Symbolic Evaluation/Execution

Today's Reading Material

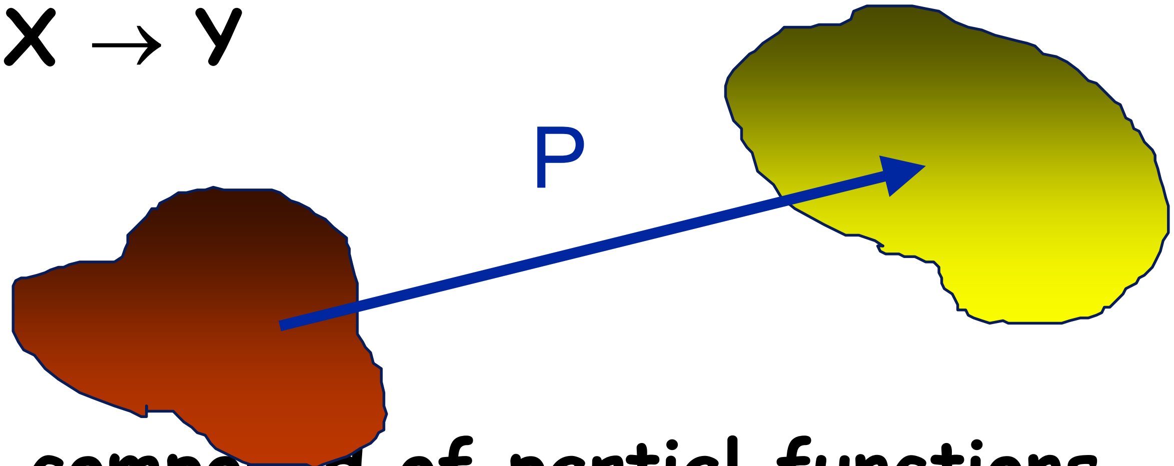
- L. A. Clarke and D. J. Richardson, "Applications of Symbolic Evaluation," Journal of Systems and Software, 5 (1), January 1985, pp.15-35.

Symbolic Evaluation/Execution

- Creates a functional representation of a path of an executable component
- For a path Pi
  - D[Pi] is the domain for path Pi
  - C[Pi] is the computation for path Pi

Functional Representation of an Executable Component

$P : X \rightarrow Y$



P is composed of partial functions corresponding to the executable paths

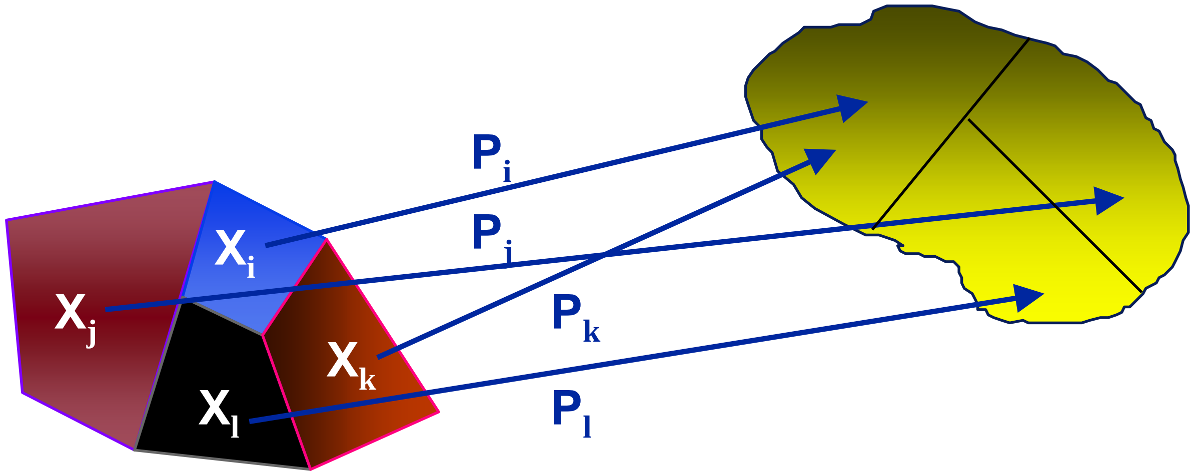
$$P = \{P_1, \ldots, P_r\}$$

$$P_i : X_i \rightarrow Y$$

Functional Representation of an Executable Component

X_i is the domain of path P_i

Denoted $D[P_i]$



$$X = D[P_1] \cup \ldots \cup D[P_r] = D[P]$$

$$D[P_i] \cap D[P_j] = \emptyset,\ i \neq j$$

Representing Computation

- **Symbolic names** represent the input values
- the **path value PV** of a variable for a path describes the value of that variable in terms of those symbolic names
- the **computation** of the path **C[P]** is described by the path values of the outputs for the path

Representing Conditionals

- an interpreted branch condition or interpreted predicate is represented as an inequality or equality condition
- the path condition PC describes the domain of the path and is the conjunction of the interpreted branch conditions
- the domain of the path D[P] is the set of input values that satisfy the PC for the path

Example program

```
   procedure Contrived is
   X, Y, Z : integer;
1    read X, Y;
2    if X ≥ 3 then
3        Z := X+Y;
     else
4        Z := 0;
     endif;
5    if Y > 0 then
6        Y := Y + 5;
     endif;
7    if X - Y < 0 then
8        write Z;
   else
9        write Y;
   endif;
   end Contrived;
```

Stmt	PV	PC
1	$X \leftarrow x$ $Y \leftarrow y$	true
2,3	$Z \leftarrow x+y$	$\text{true} \wedge x \geq 3 = x \geq 3$
5,6	$Y \leftarrow y+5$	$x \geq 3 \wedge y > 0$
7,9		$x \geq 3 \wedge y > 0 \wedge x-(y+5) \geq 0$ $= x \geq 3 \wedge y > 0 \wedge (x-y) \geq 5$

Presenting the results

```
   procedure Contrived is
   X, Y, Z  : integer;
1          read X, Y;
2          if X ≥ 3 then
3       Z := X+Y;
           else
4       Z := 0;
           endif;
5          if Y > 0 then
6       Y := Y + 5;
           endif;
7          if X - Y < 0 then
8       write Z;
   else
9       write Y;
   endif
   end Contrived
```

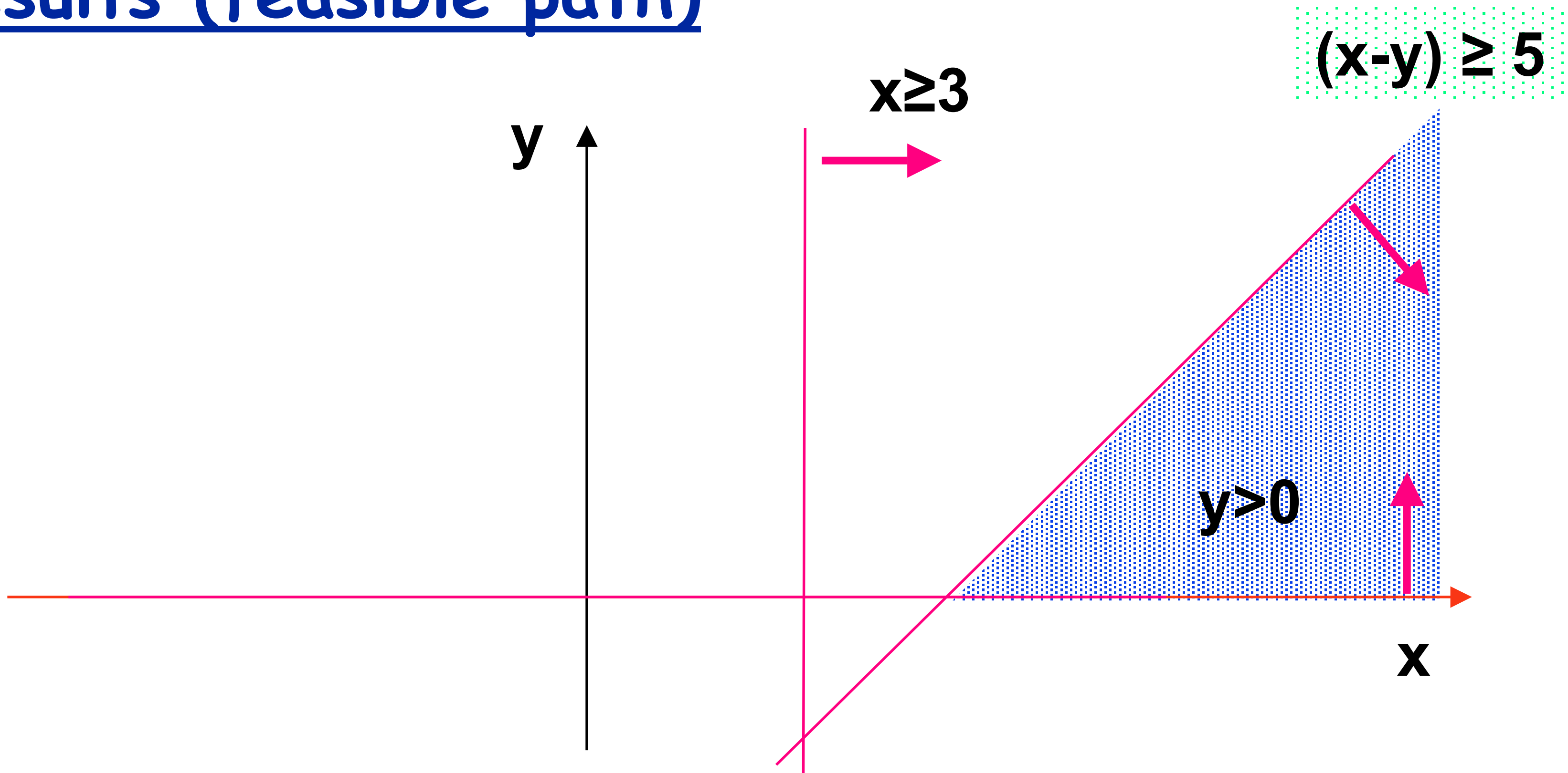
Statements	PV	PC
1	$X \leftarrow x$ $Y \leftarrow y$	true
2,3	$Z \leftarrow x+y$	$true \wedge x \geq 3 = x \geq 3$
5,6	$Y \leftarrow y+5$	$x \geq 3 \wedge y > 0$
7,9		$x \geq 3 \wedge y > 0 \wedge x-(y+5) \geq 0 =$ $x \geq 3 \wedge y > 0 \wedge (x-y) \geq 5$

