Using CSP refutation capabilities to refine AI-based Approximations

Michel RUEHER

University of Nice Sophia-Antipolis / I3S – CNRS, France

(joined work with **Olivier Ponsini, Claude Michel**)

November, 2011

INFORMS Charlotte – 2011

"Session TA08: Hybrid Methods III: Applications"

Outline

Verifying programs with floating-point computations

Context :

- *Embedded Systems* (transportation, nuclear energy...) rely more and more on floating-point computations
- *C language* is widely used for such applications

Floats : an additional source of errors

Problems with floating-point numbers

- *Counter intuitive properties* and "pitfalls" of Floating-point arithmetic:
  - Arithmetic operators are neither associative nor distributive
  - Reasoning with *rounding*, absorption, cancellation

- *Examples* (in simple precision)
  - Absorption : $10^7 + 0.5 = 10^7$
  - Cancellation : $((1 - 10^{-7}) - 1) * 10^7 = -1.192... \neq -1$
  - $(10000001 - 10^7) + 0.5 \neq 10000001 - (10^7 + 0.5)$
  - $0.1 = 0.000110011001100\ldots$

Semantics of program with floating-point numbers

Programs are run on the floats but:

- *Specification*, *properties* of programs
  $\rightsquigarrow$ Users are *reasoning with real numbers*
- *Programs* are sometimes written with the semantics of real numbers "in mind"
- *Differences* between computations over real numbers and computations over the floats
  $\rightarrow$ reveal *problems with floats*

Abstract Interpretation

$\rightarrow$ *Approximations* of computations over *floats*
$\rightarrow$ *Approximations* of computations over the *real numbers*

How works abstract interpretation ?

Abstract Interpretation *requires*:

1. A *semantics* to compute the states of a program at various checkpoints
2. An *abstraction* to represent the states of a program

$\rightarrow$ *Zonotopes* (in Fluctuat)

Abstract Interpretation *computes a fixed point* of the equations of the semantics

Zonotopes

- **Intuition:** *convex polytopes* with a *central symmetry*
  Sets of *affine forms* $x = x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n$
  with $\varepsilon_i \in [-1, 1]$
- **Advantages:**
  - *Linear correlations* between variables are preserved
  - Nonlinear operations are over-approximated by introducing an error term
  - *Good trade-off* between performance and precision
- **Limits:**
  - Better than the intervals but not as good as polyhedra
  - Not very accurate for nonlinear terms
  - Not accurate on *conditional statements* (domain intersection)

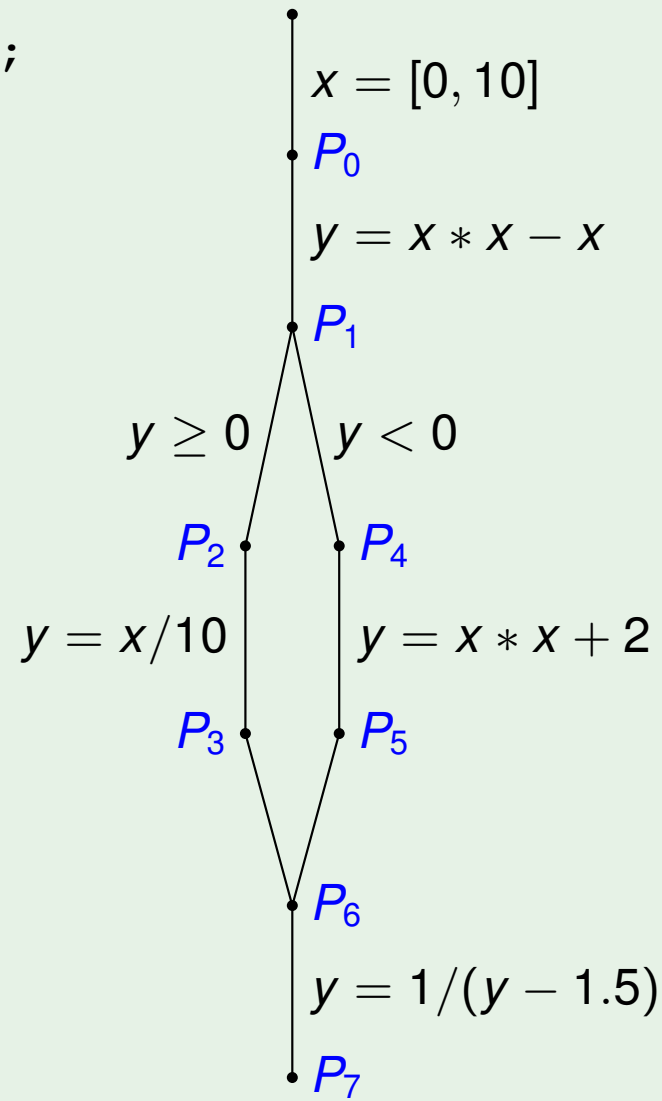
Fluctuat

Static analyzer based on abstract interpretation of C programs
→ using *Zonotopes*
→ estimates *rounding errors and their propagation*

- Programs are considered both:
  - As a *specification* over the reals
  - As an *implementation* over the floats
- Fluctuat computes:
  - An *over-approximation* of the domain-bounds of each variable considered as a *real number*
  - An *over-approximation* of the domain-bounds of each variable considered as a *floating-point number*
  - An *over-approximation* of the *error* associated with the variable (difference between floating-point and real number values)
  - The *contribution of each instruction* to the error

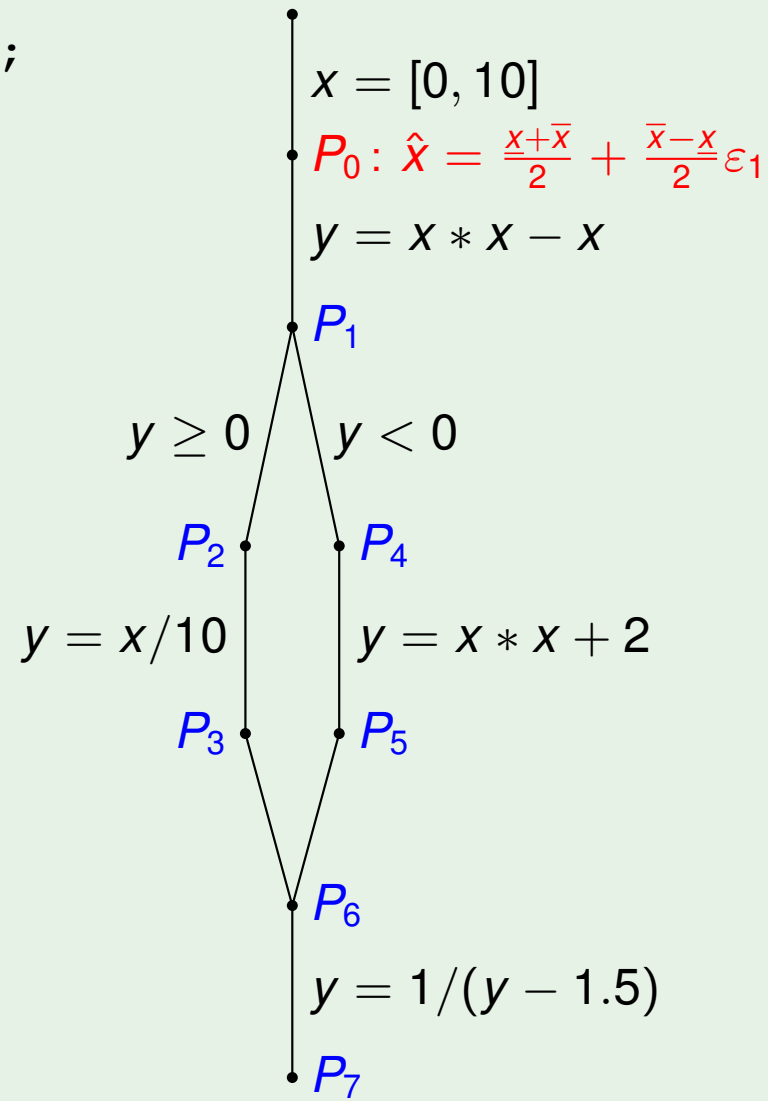
Example: How works abstract interpretation

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```



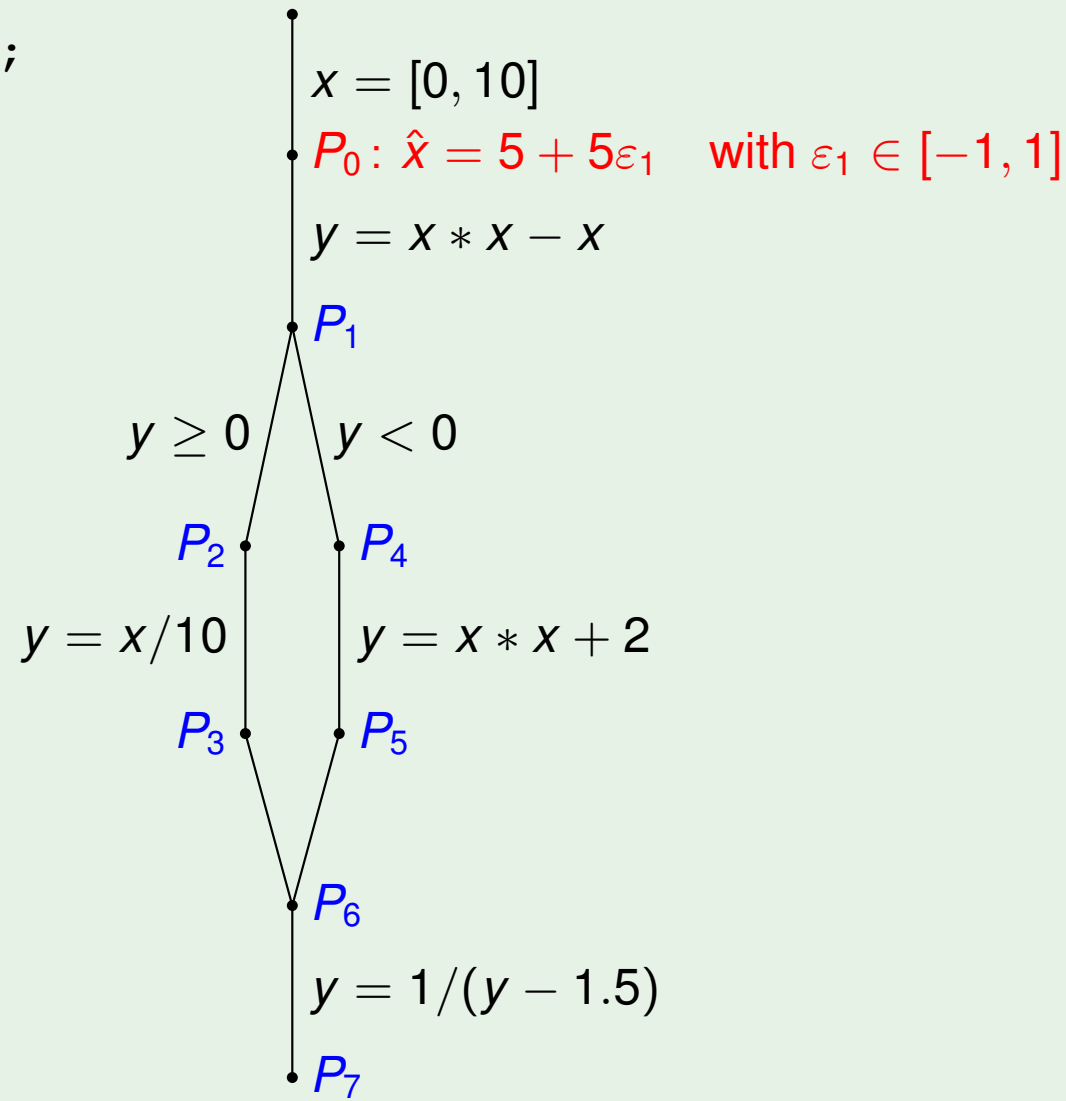
Example: How works abstract interpretation

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```



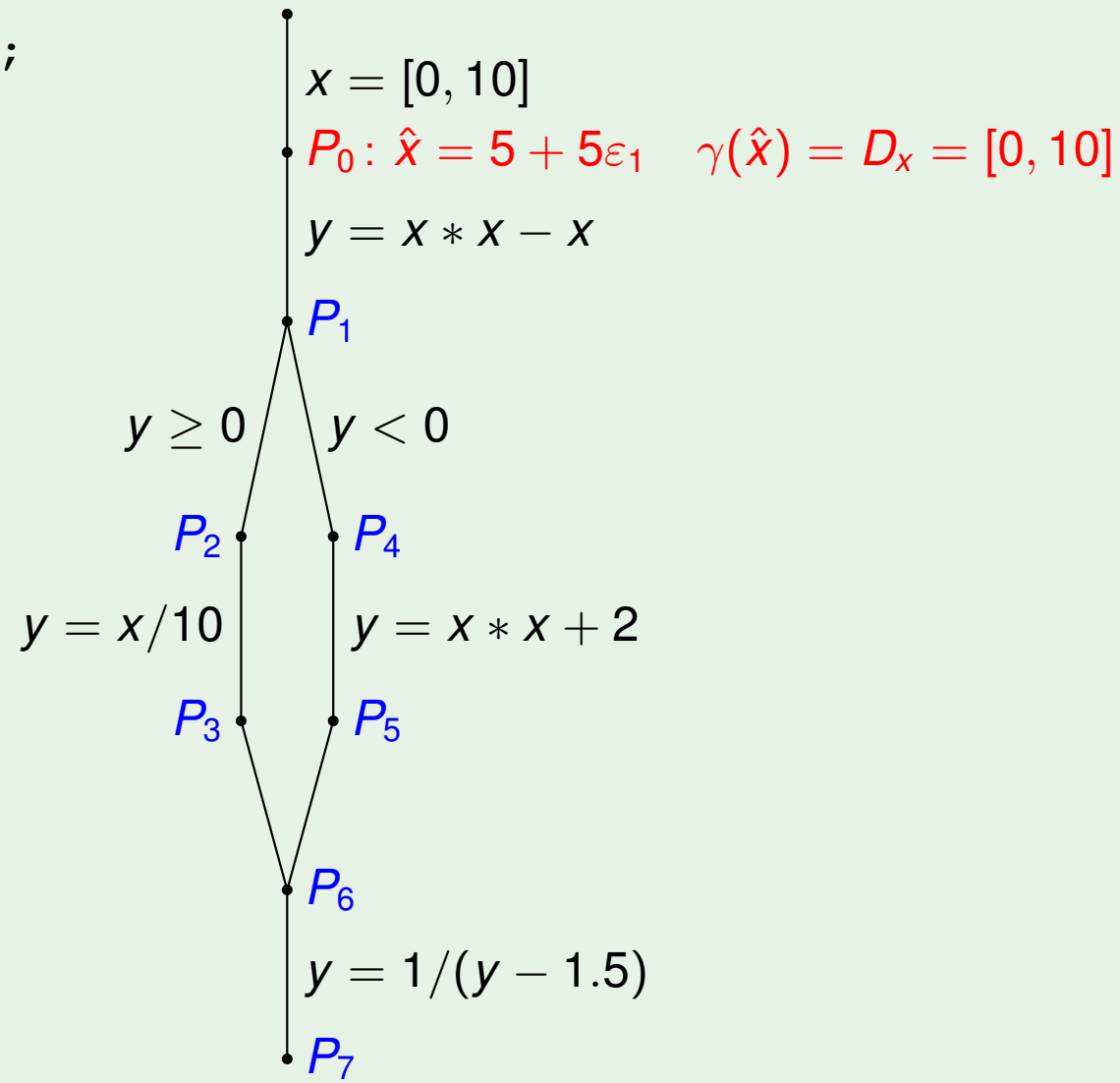
Example: How works abstract interpretation

