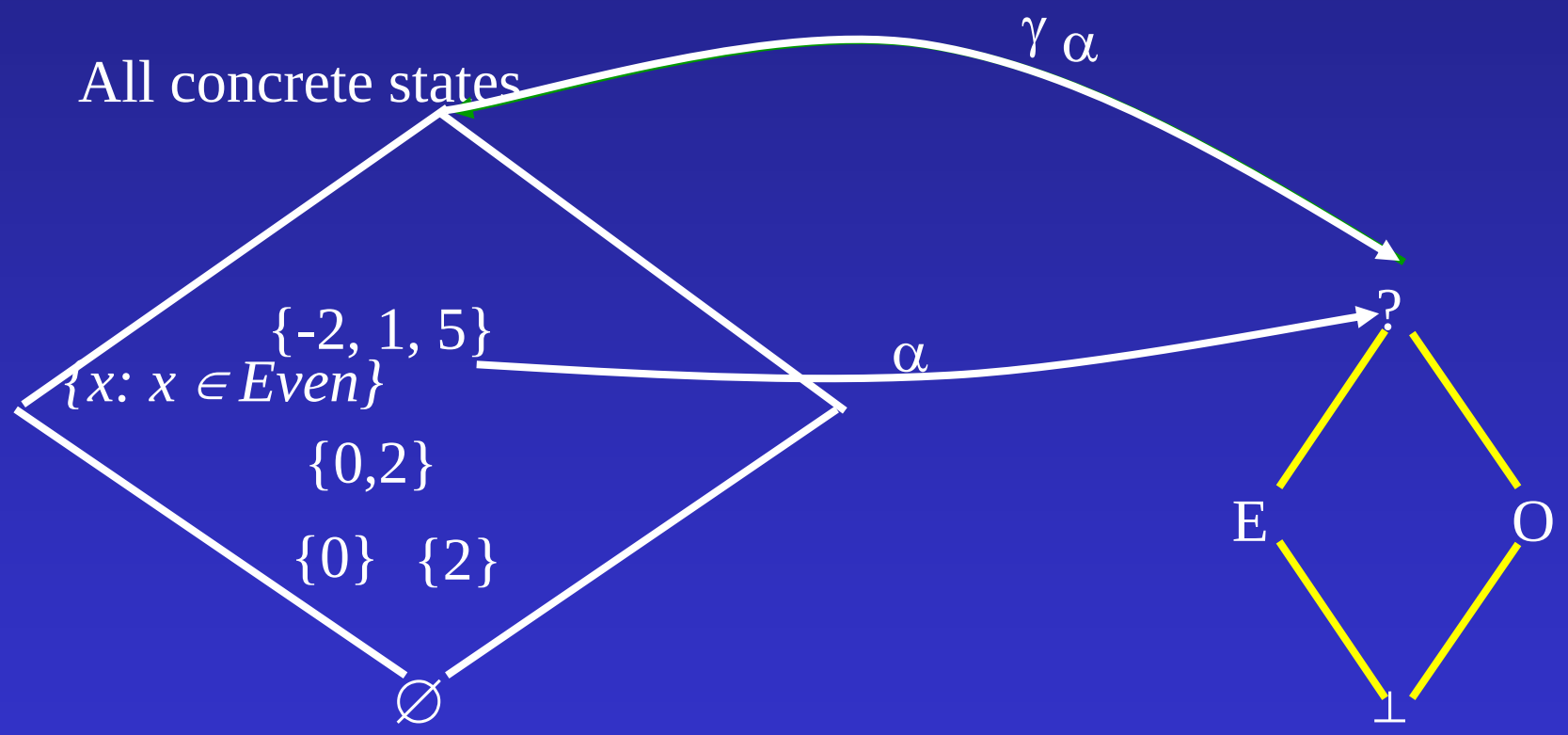
Odd/Even Abstract Interpretation



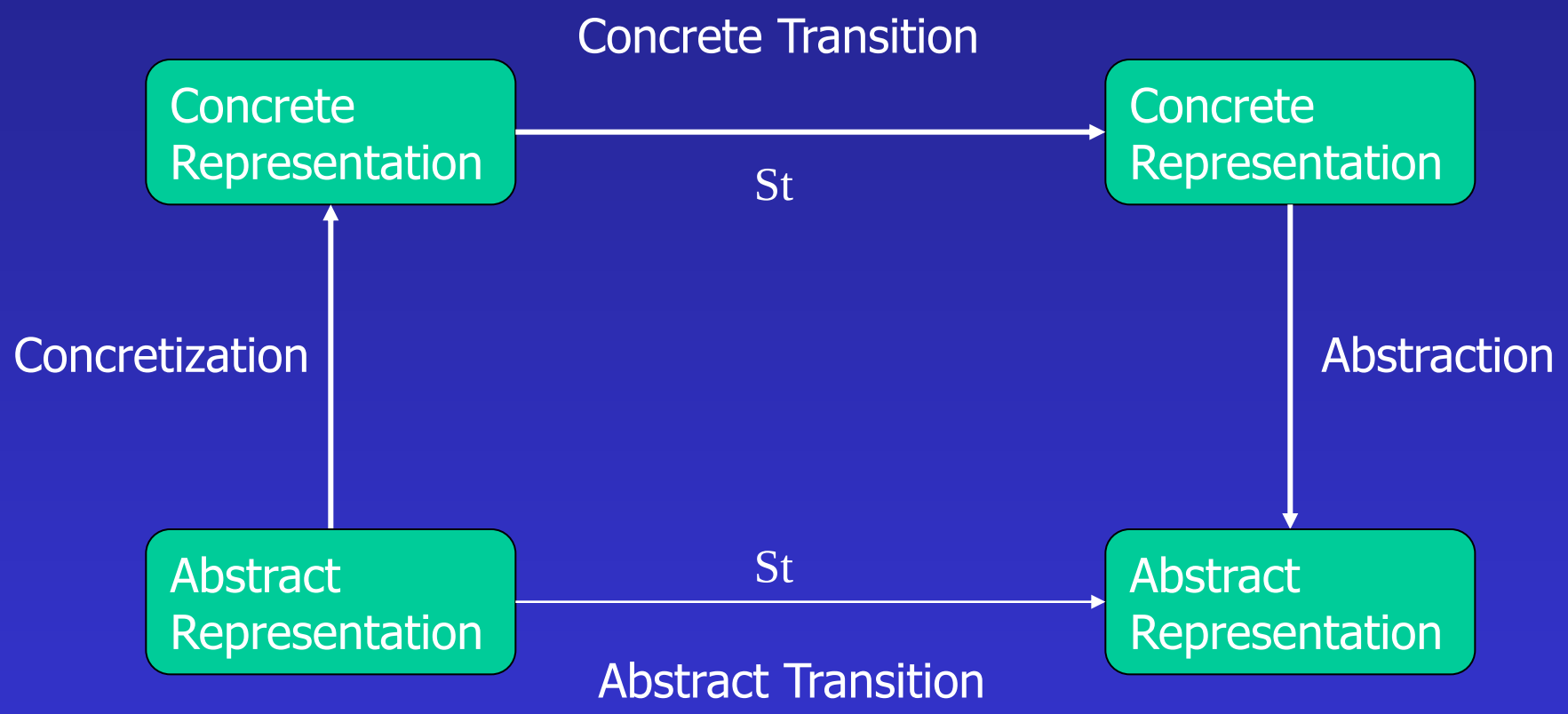
Odd/Even Abstract Interpretation



Odd/Even Abstract Interpretation

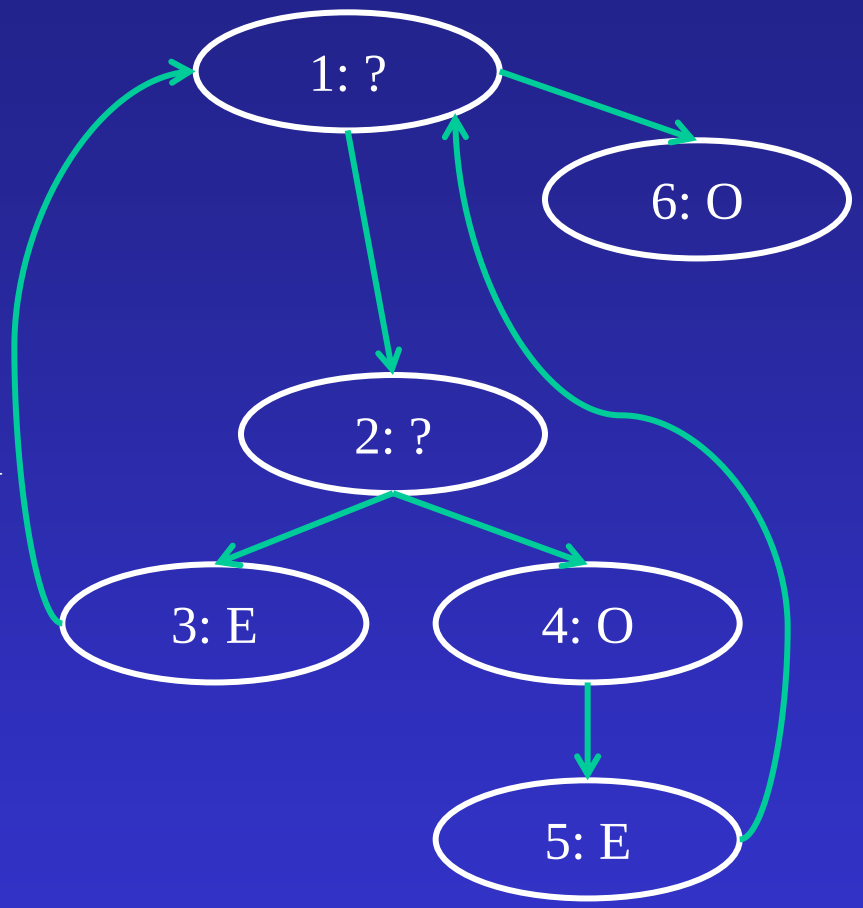


(Best) Abstract Transformer



Odd/Even Abstract Interpretation

```
1: while  (x !=1)   do {
        2: if   (x %2) == 0
            { 3: x := x / 2; }
             else
            { 4 : x := x * 3 + 1;
           5: assert (x %2 ==0); }
6: }
```



Summary Abstract Interpretation

- Conceptual method for building static analyzers
- A lot of techniques:
  - join, meet, widening, narrowing, procedures
- Can be combined with theorem provers

Tentative Schedule

Date	Class
10/3	Introduction
20/3 10-13, Scriber 309	SAT Solvers for propositional logic
24/3	Beyond propositional logic: SMT Solvers
14/4	Symbolic Reasoning with SMT Solvers
21/4	Concolic Testing
28/4	Bounded Model Checking
5/5	SAT based inductive reasoning(Interpolants)
12/5	Introduction to Abstract Interpretation
29/5	Pointers and Shape Analysis
2/6	Applications to Shape Analysis
9/6	Property Directed Abstract Interpretation