P = 1, 2, 3, 5, 6, 7, 9

$D[P] = \{ (x,y) \mid x \geq 3 \wedge y > 0 \wedge x-y \geq 5 \}$

$C[P] = PV.Y = y + 5$

Results (feasible path)



$P = 1, 2, 3, 5, 6, 7, 9$

$D[P] = \{ (x,y) | x \geq 3 \wedge y > 0 \wedge x - y \geq 5 \}$

$C[P] = PV.Y = y + 5$

Evaluating another path

```
  procedure Contrived is
  X, Y, Z : integer;
1   read X, Y;
2   if X ≥ 3 then
3        Z := X+Y;
    else
4        Z := 0;
    endif;
5   if Y > 0 then
6        Y := Y + 5;
    endif;
7   if X - Y < 0 then
8        write Z;
  else
9        write Y;
  endif;
  end Contrived;
```

Stmts	PV	PC
1	$X \leftarrow x$ $Y \leftarrow y$	true
2,3	$Z \leftarrow x+y$	$true \wedge x \geq 3 = x \geq 3$
5,7		$x \geq 3 \wedge y \leq 0$
7,8		$x \geq 3 \wedge y \leq 0 \wedge x-y < 0$

```
        procedure EXAMPLE is
        X, Y, Z  : integer;
1                read X, Y;
2                if X ≥ 3 then
3            Z := X+Y;
                 else
4            Z := 0;
                 endif;
5                if Y > 0 then
6            Y := Y + 5;
                 endif;
7                if X - Y < 0 then
8            write Z;
        else
9            write Y;
        endif
        end EXAMPLE
```

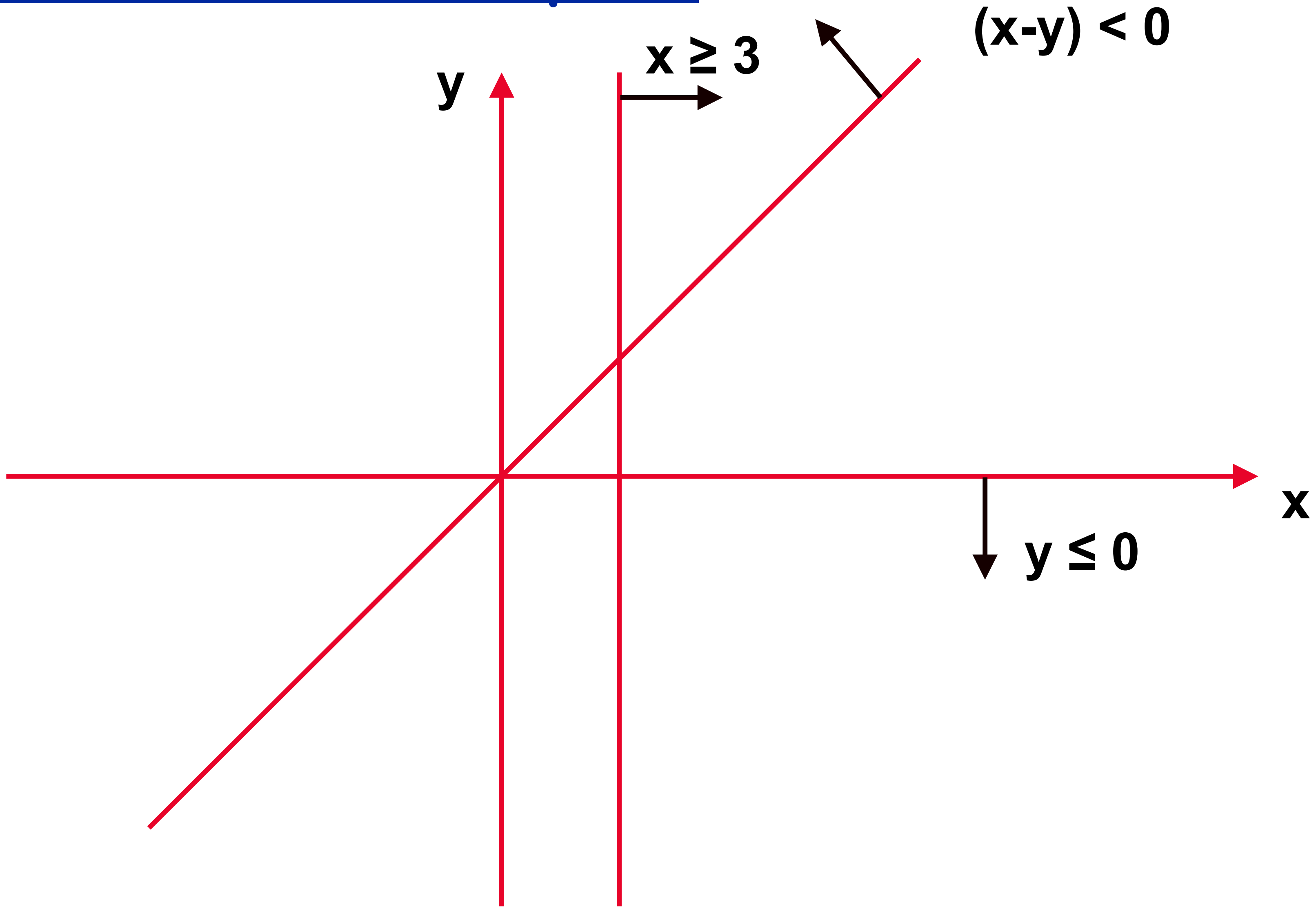
Stmts	PV	PC
1	$X \leftarrow x$ $Y \leftarrow y$	true
2,3	$Z \leftarrow x+y$	$\text{true} \wedge x \geq 3 = x \geq 3$
5,7		$x \geq 3 \wedge y \leq 0$
7,8		$x \geq 3 \wedge y \leq 0 \wedge x-y < 0$

$P = 1, 2, 3, 5, 7, 8$

$D[P] = \{ (x,y) \mid x \geq 3 \wedge y \leq 0 \wedge x-y<0 \}$

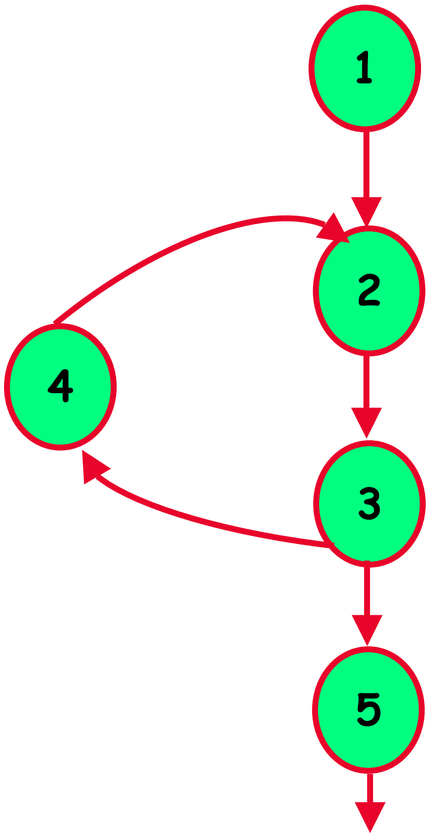
infeasible path!

Results (infeasible path)



what about loops?

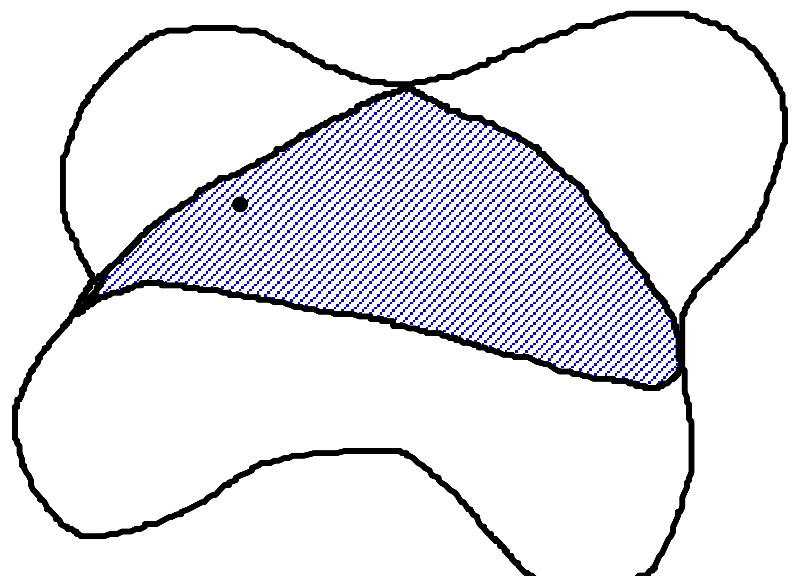
- Symbolic evaluation requires a full path description



- Example Paths
  - P= 1, 2, 3, 5
  - P= 1, 2, 3, 4, 2, 3, 5
  - P= 1, 2, 3, 4, 2, 3, 4, 2, 3, 5
  - Etc.