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```



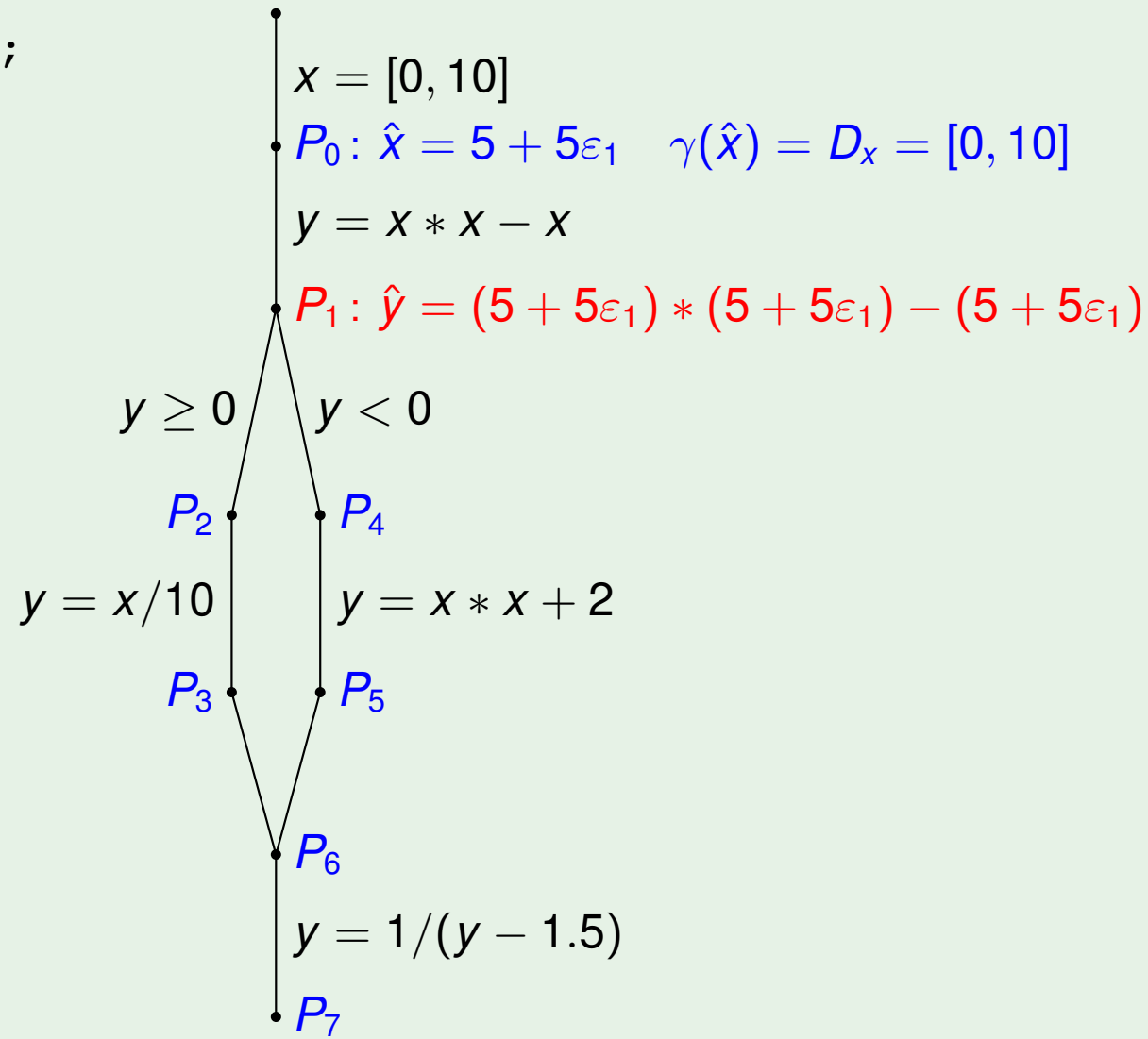
Example: How works abstract interpretation

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```



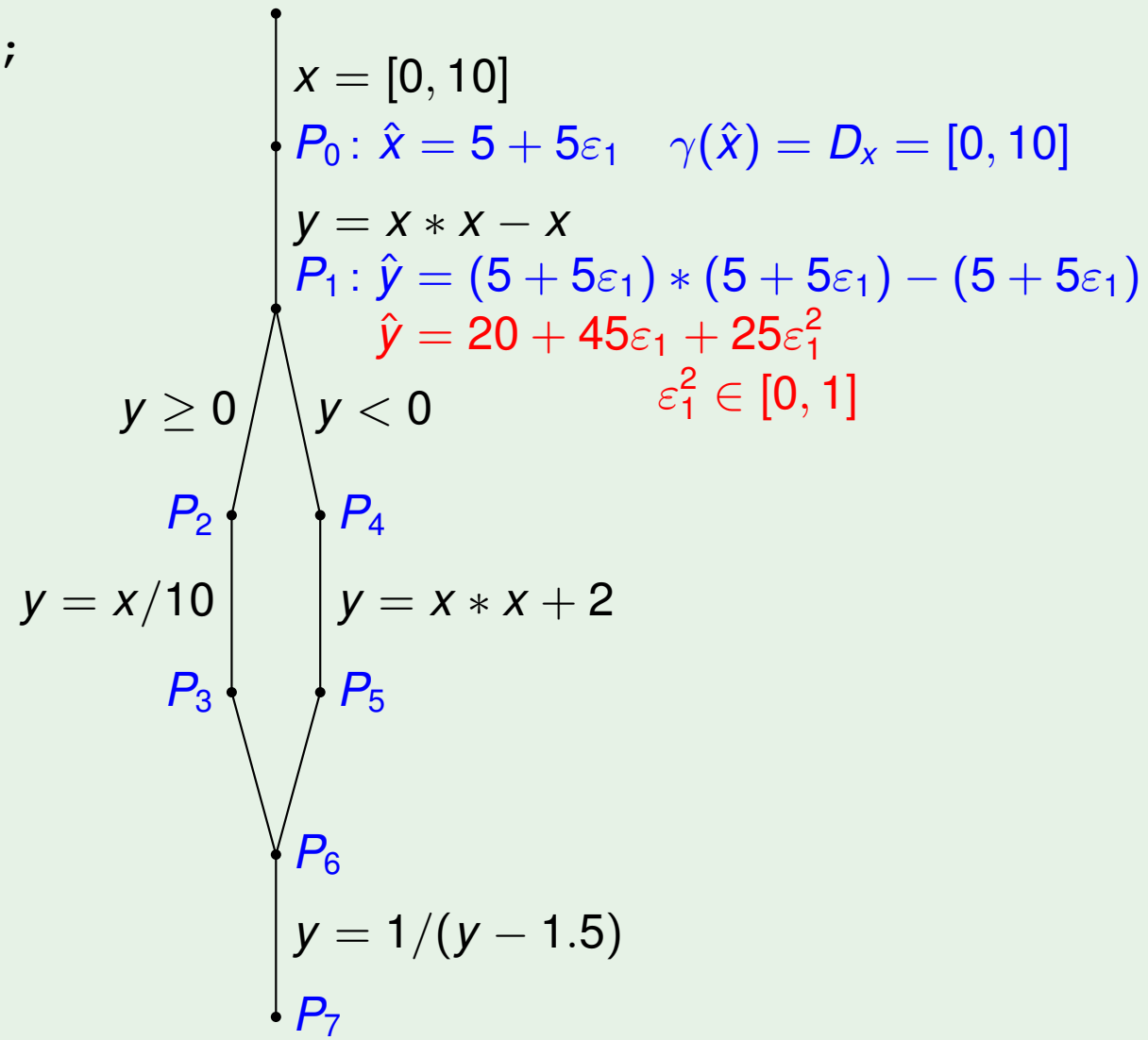
Example: How works abstract interpretation

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```



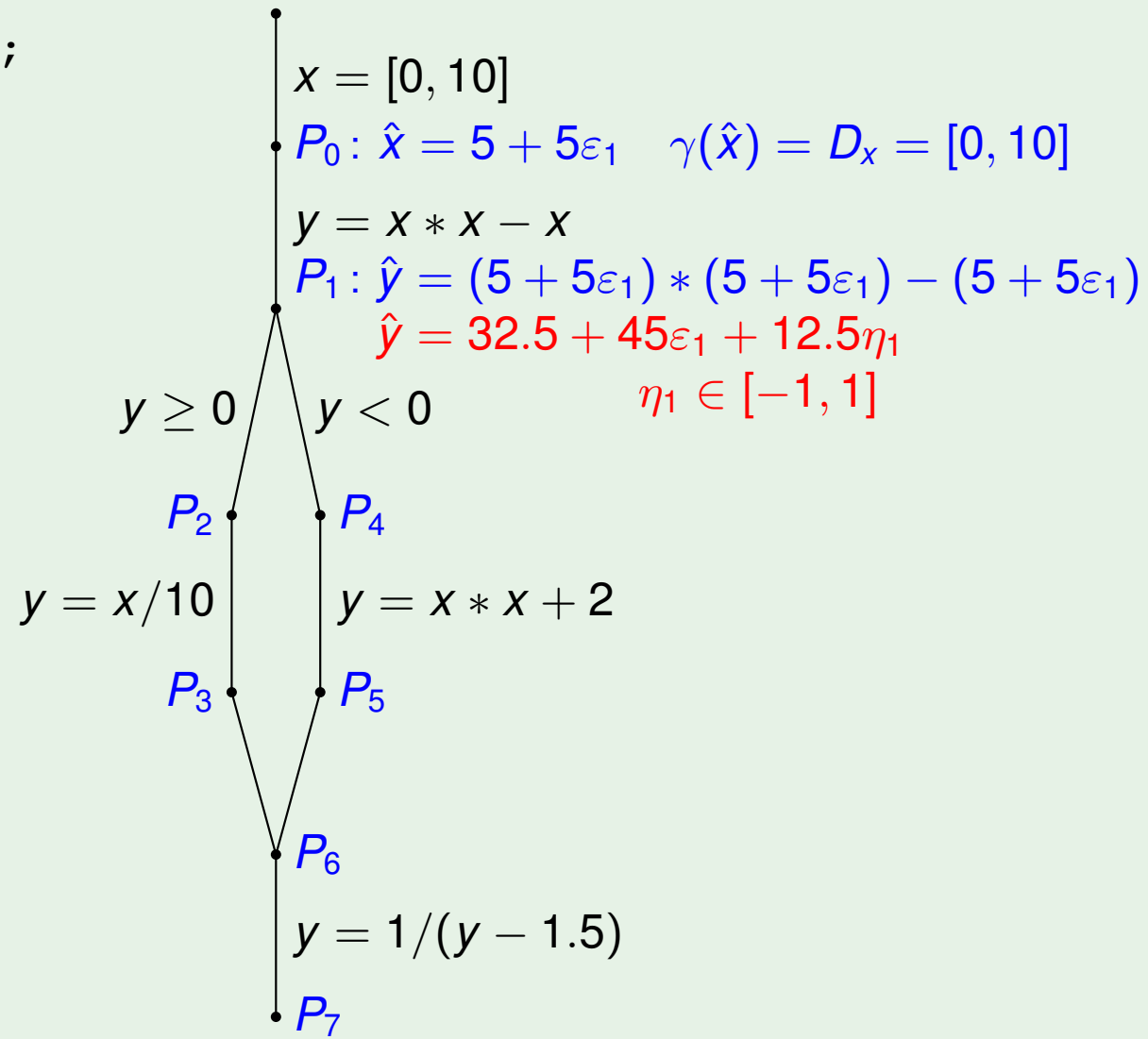
Example: How works abstract interpretation

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```



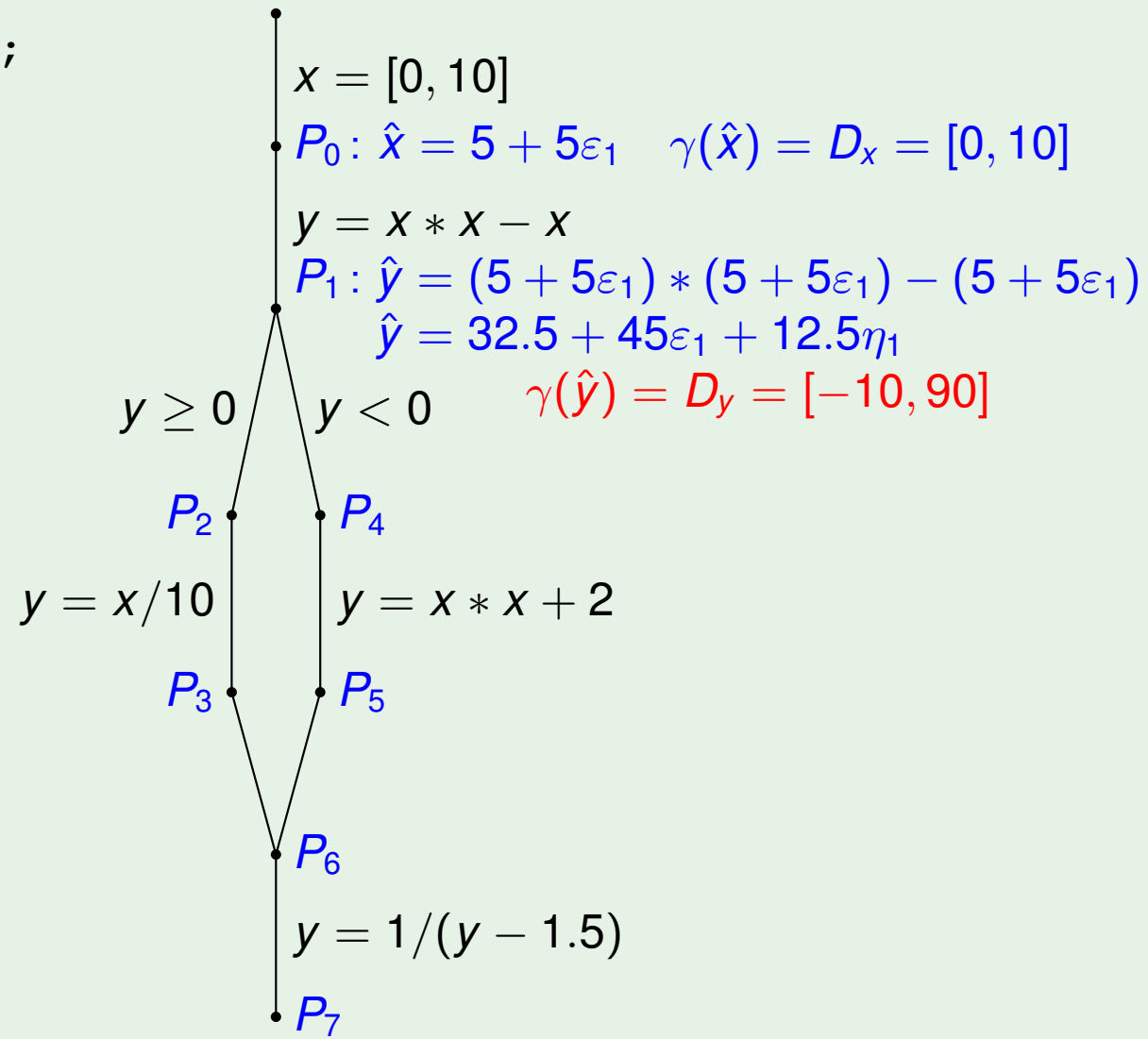
Example: How works abstract interpretation