Symbolic Testing

- Path Computation provides [concise] functional representation of behavior for entire Path Domain
- Examination of Path Domain and Computation often useful for detecting program errors
- Particularly beneficial for scientific applications or applications w/o oracles



Simple Symbolic Evaluation

- Provides symbolic representations given path Pi
  - path condition PC =
  - path domain D[Pi] ={(x1, x1, ... ,x1)|pc true }
  - path values PV.X1=
  - path computation C[Pi] =

P = 1, 2, 3, 5, 6, 7, 9

D[P] = { (x,y) | $x \geq 3 \wedge y > 0 \wedge x - y \geq 5$}

C[P] = PV.Y = y +5

Additional Features:

- Simplification
- Path Condition Consistency
- Fault Detection
- Path Selection
- Test Data Generation

Simplification

- Reduces path condition to a canonical form
- Simplifier often determines consistency

  PC = (x >= 5) and (x < 0)

- May want to display path computation in simplified and unsimplified form

  PV.X = x + (x +1) + (x + 2) + (x + 3)
  = 4 * x + 6

Path Condition Consistency

- strategy = solve a system of constraints
  - theorem prover
    - *consistency*
  - algebraic, e.g., linear programming
    - *consistency and find solutions*
    - solution is an example of automatically generated test data

... but, in general we cannot solve an arbitrary system of constraints!

Fault Detection

- Implicit fault conditions
    - E.g. Subscript value out of bounds
    - E.g. Division by zero e.g., Q:=N/D
  - Create assertion to represent the fault and conjoin with the pc
    - Division by zero assert(divisor $\neq$ 0)
    - Determine consistency
      PC_P and (PV.divisor = 0)
    - if consistent then error possible
  - Must check the assertion at the point in the path where the construct occurs

Checking user-defined assertions

- example
  - Assert (A > B)
  - PC and (PV.A) ≤ PV.B)
  - if consistent then assertion not valid

Comparing Fault Detection Approaches

- assertions can be inserted as executable instructions and checked during execution
  - dependent on test data selected (*dynamic testing*)
- use symbolic evaluation to evaluate consistency
  - dependent on path, but not on the test data
  - looks for violating data in the path domain

Additional Features:

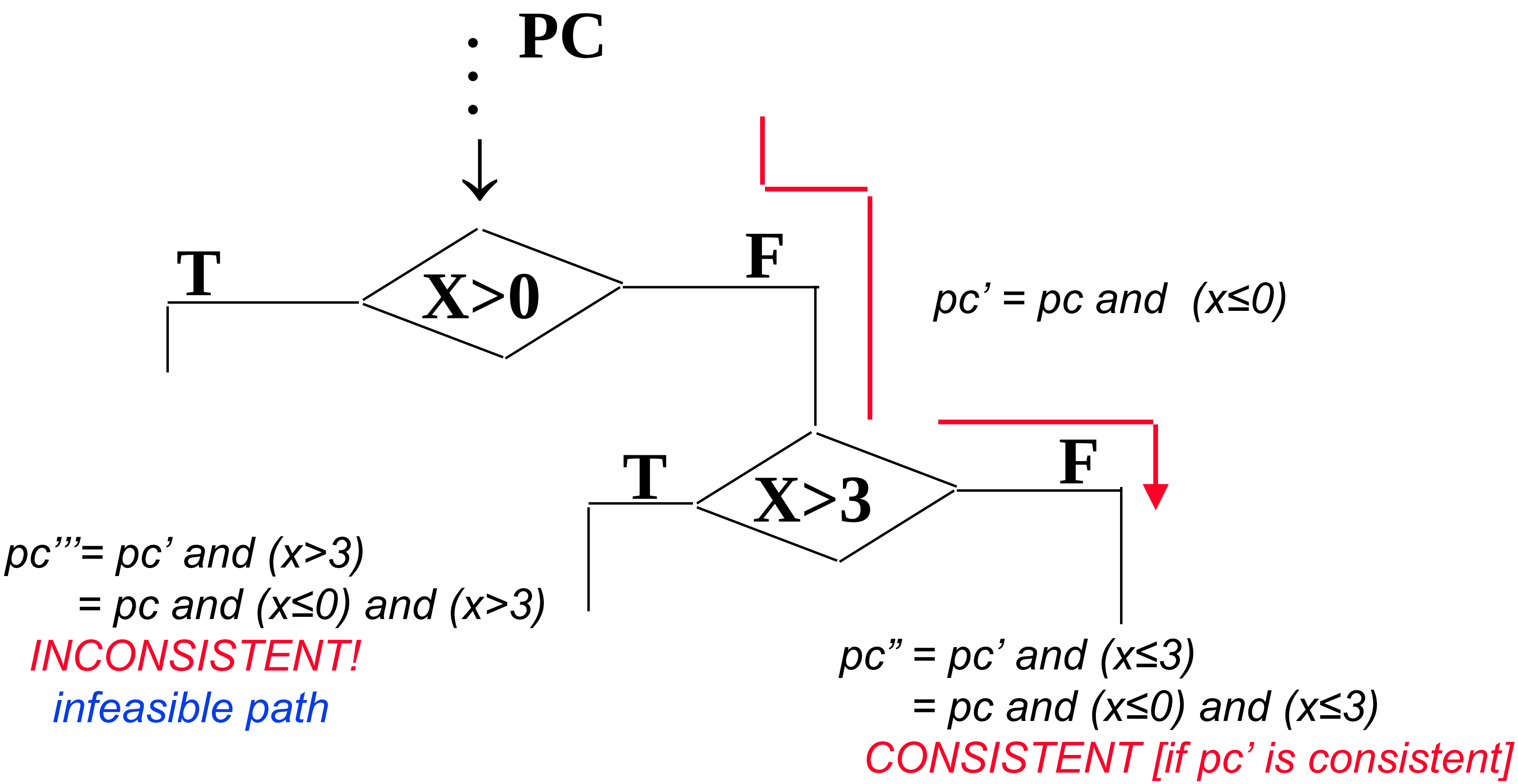
- Simplification
- Path Condition Consistency
- Fault Detection
- Path Selection
- Test Data Generation

Path Selection

- User selected
- Automated selection to satisfy some criteria
  - e.g., exercise all statements at least once
- Because of infeasible paths, best if path selection done incrementally

Incremental Path Selection

- PC and PV maintained for partial path
- Inconsistent partial path can often be salvaged



Path Selection (continued)

Can be used in conjunction with other static analysis techniques to determine path feasibility

- Testing criteria generates a path that needs to be tested
- Symbolic evaluation determines if the path is feasible
    - Can eliminate some paths from consideration

Additional Features:

- Simplification
- Path Condition Consistency
- Fault Detection
- Path Selection
- Test Data Generation

Test Data Generation

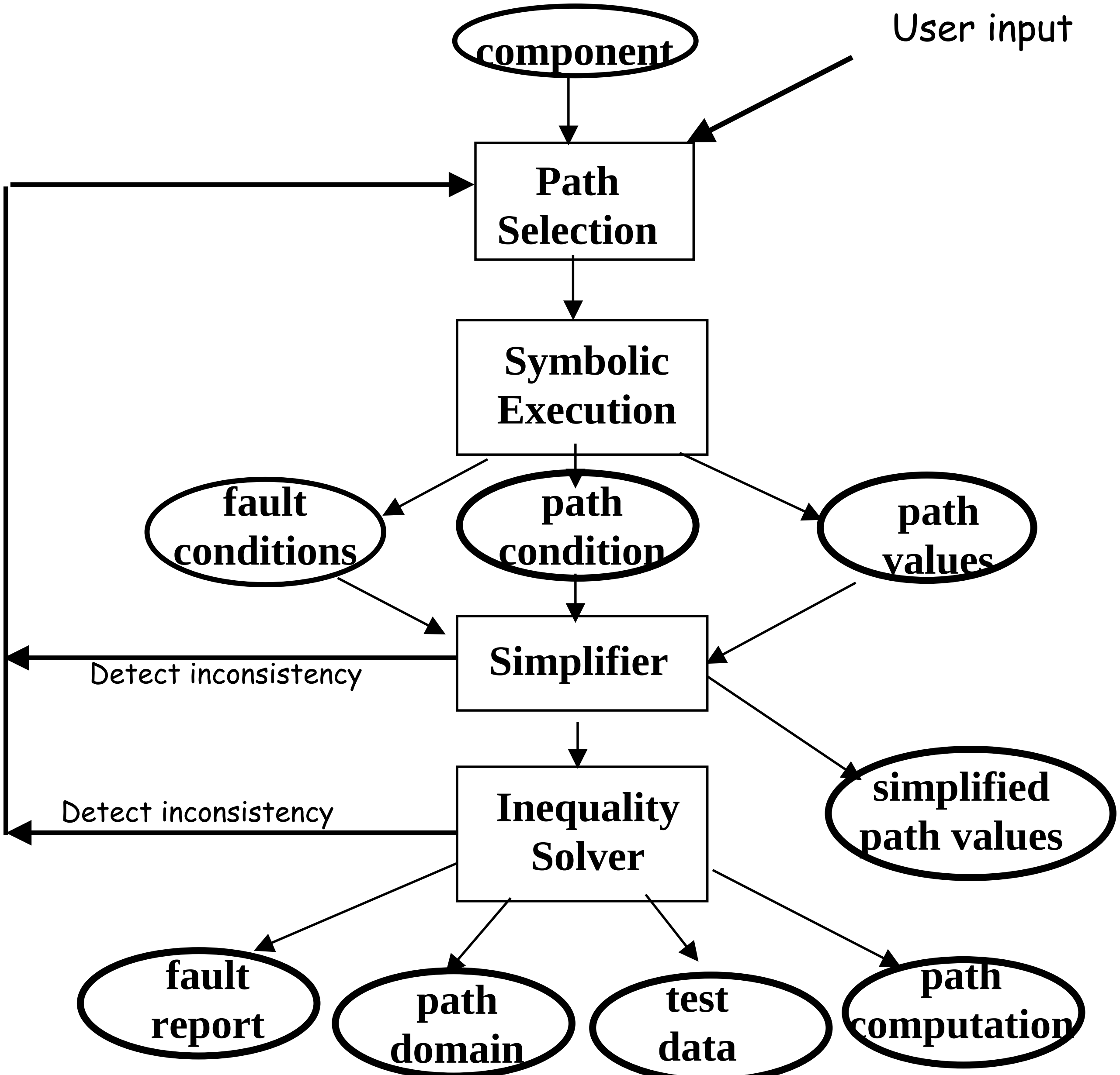
- Simple test date selection: Select test data that satisfies the path condition pc
- Error based test date selection
  - Try to select test cases that will help reveal faults
  - Use information about the path domain and path values to select test data
    - e.g., PV.X = a * (b + 2);

      a = 1 combined with min and max values of b
      b = -1 combined with min and max values for a

Enhanced Symbolic Evaluation Capabilities

- Creates symbolic representations of the Path Domains and Computations
  - "Symbolic Testing"
- Determine if paths are feasible
- Automatic fault detection
  - system defined
  - user assertions
- Automatic path selection
- Automatic Test Data Generation

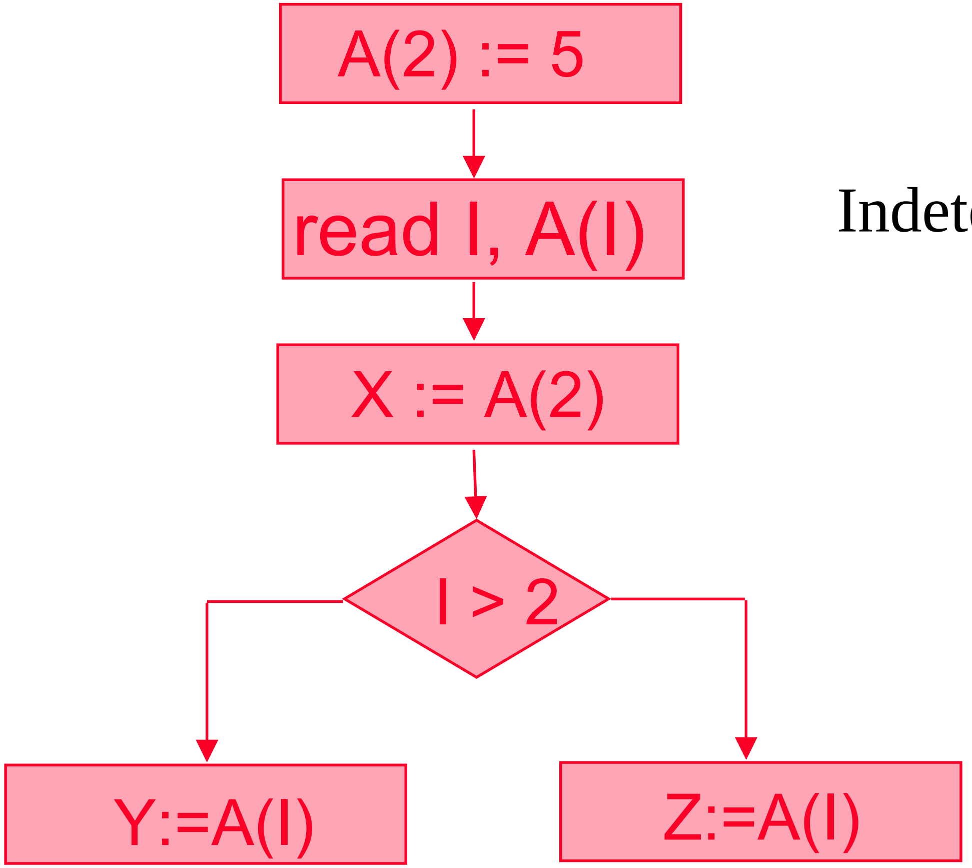
An Enhanced Symbolic Evaluation System



Problems

- Information explosion
- Impracticality of all paths
- Path condition consistency
- Aliasing
  - elements of a compound type e.g., arrays and records
  - pointers

Alias Problem



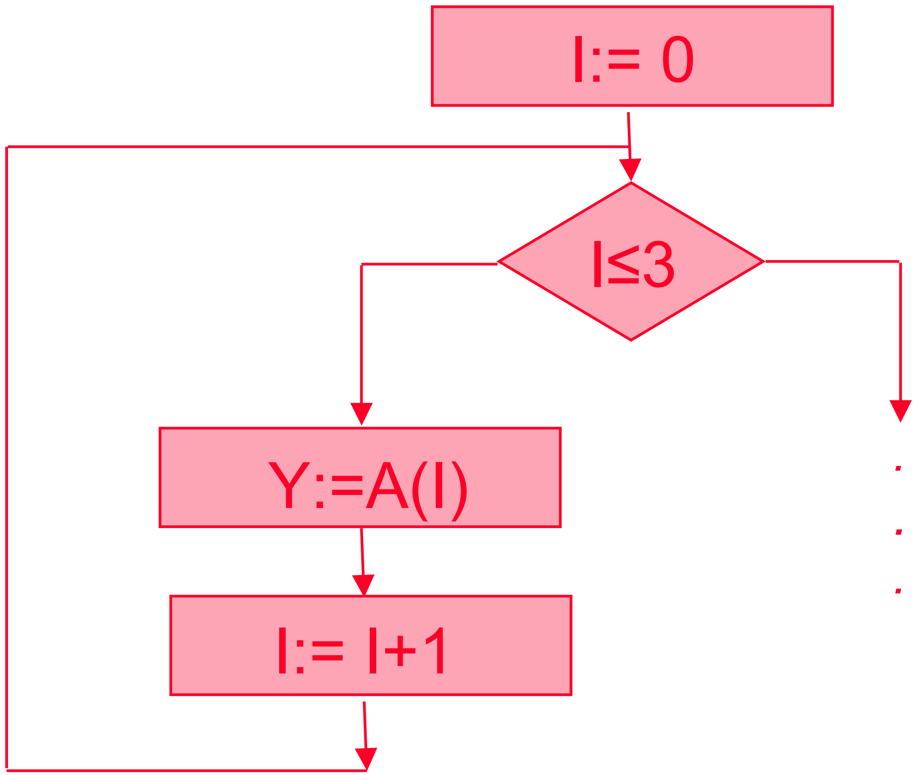
Indeterminate subscript

constraints on subscript value due to path condition

Escalating problem

- Read I
- X := A[I]		PV.X = unknown
- Y := X + Z		PV.Y = unknown + PV.Z
  = unknown

Can often determine array element



. *subscript value*
. *is constant*
.

Symbolic Evaluation Approaches

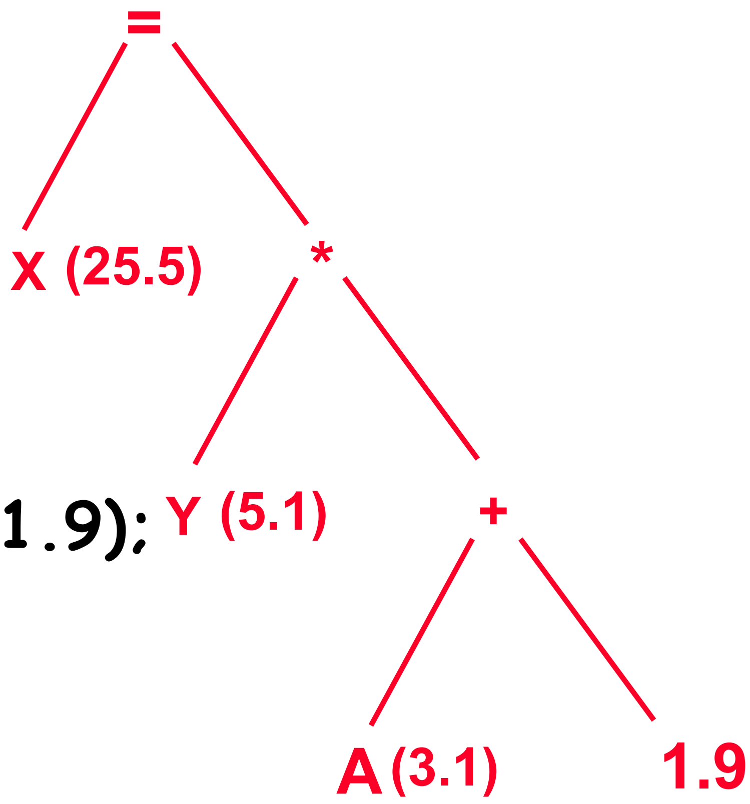
- symbolic evaluation
  - With some enhancements
  - Data independent
  - Path dependent
- dynamic symbolic evaluation
  - Data dependent--> path dependent
- global symbolic evaluation
  - Data independent
  - Path independent

Dynamic Symbolic Execution

- Data dependent
- Provided information
  - Actual value:

    X := 25.5
  - Symbolic expression:

    X := Y * (A + 1.9);
  - Derived expression:



Dynamic Analysis combined with Symbolic Execution

- Actual output values
- Symbolic representations for each path executed
  - path domain
  - path computation
- Fault detection
  - data dependent
  - path dependent (if accuracy is available)

Dynamic Symbolic Execution

- Advantages
  - No path condition consistency determination
  - No path selection problem
  - No aliasing problem (e.g., array subscripts)
- Disadvantages
  - Test data selection (path selection) left to user
  - Fault detection is often data dependent
- Applications
  - Debugging
  - Symbolic representations used to support path and data selection