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```



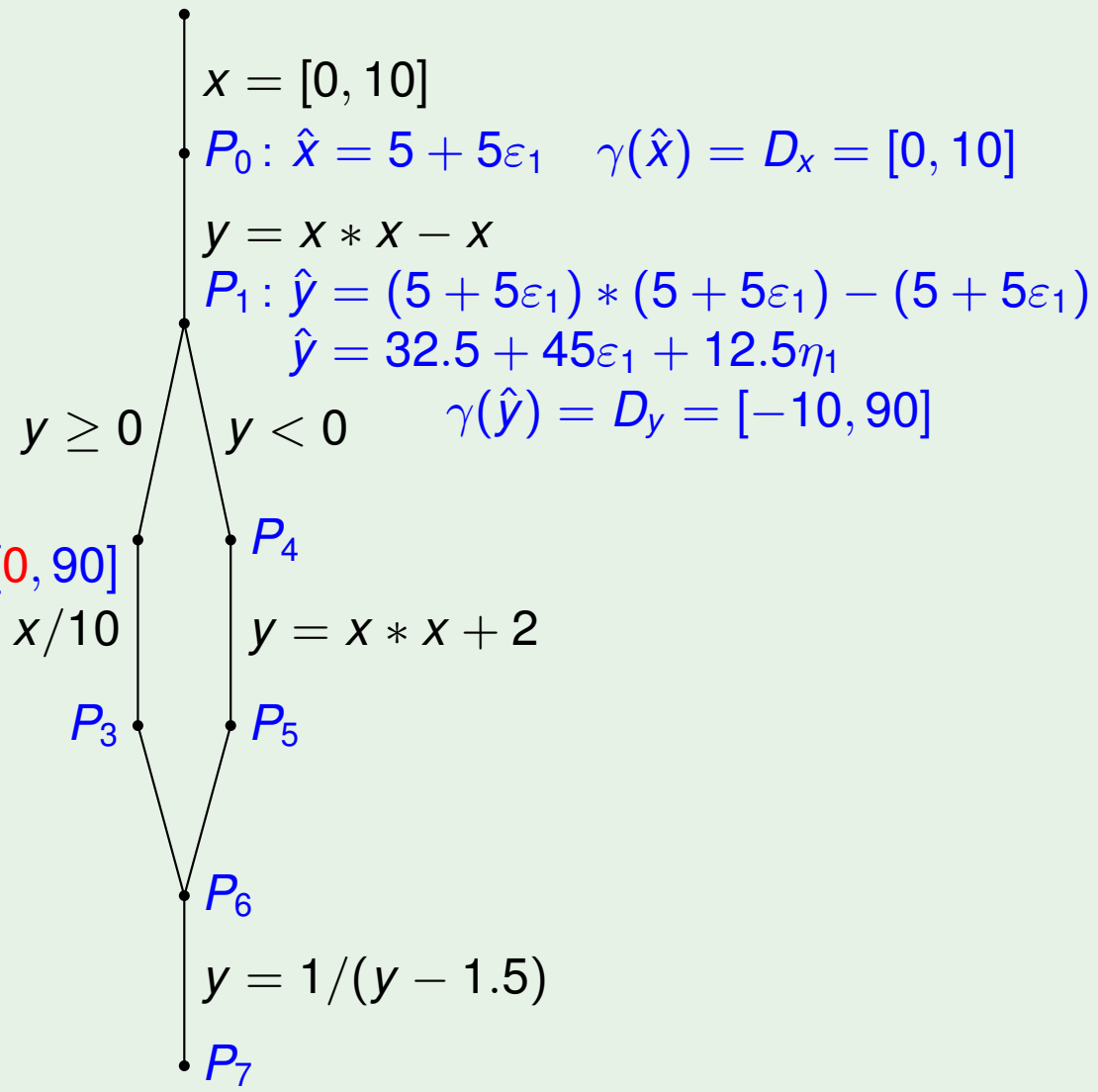
Example: How works abstract interpretation

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```



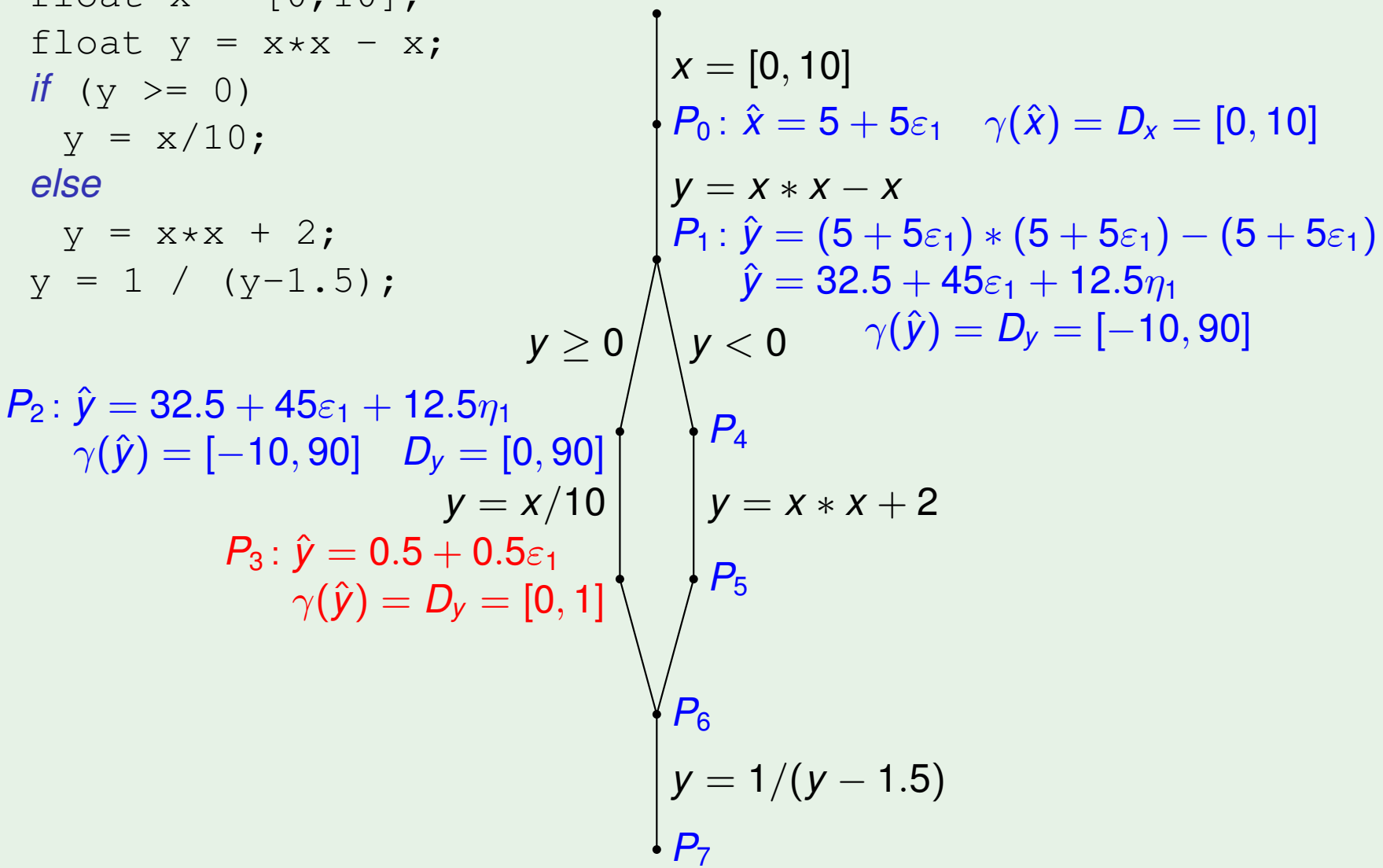
Example: How works abstract interpretation

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```



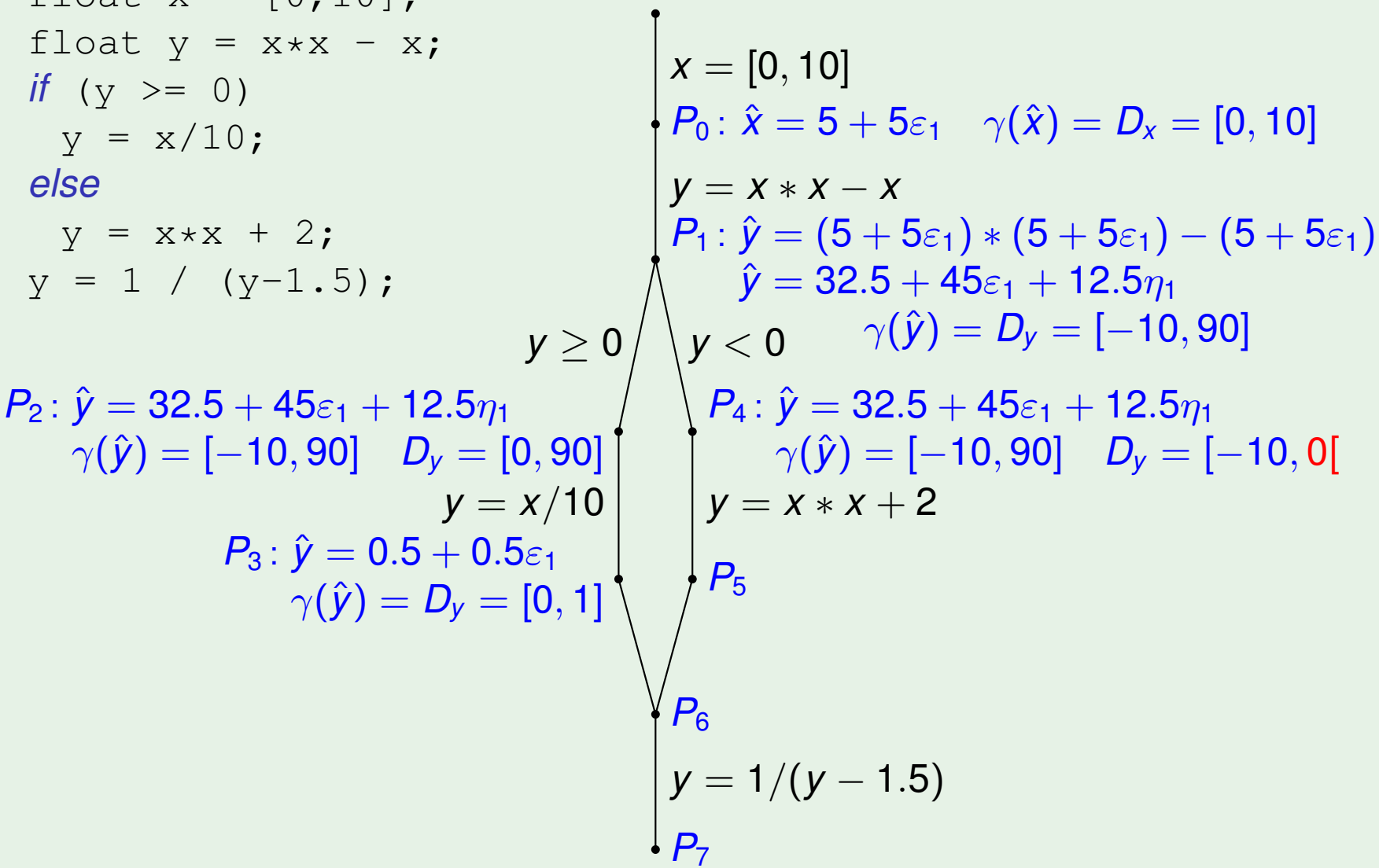
Example: How works abstract interpretation

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```



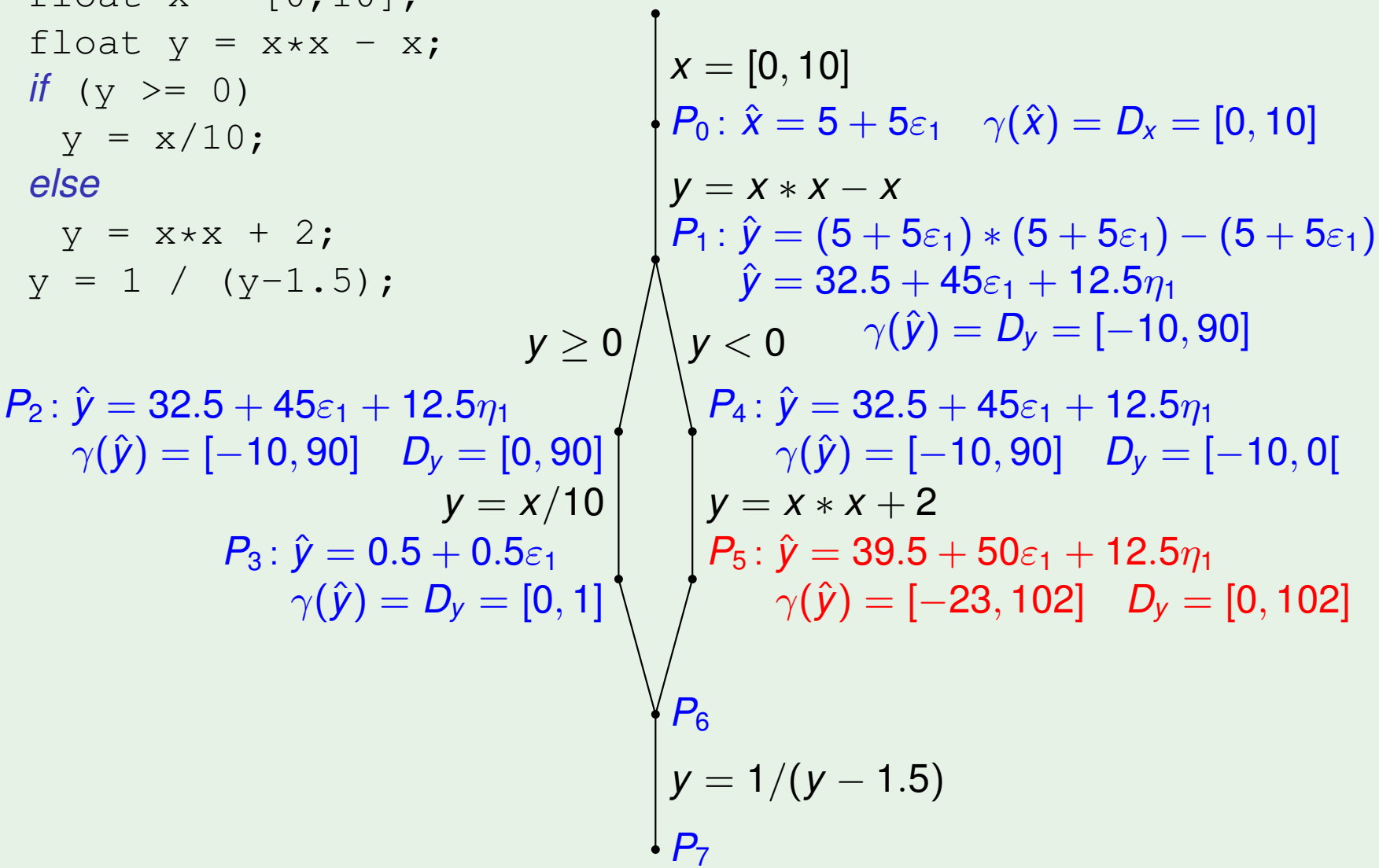
Example: How works abstract interpretation

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```



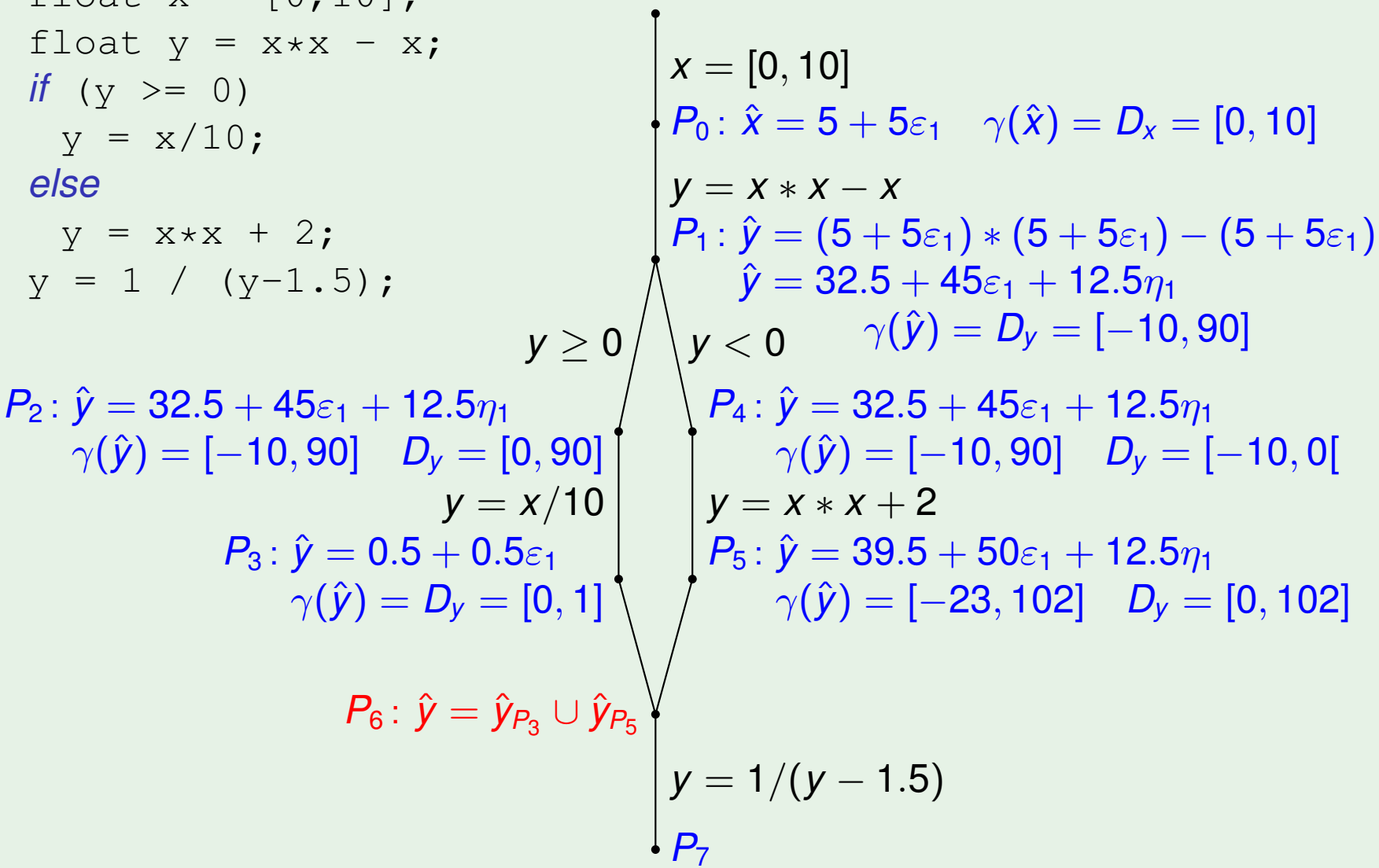
Example: How works abstract interpretation

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```



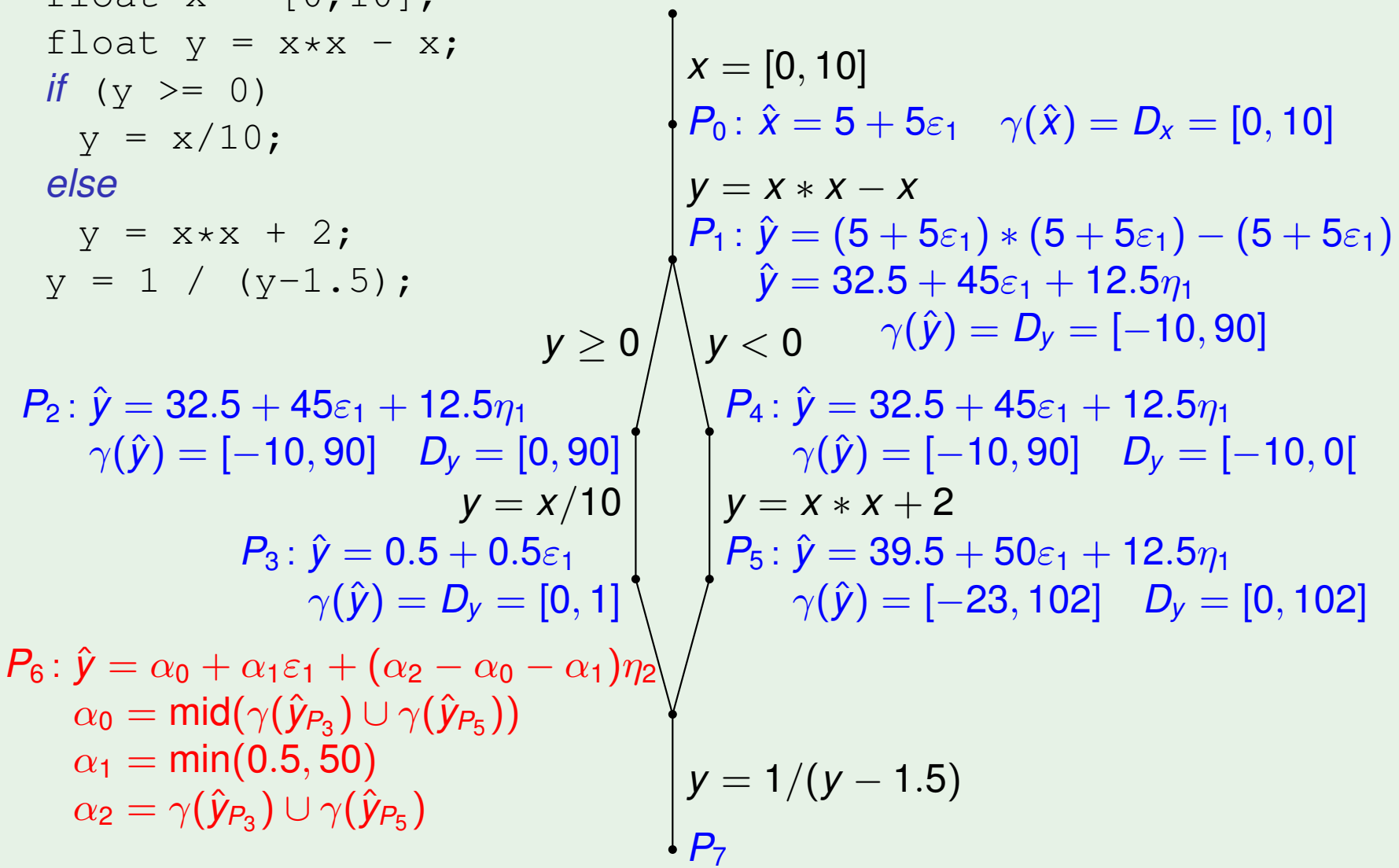
Example: How works abstract interpretation

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```



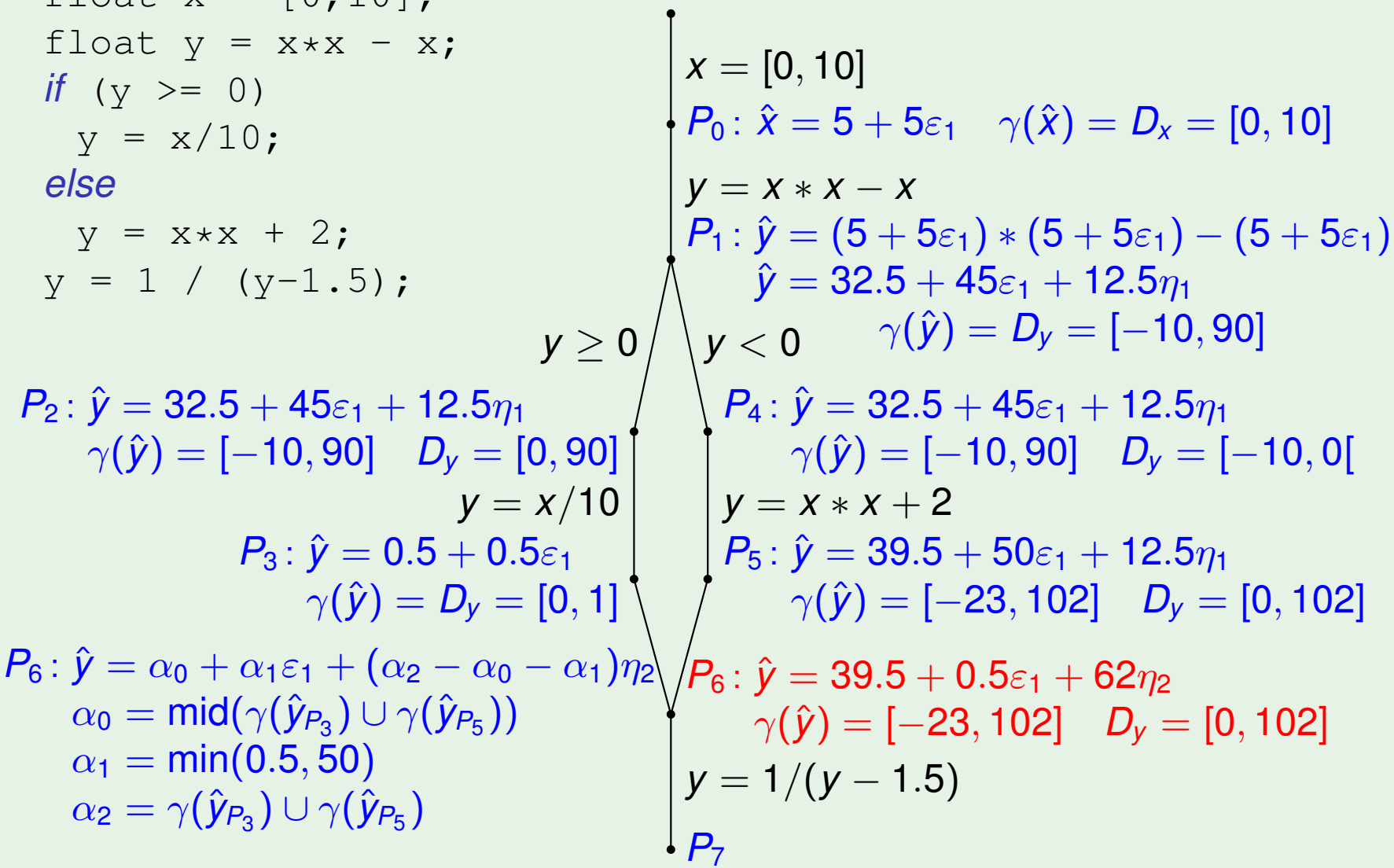
Example: How works abstract interpretation

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```



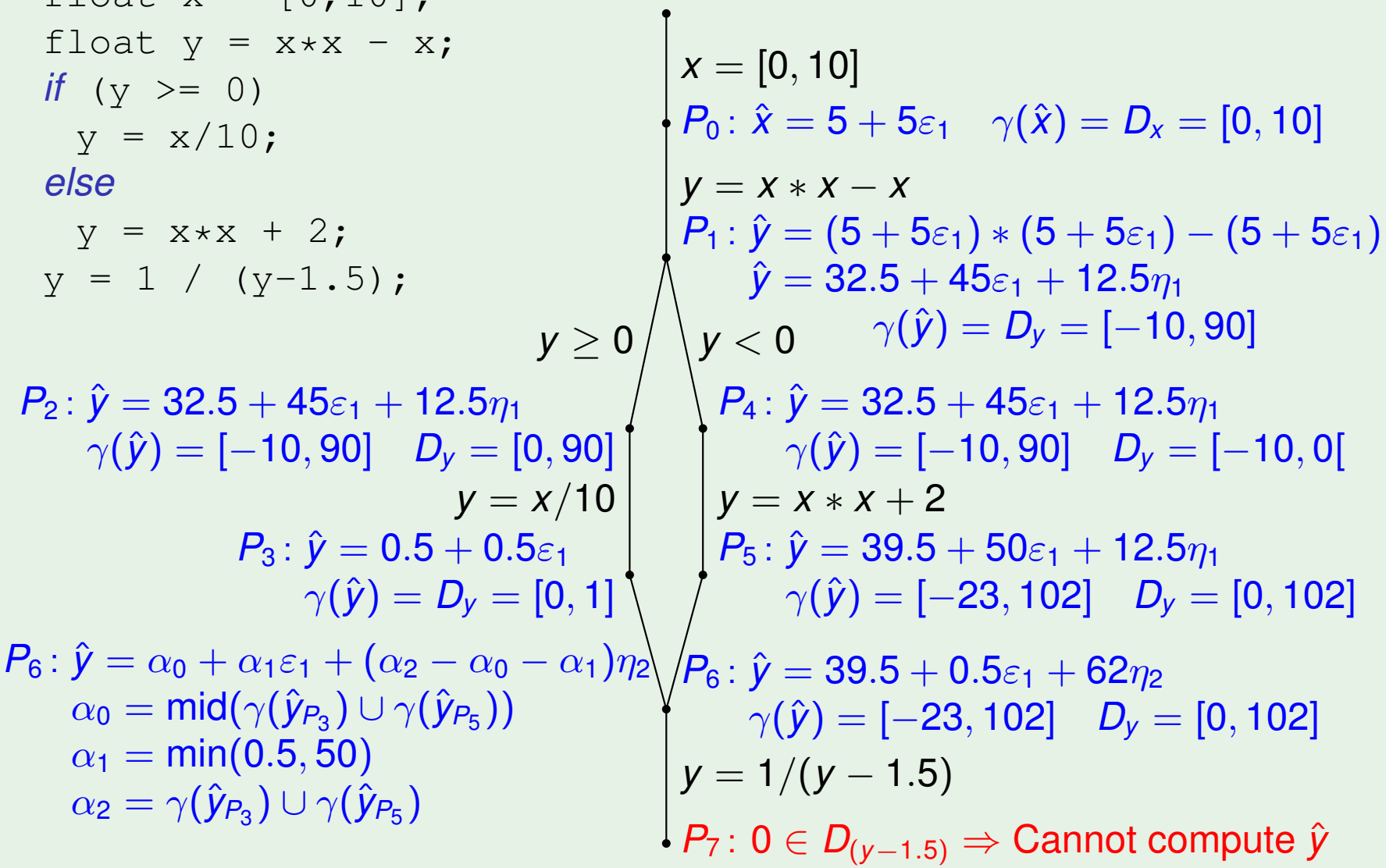
Example: How works abstract interpretation