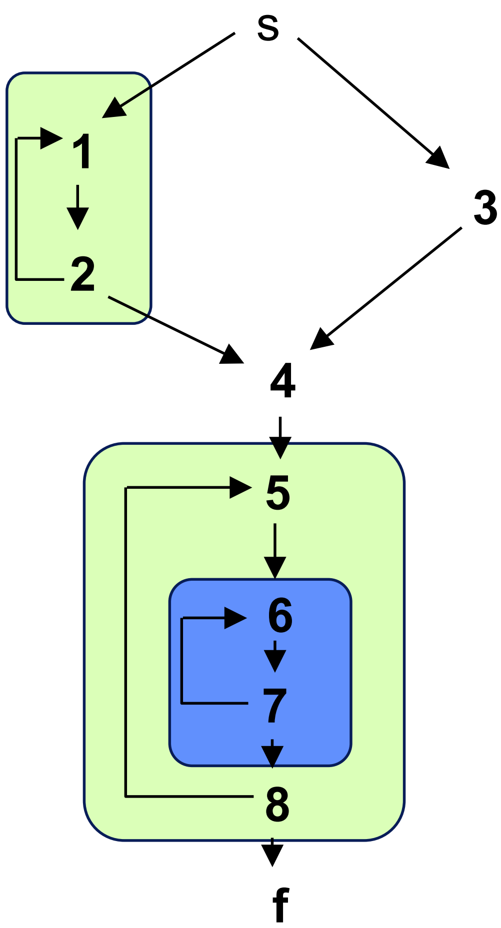
Symbolic Evaluation Approaches

- simple symbolic evaluation
- dynamic symbolic evaluation
- global symbolic evaluation
  - Data and path independent
  - Loop analysis technique classifies paths that differ only by loop iterations
  - Provides global symbolic representation for each class of paths

Global Symbolic Evaluation

- Loop Analysis
  - creates recurrence relations for variables and loop exit condition
  - solution is a closed form expression representing the loop
  - then, loop expression evaluated as a single node

Global Symbolic Evaluation



2 classes of paths:

P_1:(s,(1,2),4,(5,(6,7),8),f)

P_2: (s,3,4,(5,(6,7),8),f)

global analysis

case

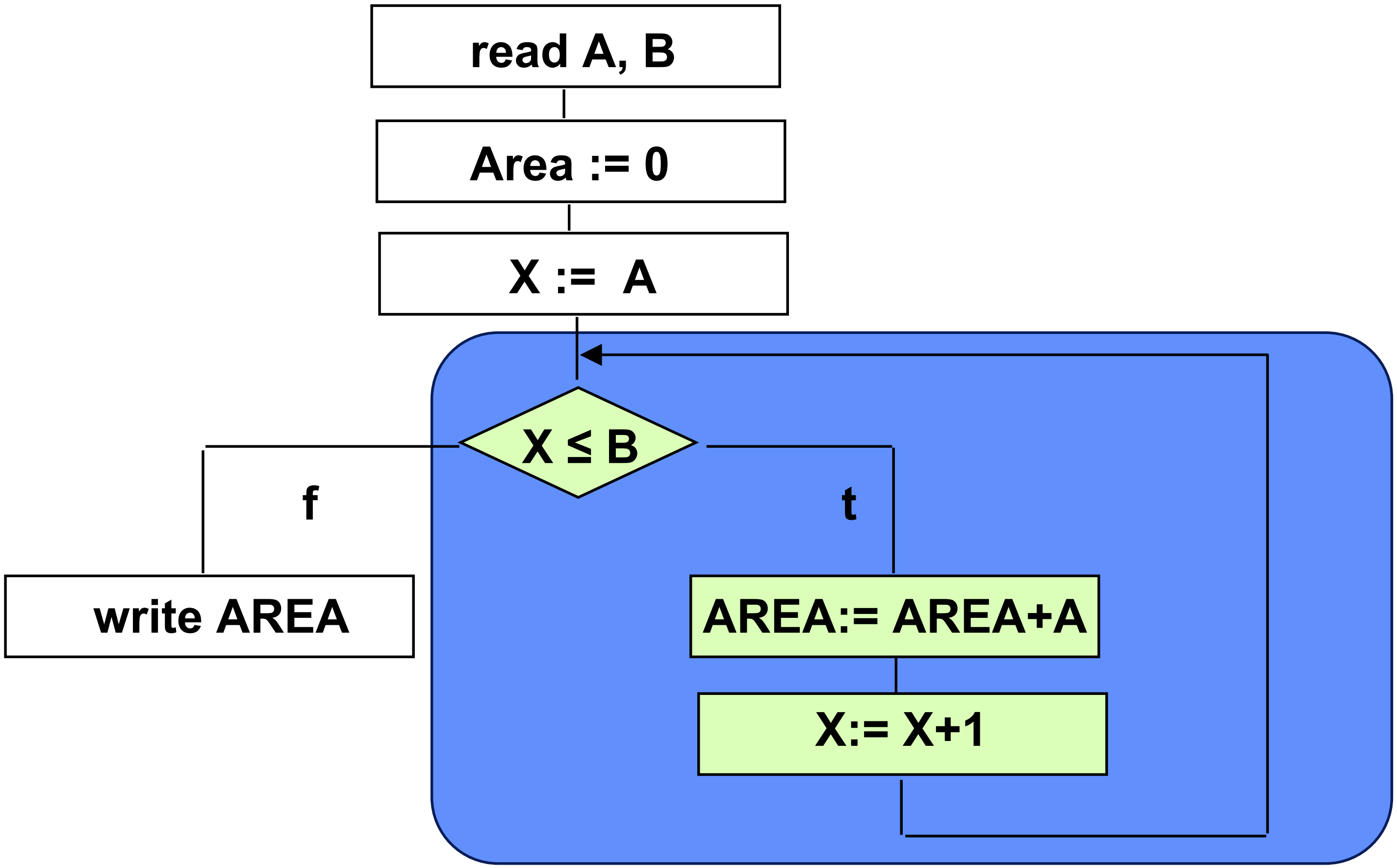
D[P_1]: C[P_1]

D[P_2]: C[P_2]

Endcase

- *analyze the loops first*
- *consider all partial paths up to a node*

Loop analysis example



Loop Analysis Example

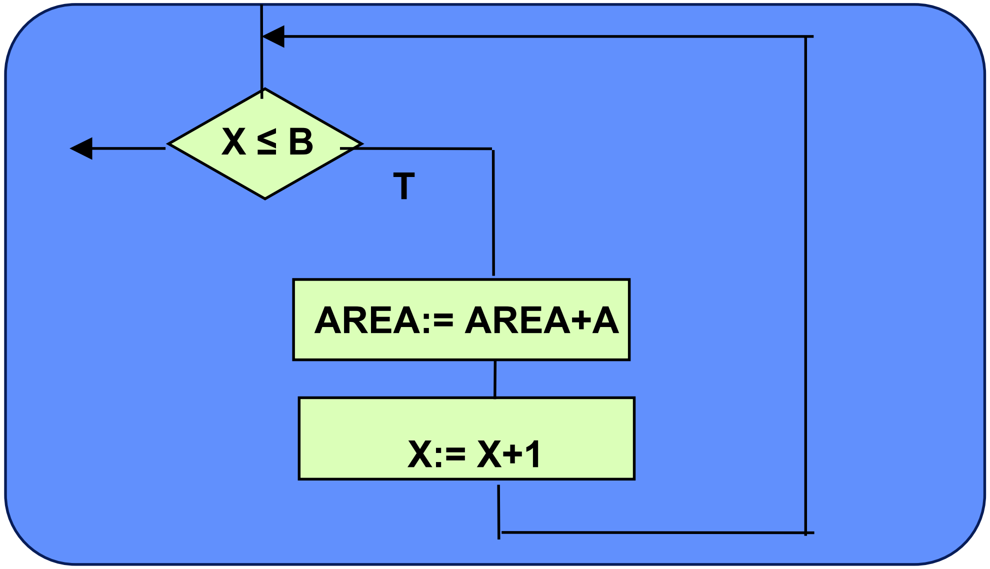
- **Recurrence Relations**

  $AREA_k = AREA_{k-1} + A_0$

  $X_k = X_{k-1} + 1$

- **Loop Exit Condition**

  $lec(k) = (X_k > B_0)$



Loop Analysis Example (continued)

- solved recurrence relations

  $$AREA(k) = AREA_0 + \sum_{i = X_0}^{X_0 + k - 1} A_0$$

  $$X(k) = X_0 + k$$

- solved loop exit condition

  $$lec(k) = (X_0 + k > B_0)$$

- loop expression

  $$k_e = \min \{k \mid X_0 + k > B_0 \text{ and } k \geq 0\}$$

  $$AREA := AREA_0 + \sum_{i = X_0}^{X_0 + k_e - 1} A_0$$

  $$X := X_0 + k_e$$

- loop expression

  $k_e = \min \{k \mid X_0 + k > B_0 \text{ and } k \geq 0\}$

  $AREA := AREA_0 + \sum_{i=X_0}^{X_0+k_e-1} A_0$

  $X := X_0 + k_e$

- global representation for input (a,b)