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```



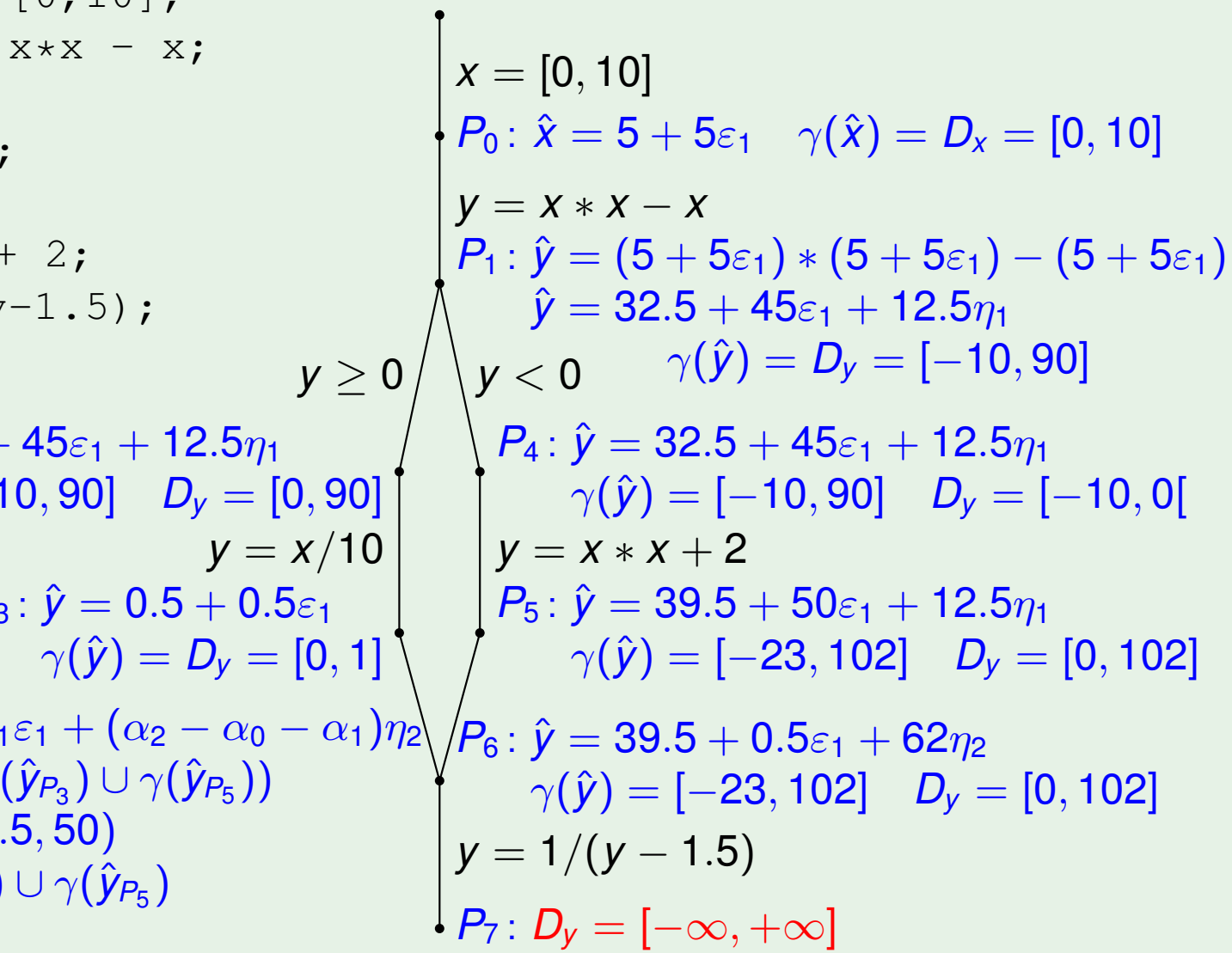
Example: How works abstract interpretation

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```



Example: How works abstract interpretation

```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```



CP framework

Overview

- *Set of CSP* generated for a C program:
  - A CSP is built *"on the fly"* while exploring a path
    Inconsistent CSP $\rightarrow$ current *path is cut off*
  - Loops are unfolded a finite number of times
    $\rightarrow$ *Bounded program verification* (the array lengths, the variable values and the loops are bounded)

- *Filtering:*
  - **Reals:** *Hull & Box consistency* – RealPaver
  - **Floats:** *3B consistency* – FPCS

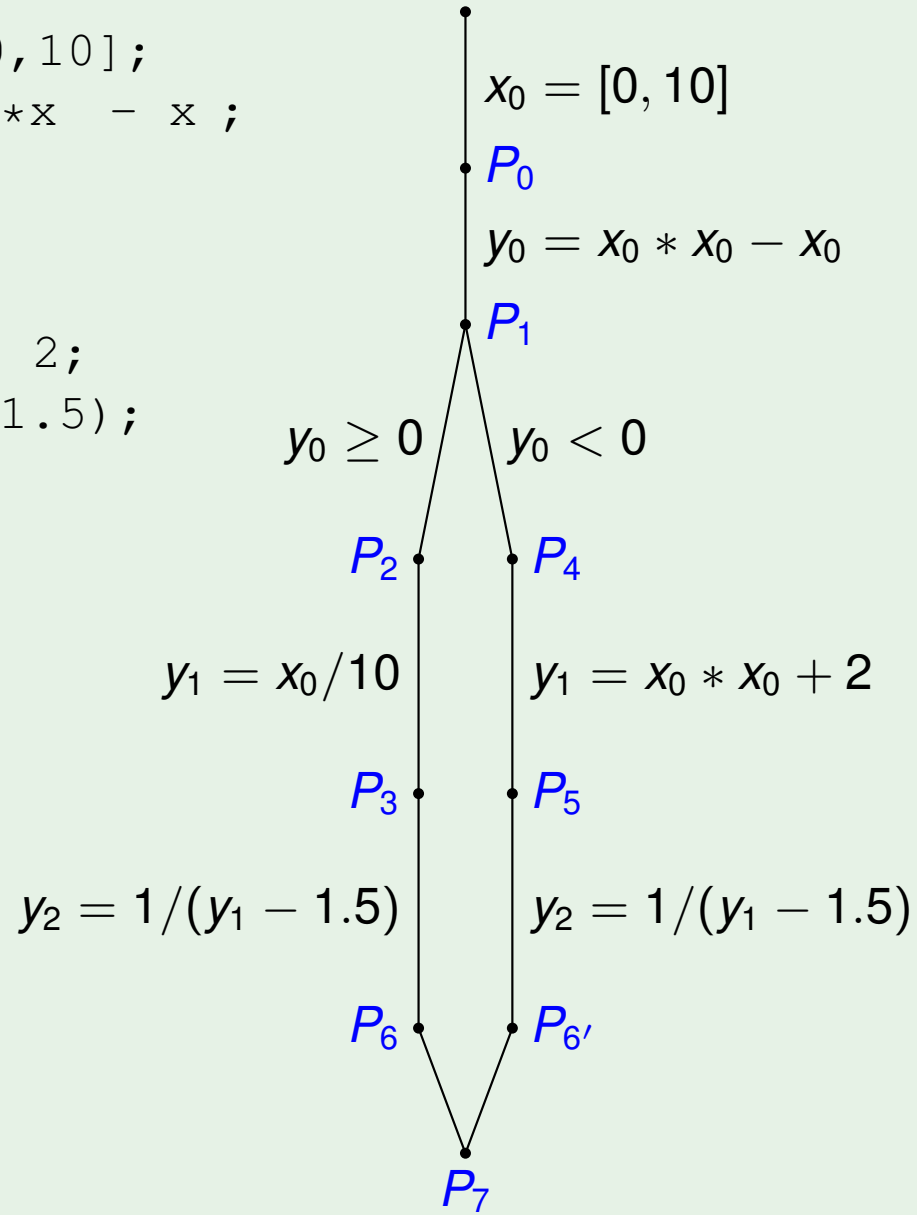
- *Reduced domain of a variable: union of the intervals* generated for this variable while filtering *all successful paths* of the program

CP framework, Pre-processing

1. P is *unwound k times*
2. *Dynamic Single Assignment form* (each variable is assigned exactly once on each program path)
3. *Simplification* of the CFG according to a specific property (slicing techniques)
4. Domains of all variables are filtered by *propagating constant values* along the simplified CFG

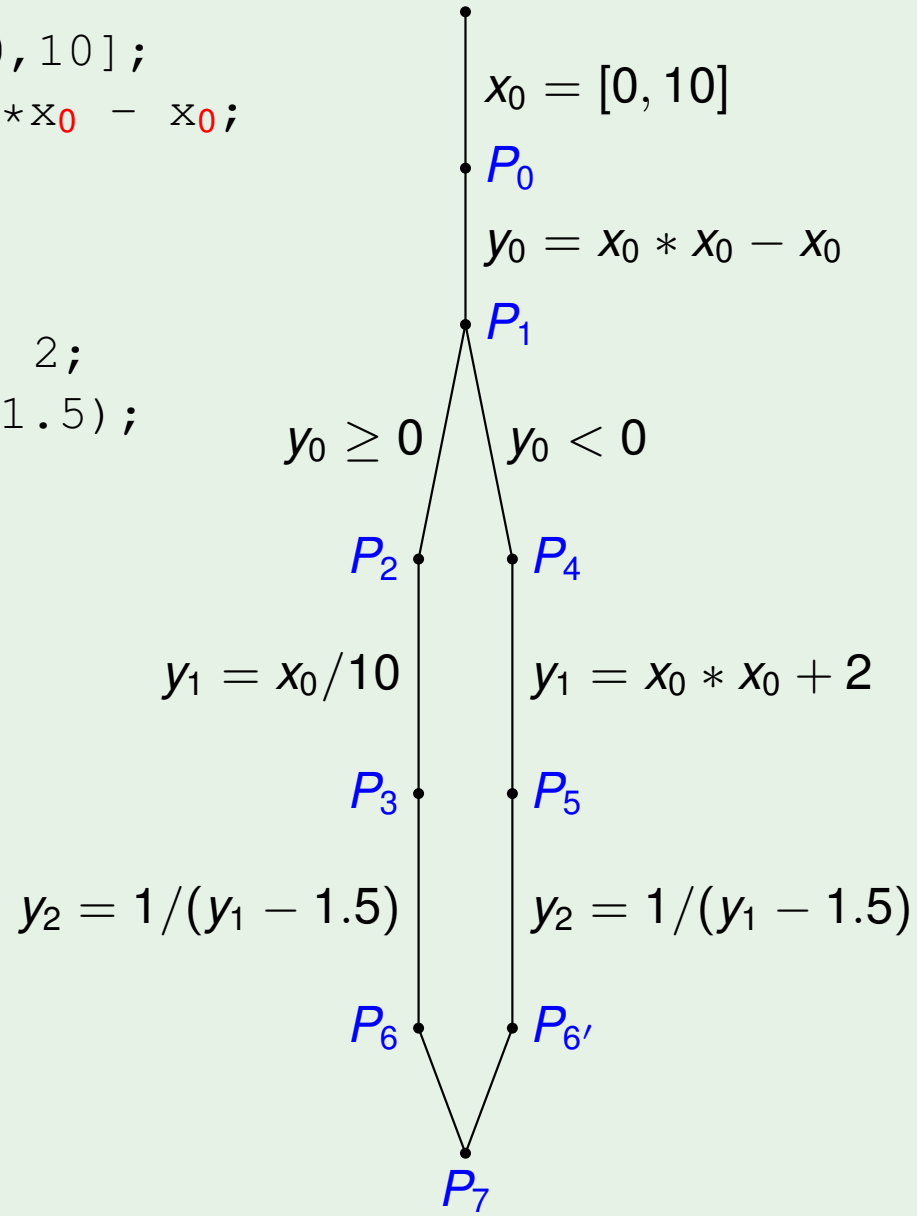
Example: What could CP do?

```
float x  = [0,10];
float y  = x *x  - x ;
if (y  >= 0)
  y  = x /10;
else
  y  = x *x  + 2;
y  = 1 / (y -1.5);
```



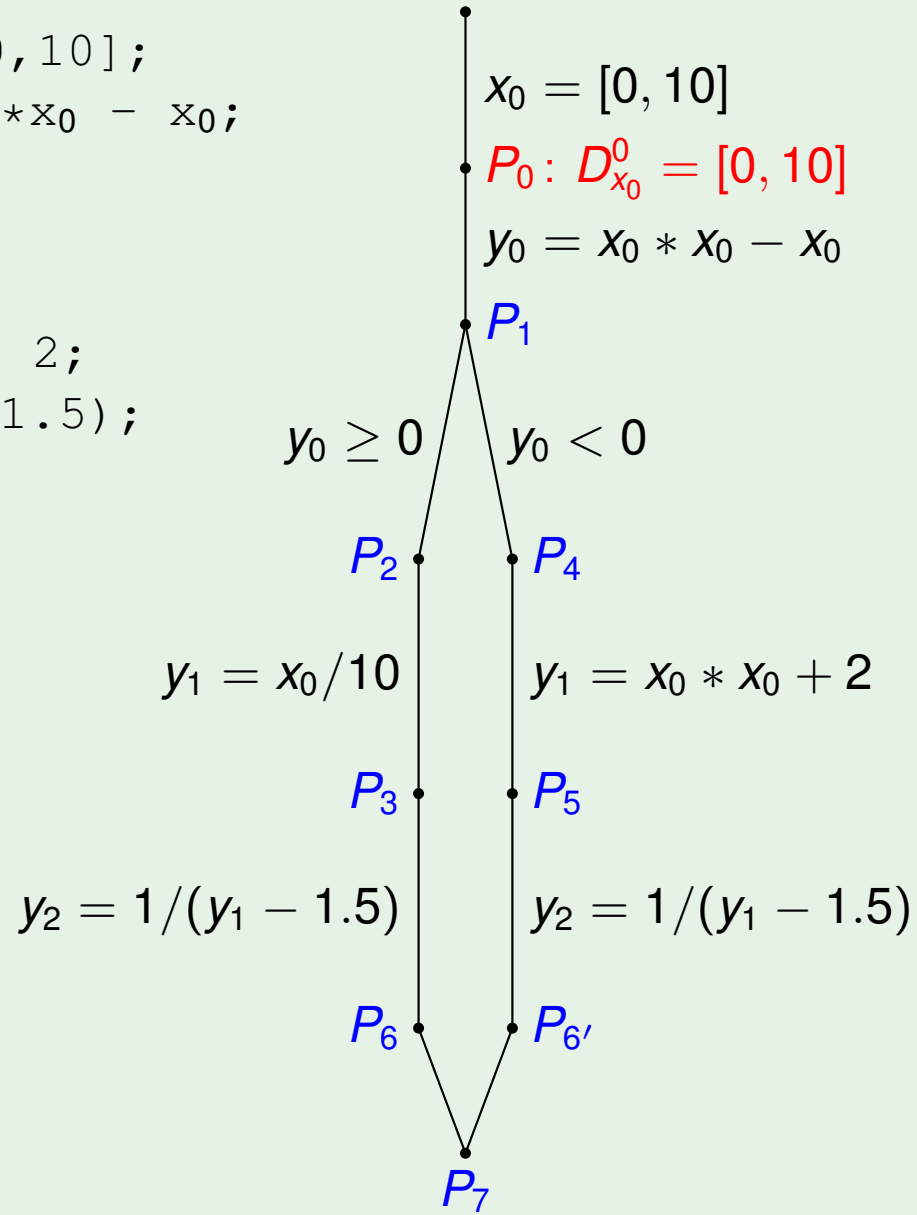
Example: What could CP do?

```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



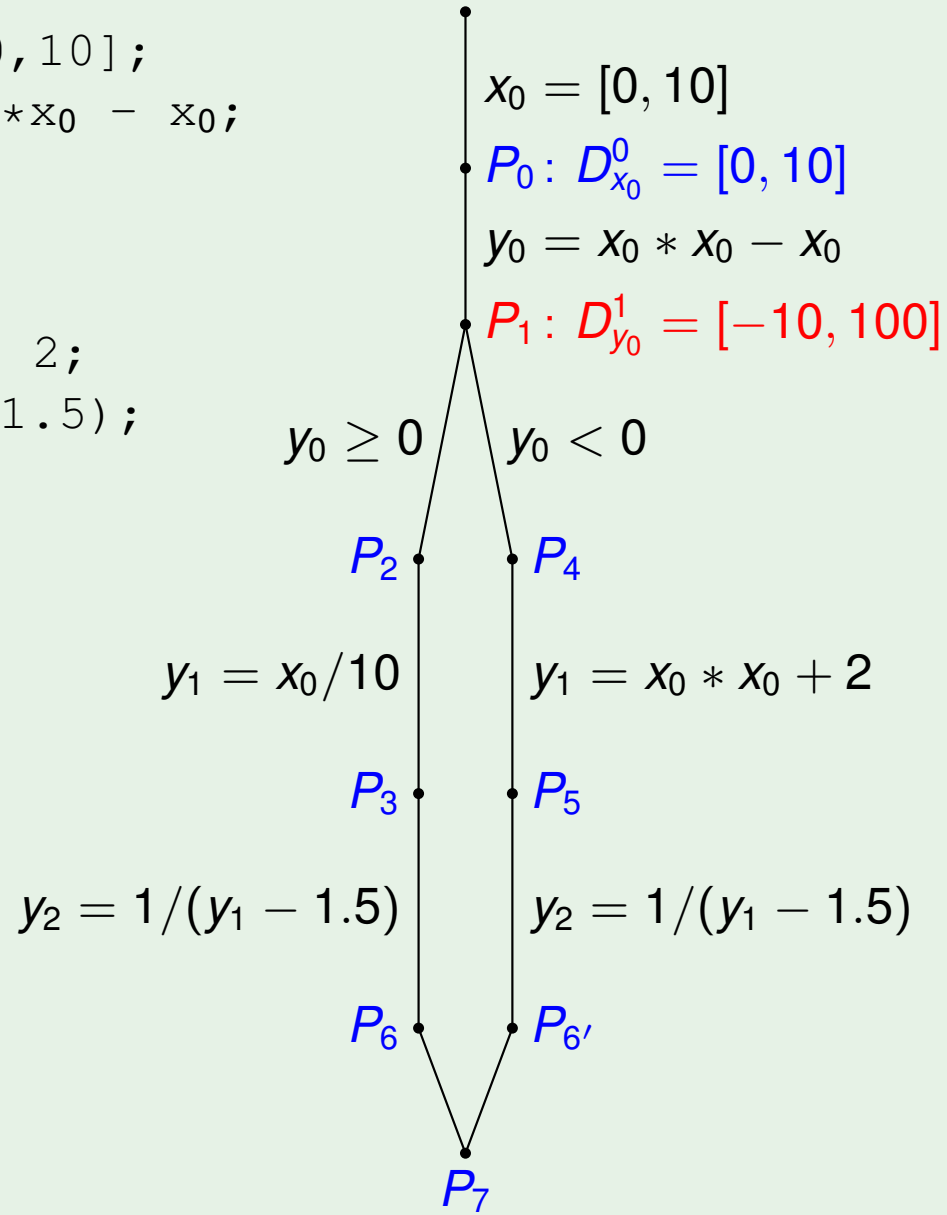
Example: What could CP do?

```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



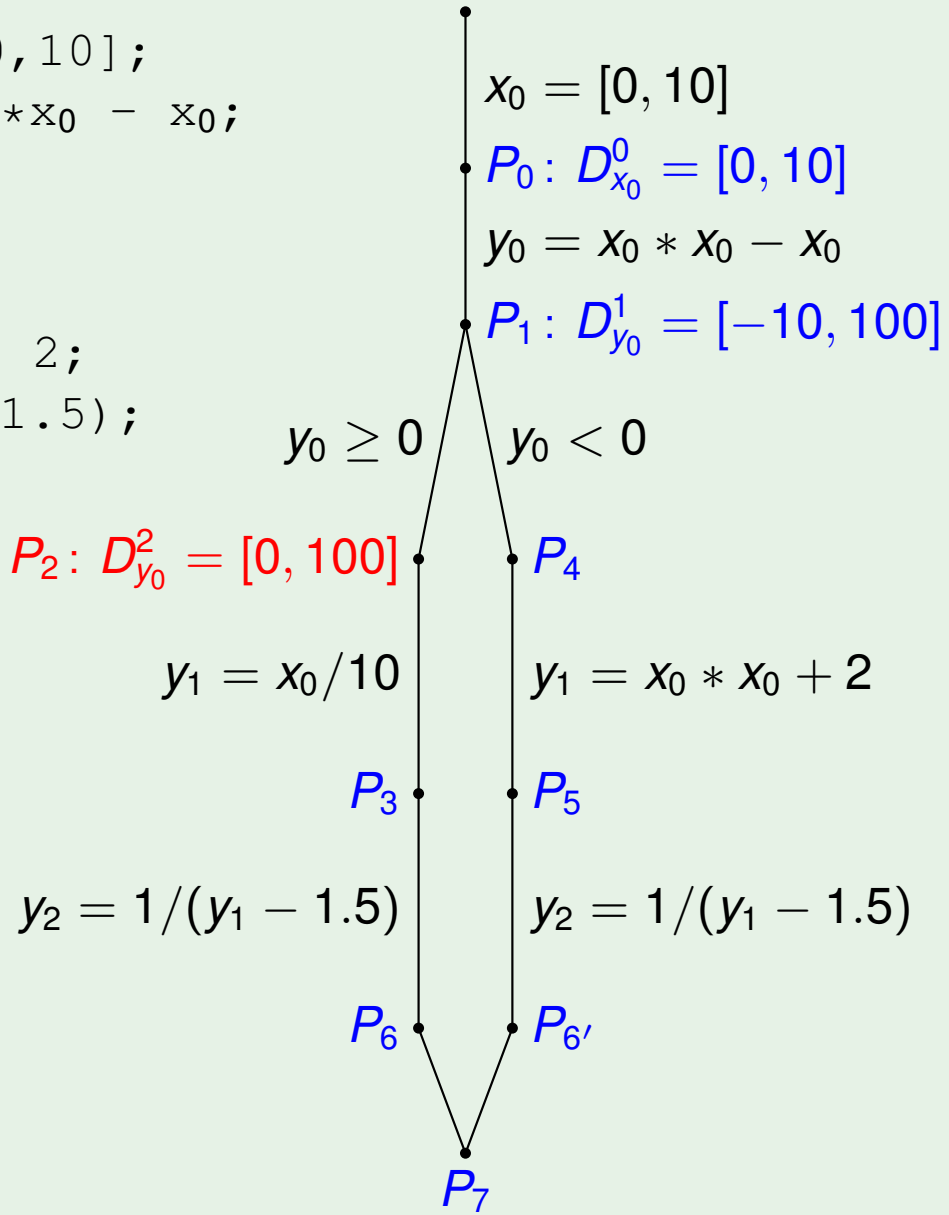
Example: What could CP do?

```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



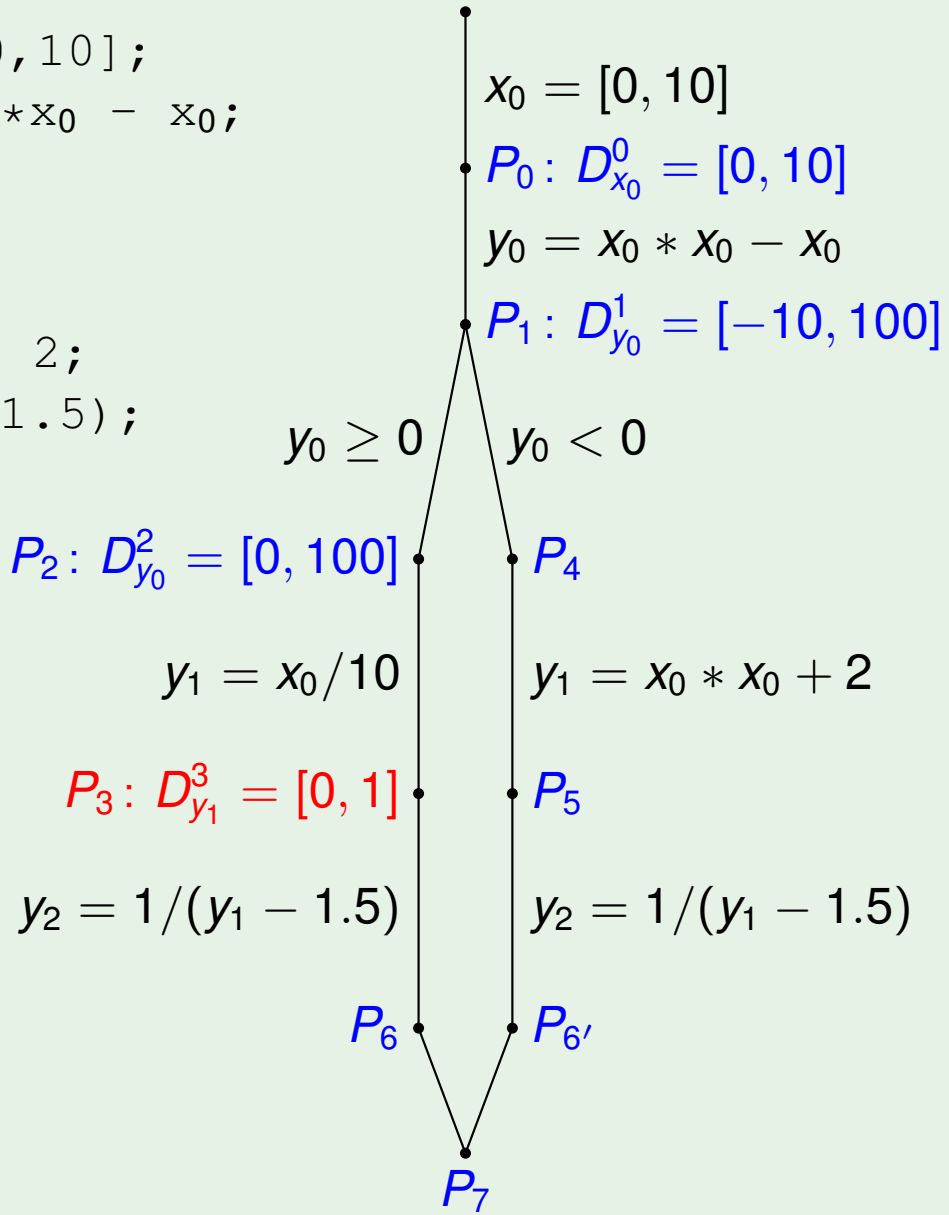
Example: What could CP do?

```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



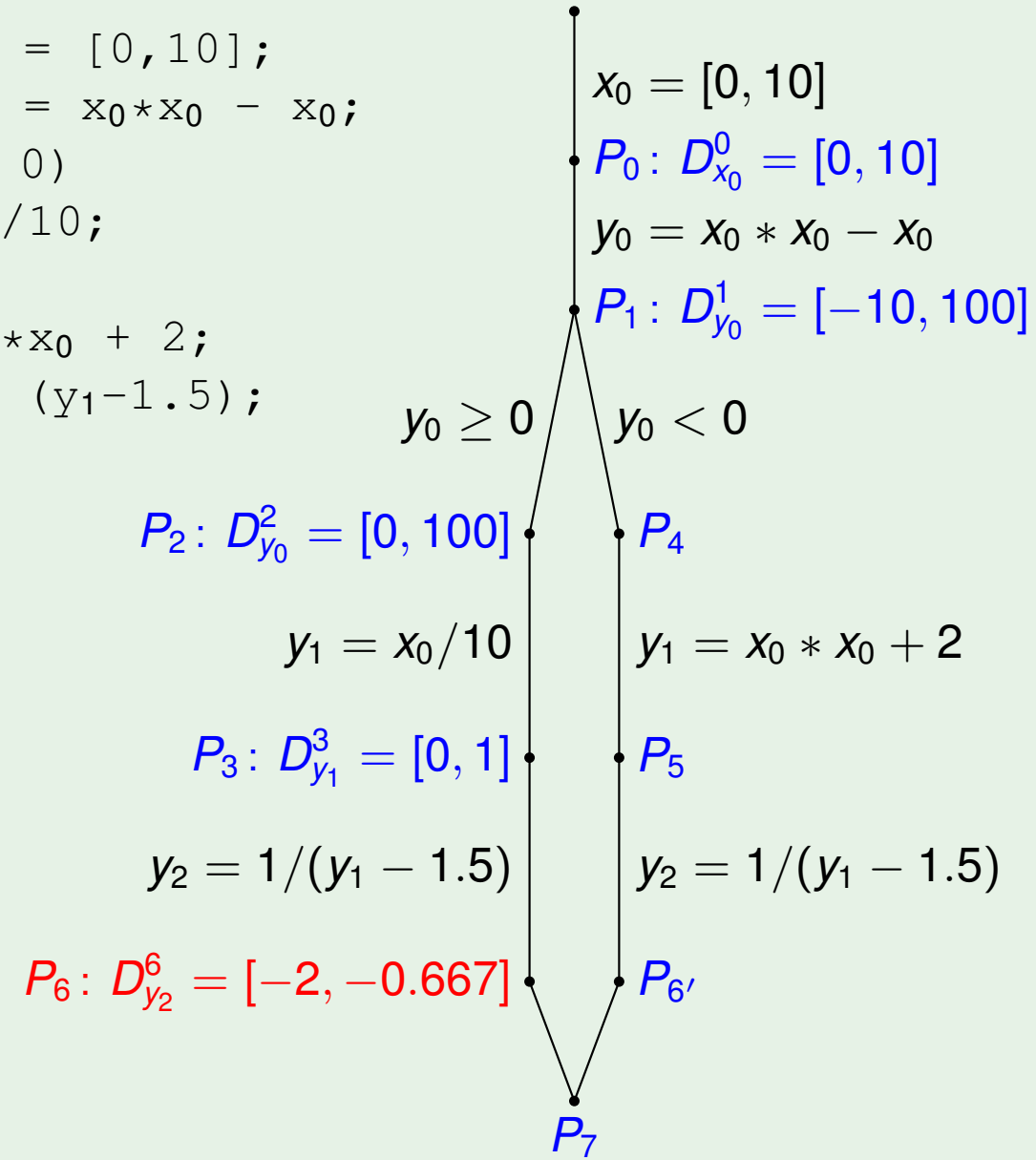
Example: What could CP do?