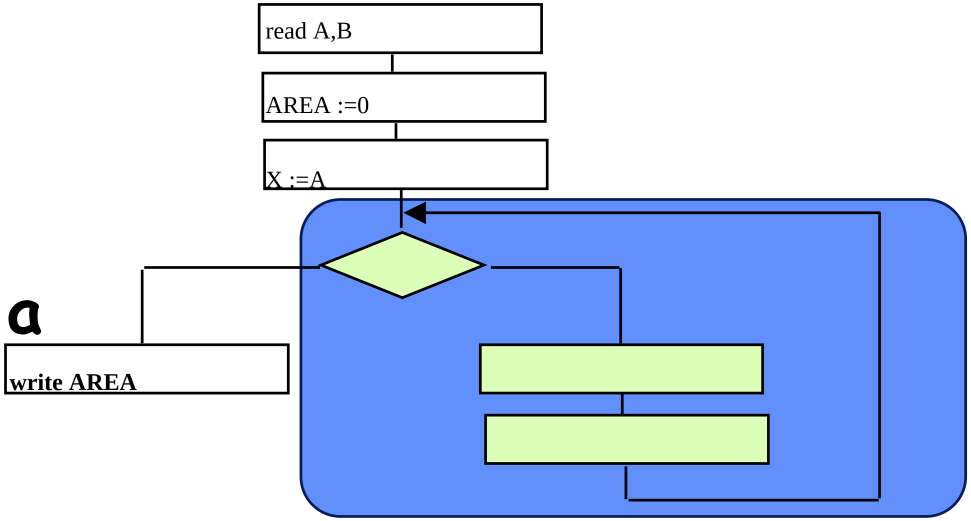
  $X_0 = a,\ A_0 = a,\ B_0 = b,\ AREA_0 = 0$

  $a + k_e > b \implies k_e > b - a$

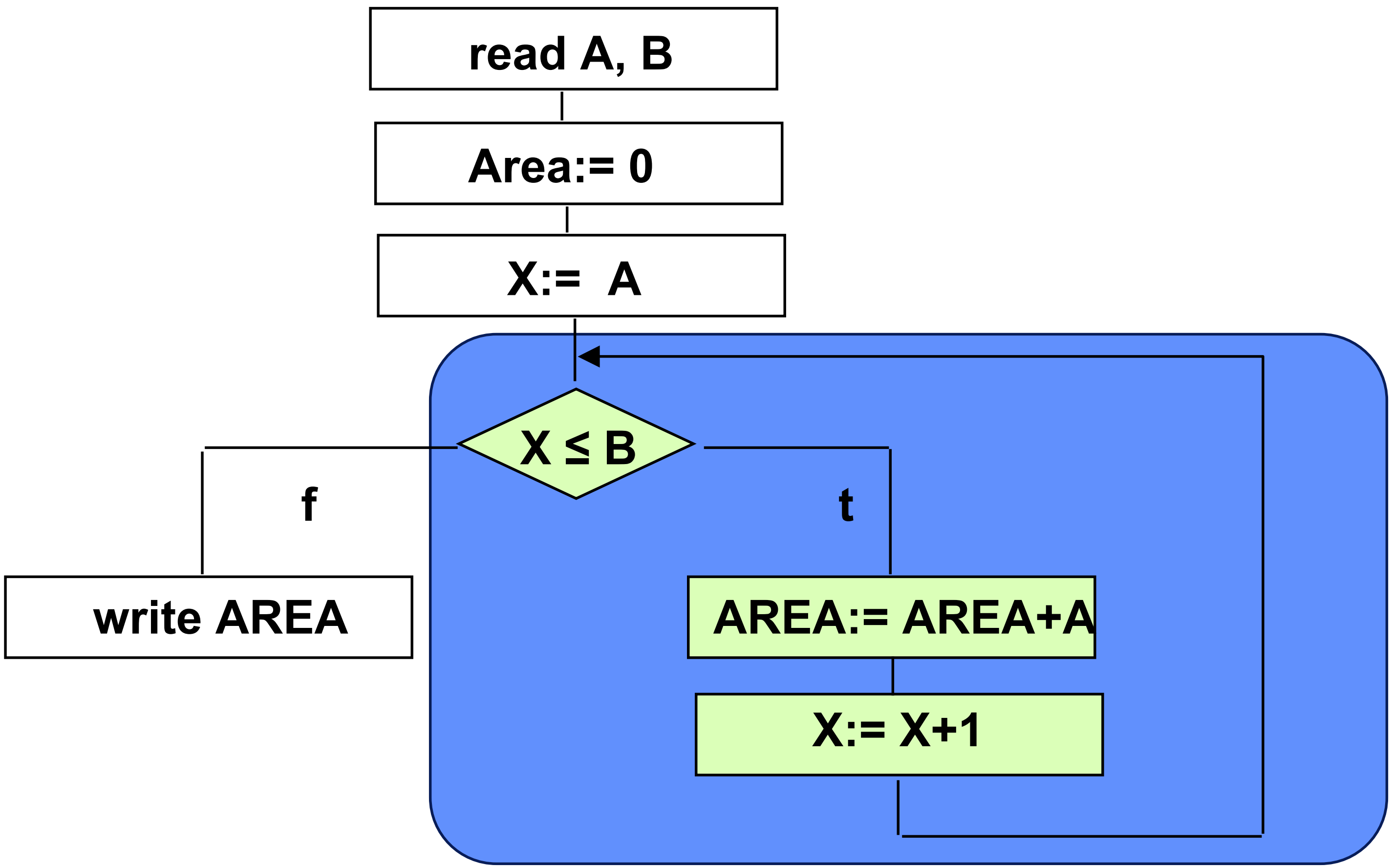
  $K_e = b - a + 1$

  $X = a + (b-a+1) = b+1$

  $AREA = \sum_{i=a}^{b} a = (b-a+1)\, a$

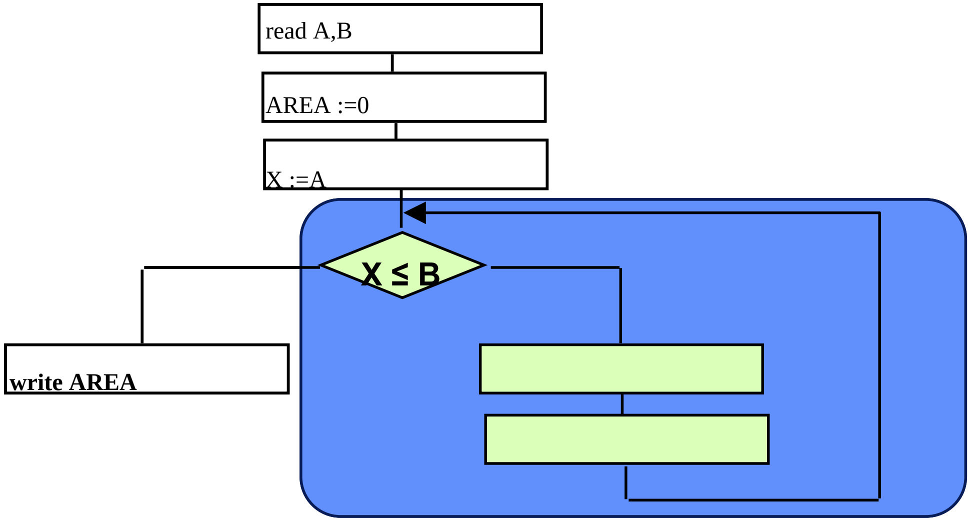


Loop analysis example



Find path computation and path domain for all classes of paths

- P1 = (1, 2, 3, 4, 7)
- D[P1] = a > b
- C[P1] = (AREA=0) and (X=a)



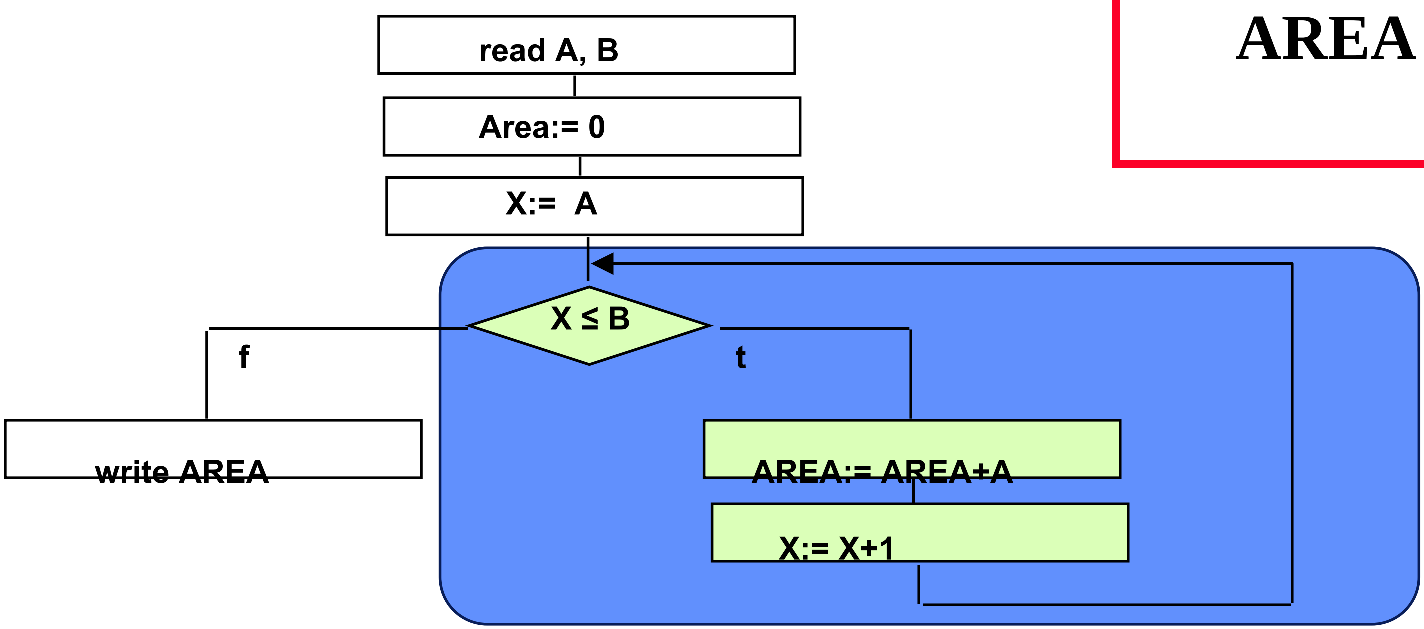
Find path computation and path domain for all classes of paths