```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



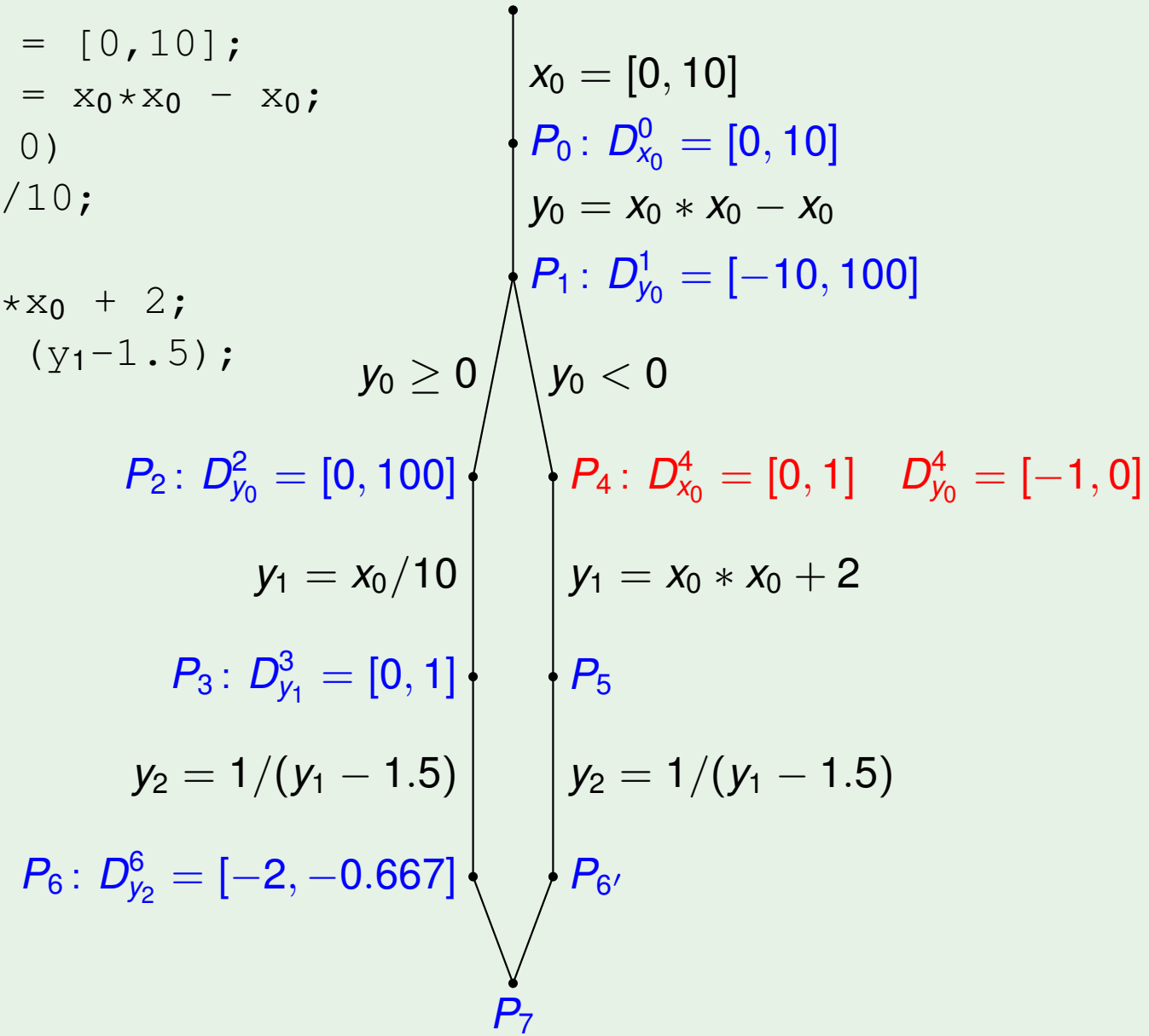
Example: What could CP do?

```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



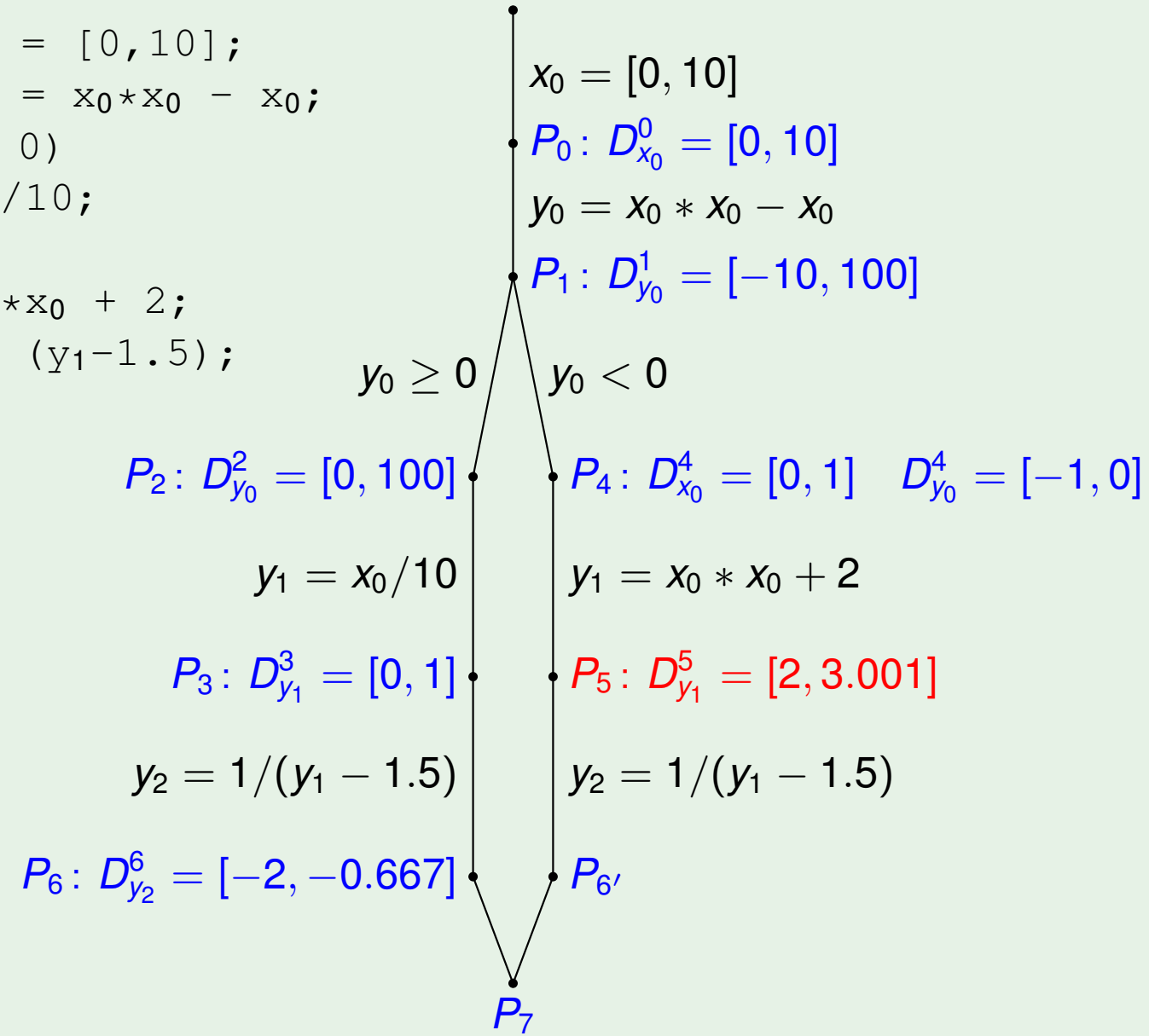
Example: What could CP do?

```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



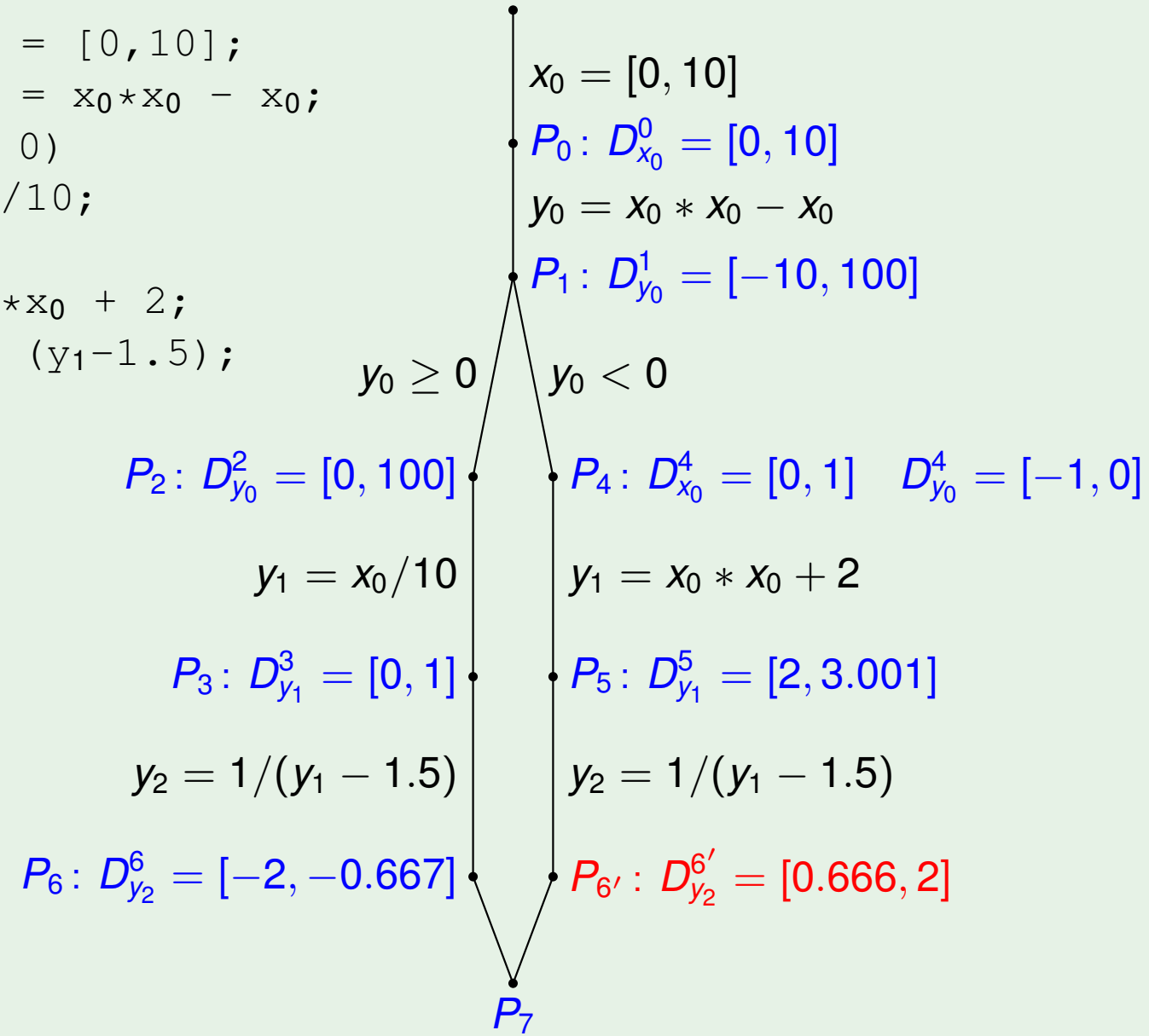
Example: What could CP do?

```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



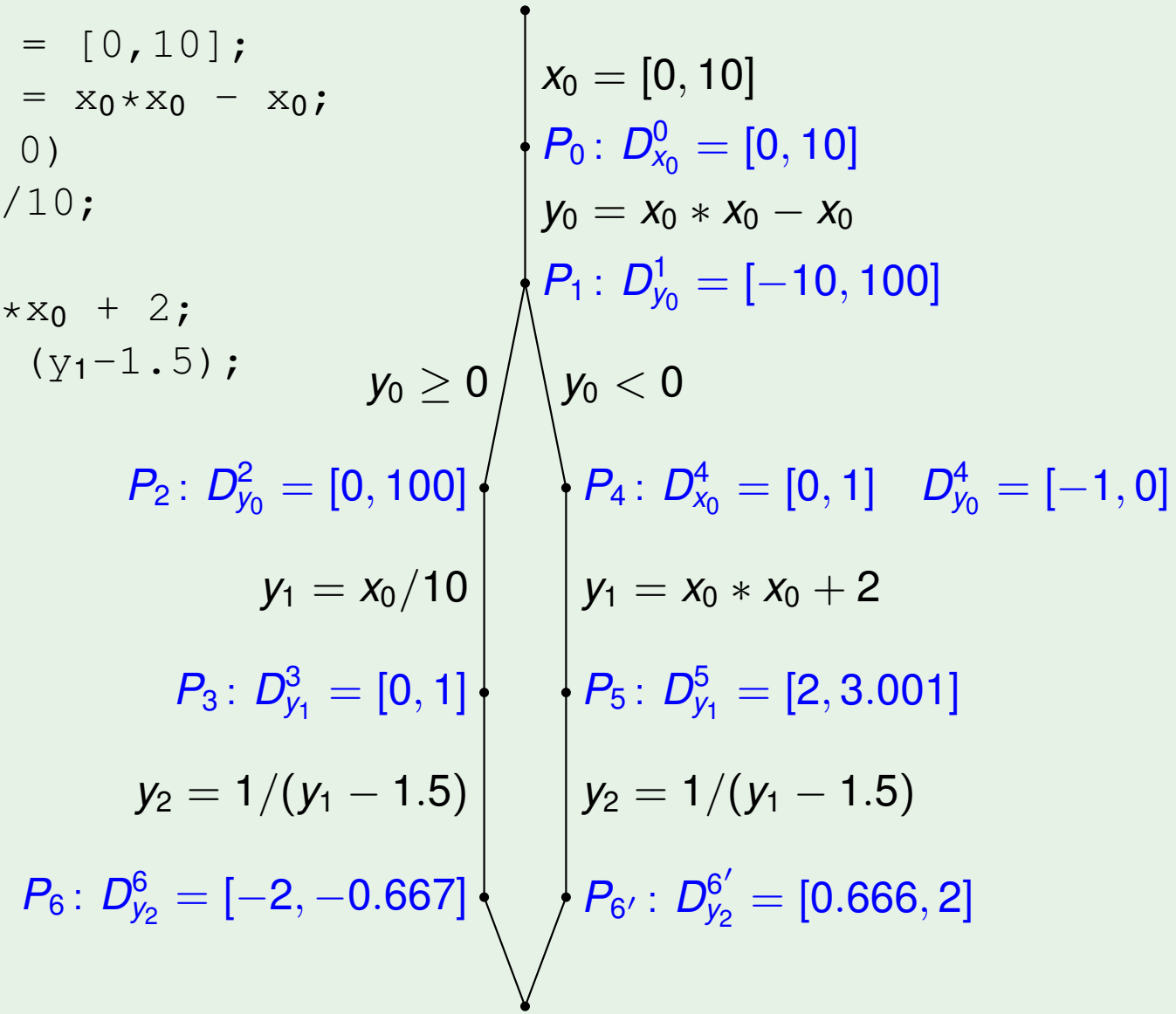
Example: What could CP do?