- P2 = (1, 2, 3, 4, (5, 6), 7)
  - D[P2] = (b>a)
  - C[P2] = (AREA = (b-a+1) a)

  k_e =b - a +1

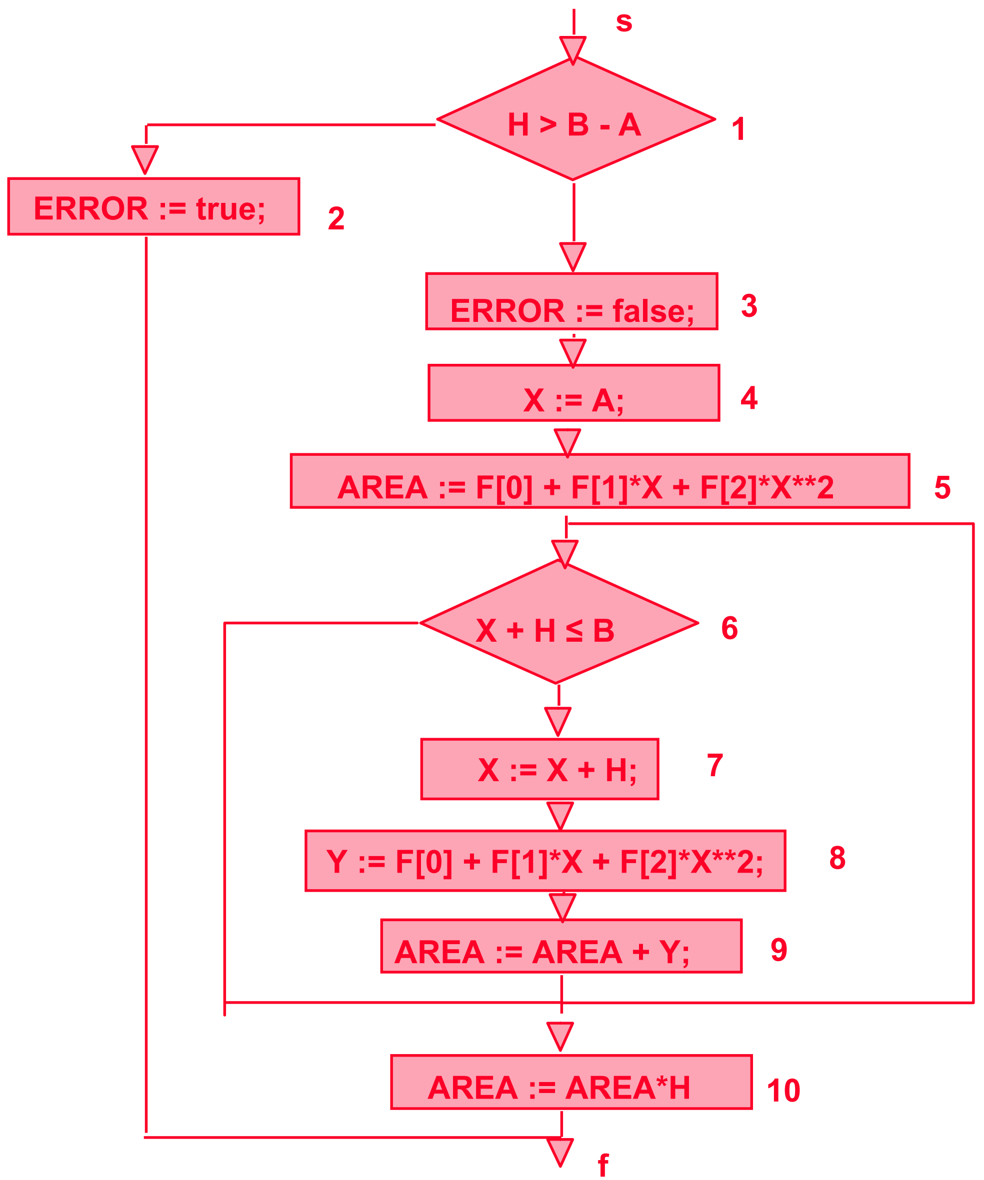
  X : = b + 1

$X_0 = a$
$B_0 = b$
$A_0 = a$
$K_e = b - a + 1$
$X = b+1$
$AREA = (b-a+1)\ a$



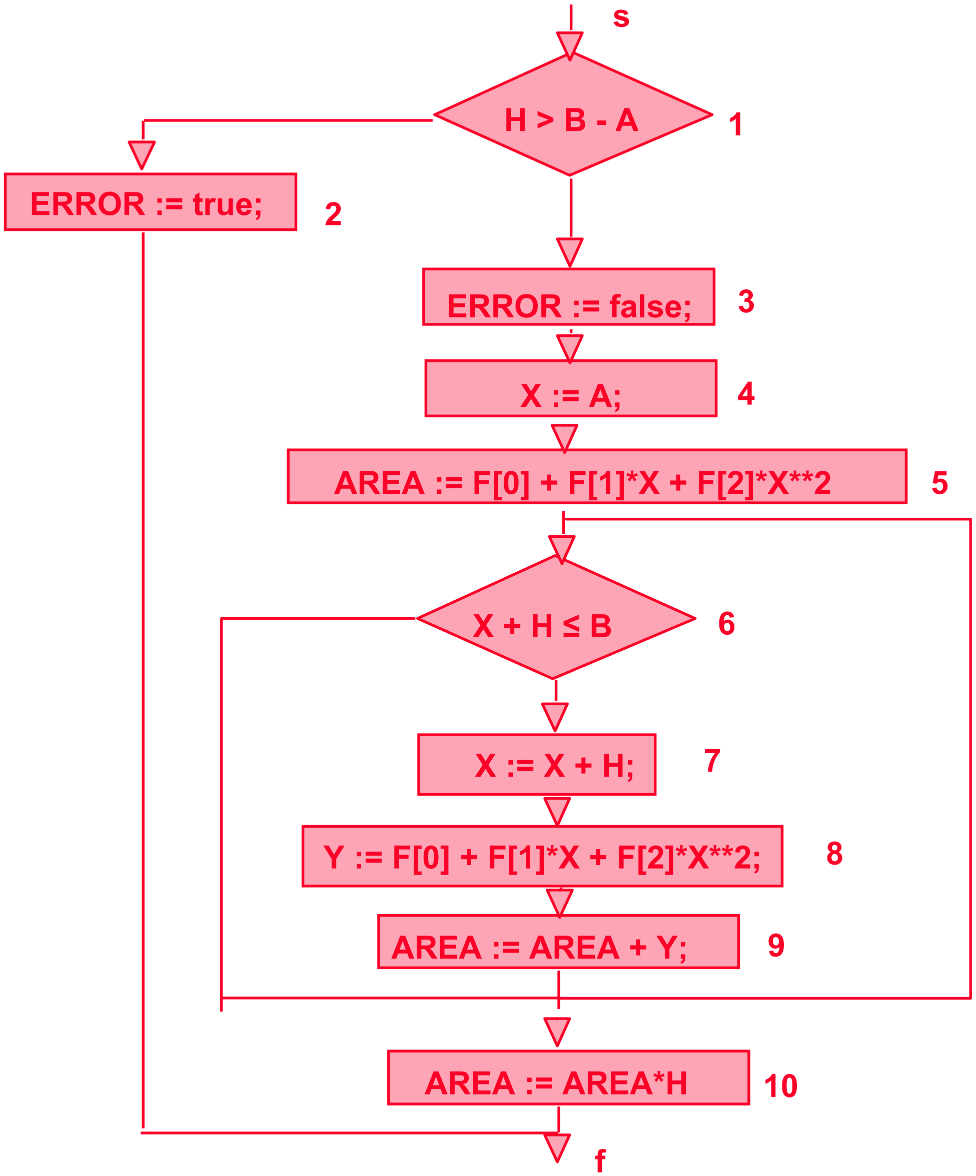
Example

```
procedure RECTANGLE (A,B: in real; H: in real range -1.0 ... 1.0;
F: in array [0..2] of real; AREA: out real; ERROR: out boolean) is
-- RECTANGLE approximates the area under the quadratic equation
-- F[0] + F[1]*X + F[2]*X**2 From X=A to X=B in increments of H.
            X,Y: real;
s    begin
            • --check for valid input
1           if H > B - A then
2                       ERROR := true;
            • else
3                       ERROR := false;
4                       X := A;
5                       AREA := F[0] + F[1]*X + F[2]*X*2;
6                       while X + H ≤ B loop
7                           X := X + H;
8                           Y := F[0] + F[1]*X + F[2]*X**2;
9                           AREA := AREA + Y;
                        end loop;
10          AREA := AREA*H;
            endif;
     end RECTANGLE
```


s
H > B - A
1
ERROR := true;
2
ERROR := false;
3
X := A;
4
AREA := F[0] + F[1]*X + F[2]*X**2
5
X + H ≤ B
6
X := X + H;
7
Y := F[0] + F[1]*X + F[2]*X**2;
8
AREA := AREA + Y;
9
AREA := AREA*H
10
f

Symbolic Representation of Rectangle

P_1	(s,1,2,f)
$D[P_1]$	(a - b + h > 0.0)
$C[P_1]$	AREA = ? ERROR = true
P_2	(s,1,3,4,5,6,10,f)
$D[P_2]$	(a - b + h <= 0.0) and (a - b + h > 0.0) = = false *** infeasible path ***
P_3	(s,1,3,4,5,(6,7,8,9),10,11,f)
$D[P_3]$	(a-b+h <= 0.0)
$C[P_3]$	AREA = a*f[1]*j+2.0*a*f[2]*h+f[0]*h +sum < i :=1 ... int (-a/h+b/h) | (a*f[1]*h+a**2*f[2]*h +2.0*a*f[2]*h**2*i+f[0]*h +f[1]*h**2*i+f[2]*h**3*i**2) > ERROR = false



Global Symbolic Evaluation

- Advantages
  - global representation of routine
  - no path selection problem
- Disadvantages
  - has all problems of
    - Symbolic Execution PLUS
      - inability to solve recurrence relations
        - *(interdependencies, conditionals)*
- Applications
  - has all applications of
    - Symbolic Execution plus
      - Verification
      - Program Optimization

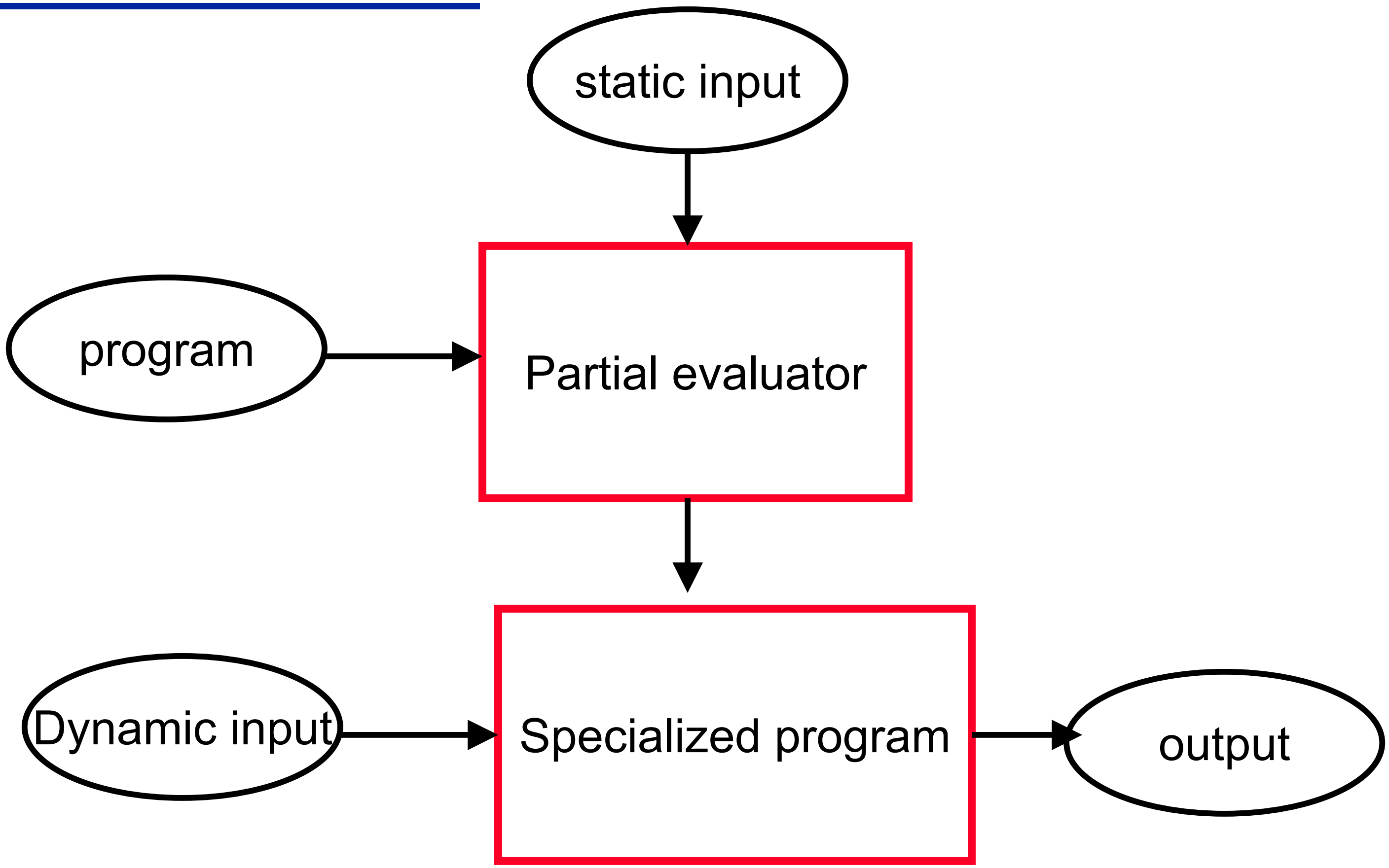
Why hasn't symbolic evaluation become widely used?

- expensive to create representations
- expensive to reason about expressions
- imprecision of results
  - current computing power and better user interface capabilities may make it worth reconsidering

Partial Evaluation

- Similar to (Dynamic) Symbolic Evaluation
- Provide some of the input values
  - If input is x and y, provide a value for x
- Create a representation that incorporates those values and that is equivalent to the original representation if it were given the same values as the preset values
  - $P(x, y) = P'(x', y)$

Partial Evaluator



Why is partial evaluation useful?

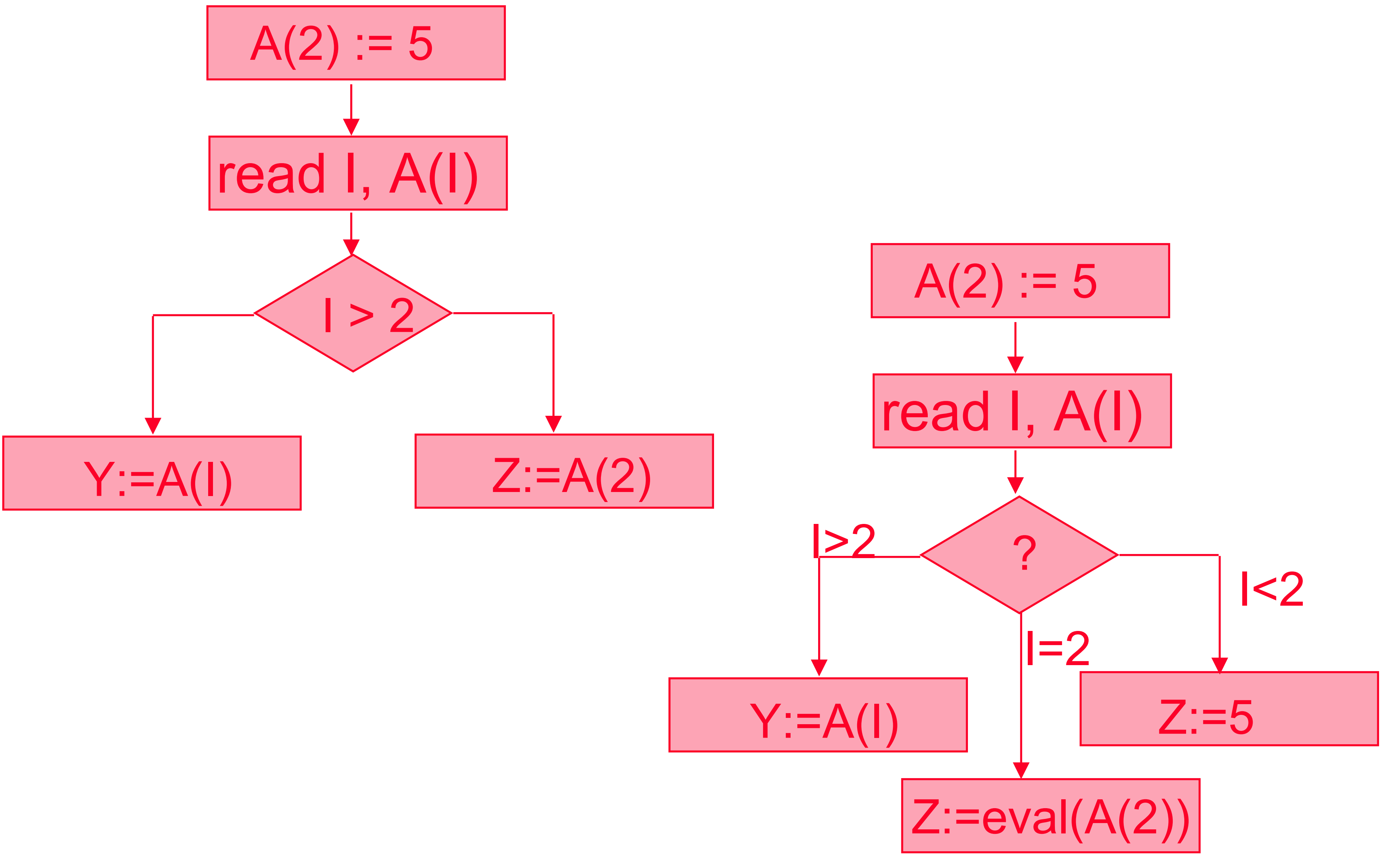
- In compilers
  - May create a faster representation
  - E.g., if you know the maximum size for a platform or domain, hardcode that into the system
  - More than just constant propagation
    - Do symbolic manipulations with the computations

Example with Ackermann's function

- A(m,n) = if m = 0 then n+1 else
  if n = 0 then A(m-1, 1) else
  A(m-1,A(m,n-1))

- A0(n) = n+1
- A1(n) = if n = 0 then A0(1) else
  A0(A1(n-1))
- A2(n) = if n = 0 then A1(1) else
  A1(A2(n-1))

Specialization using partial evaluation



Why is Partial Evaluation Useful in Analysis

- Often can not reason about dynamic information
  - Instantiates a particular configuration of the system that is easier to reason about
  - E.g., the number of tasks in a concurrent system; the maximum size of a vector

- Look at several configurations and try to generalize results
  - Induction
  - Often done informally

Reference on Partial Evaluation

- Neil Jones, An Introduction to Partial Evaluation, ACM Computing Surveys, September 1996