```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



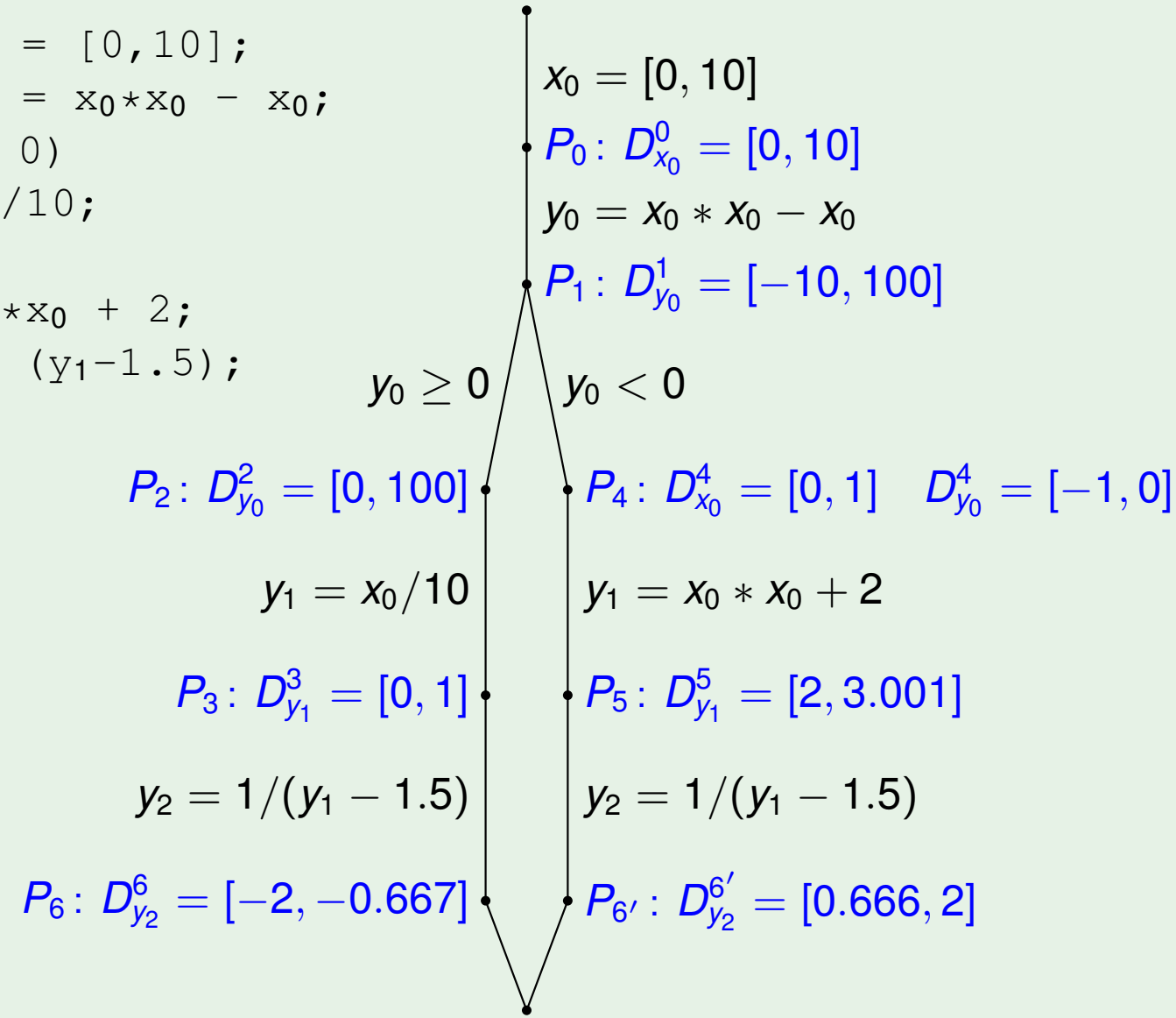
Example: What could CP do?

```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



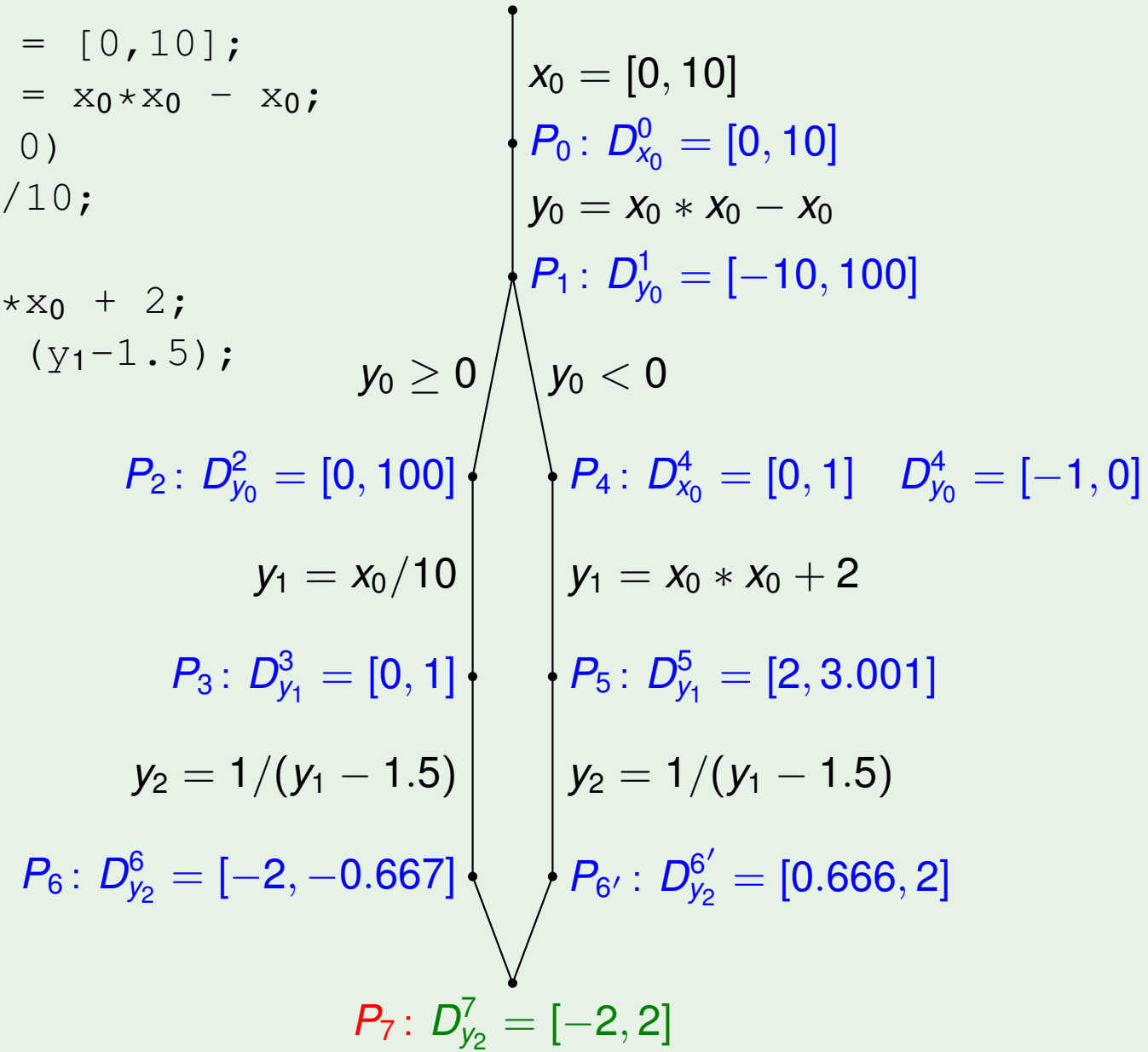
Example: What could CP do?

```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



Example: What could CP do?

```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



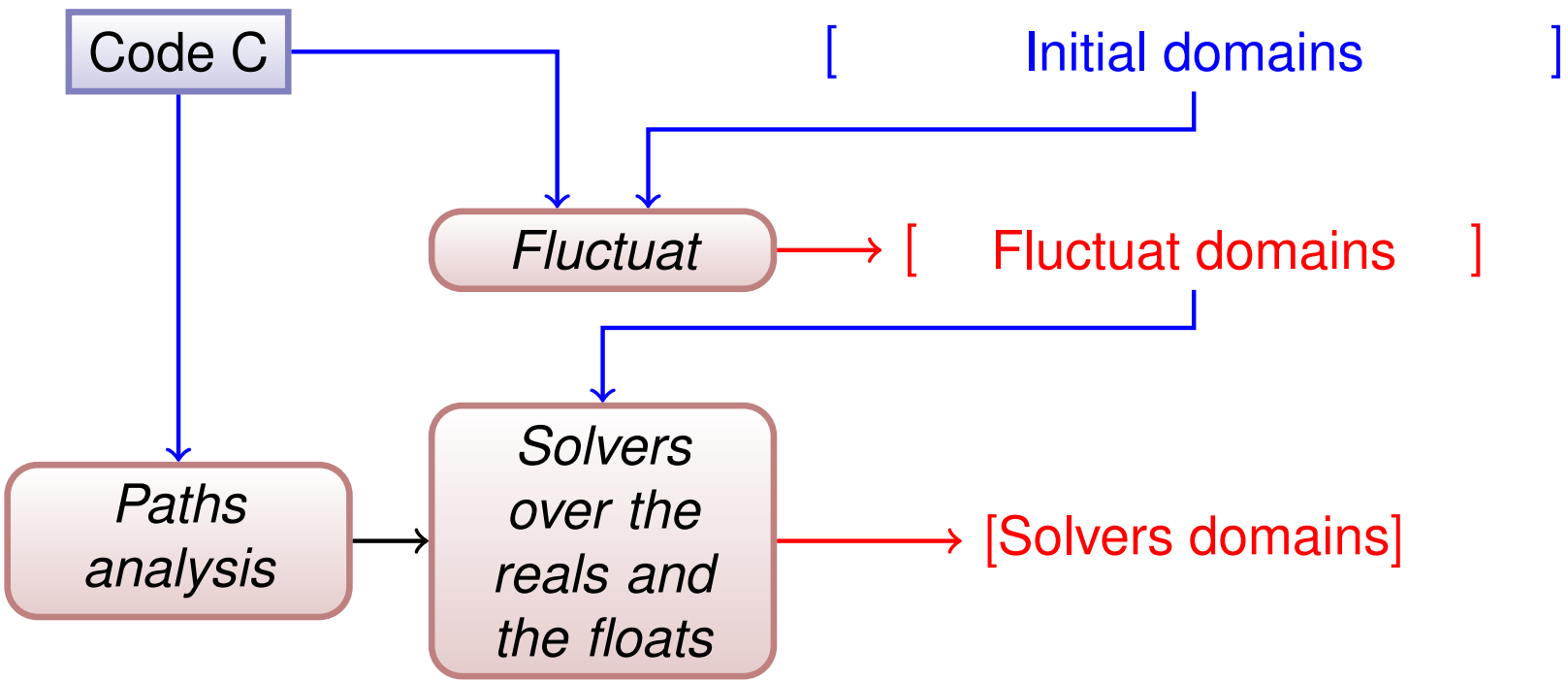
Proposed approach (1)

- **Goal:** *Refine the approximations* of the domains of the program variables computed by abstract interpretation
- **technique:** Use *local consistencies* to "shave" the domains
  $\rightarrow$ More accurate " semantics"
- *Complementary* to AI

Proposed approach (2)

Overview

We use constraint-based local consistencies to reduce the domains of variables computed by Fluctuat



Experiments, Programs

- **Quadratic** : Computes quadratic equation roots (GSL library) – *conditionals*
- **Sinus7** : Expression of the 7th-order Taylor series of function Sine – *nonlinearity*
- **Rump** : Polynomial of Rump – *nonlinearity*
- **Sqrt** : Square root computation (Babylonian method) – *iterative program*

  sqrt $\#_1$: $x \in [4.5\ ,\ 5.5]$

  sqrt $\#_2$: $x \in [5\ ,\ 10]$

Quadratic

quadratic #1: $a \in [-1\ ,\ 1]$, $b \in [0.5\ ,\ 1]$ and $c \in [0\ ,\ 2]$
quadratic #2: $a, b, c \in [1\ ,\ 1e6]$

```
int quadratic(double a, double b, double c) {
  double r, sgnb, temp, r1, r2;
  double disc = b * b - 4 * a * c;
  if (a == 0)
    if (b == 0) return 0; else { x0 = -c / b; return 1; }
  if (disc > 0) {
    if (b == 0) {
      r = fabs (0.5 * sqrt (disc) / a);
      x0 = -r; x1 = r;
    } else {
      sgnb = (b > 0 ? 1 : -1);
      temp = -0.5 * (b + sgnb * sqrt (disc));
      r1 = temp / a; r2 = c / temp;
      if (r1 < r2) { x0 = r1; x1 = r2; } else { x0 = r2; x1 = r1; }
    }
    return 2;
  } else if (disc == 0) {
    x0 = -0.5 * b / a; x1 = -0.5 * b / a; return 2;
  }
  else return 0; }
```

Sinus7, Rump, Sqrt

```
double sinus7(double x) { /* x ∈ [−1, 1] */
  return x - x*x*x/6 + x*x*x*x*x/120 + x*x*x*x*x*x*x/5040;
}

double rump(double x, double y) { /* x ∈ [7e4, 8e4]  y ∈ [3e4, 4e4] */
  double f;
  f = 333.75*y*y*y*y*y*y;
  f = f + x*x*(11*x*x*y*y - y*y*y*y*y*y
      - 121*y*y*y*y - 2);
  f = f + 5.5*y*y*y*y*y*y*y*y;
  f = f + x / (2*y);
  return f;
}

double sqrt(double x) {
  double xn, xn1;
  xn = x/2;
  xn1 = 0.5*(xn + x/xn);
  while (xn-xn1 > 1e-2) {
    xn = xn1;
    xn1 = 0.5*(xn + x/xn);
  }
  return xn1;
}
```

Results over the reals

	Fluctuat (AI)		RealPaver (CP)	
	Domain	Time	Domain	Time
quadratic $\#_1$				
$a \in [-1\ ,\ 1]$ x_0	$[-\infty, \infty]$	0.1 s	$[-\infty, 0]$	1.5 s
$b \in [0.5\ ,\ 1]$ x_1	$[-\infty, \infty]$	0.1 s	$[-8.011, \infty]$	1.5 s
$c \in [0\ ,\ 2]$				
quadratic $\#_2$				
$a \in [1\ ,\ 1e6]$ x_0	$[-2e6, 0]$	0.1 s	$[-1e6, 0]$	0.5 s
$b \in [1\ ,\ 1e6]$ x_1	$[-1e6, 0]$	0.1 s	$[-5.186e5, 0]$	0.5 s
$c \in [1\ ,\ 1e6]$				
rump				
$x \in [7e4\ ,\ 8e4]$	$[-1e37, 2e37]$	0.1 s	$[-1e36, 1.7e37]$	1.2 s
$y \in [3e4\ ,\ 4e4]$				
sinus7 $x \in [-1\ ,\ 1]$	$[-1.009, 1.009]$	0.1 s	$[-0.842, 0.843]$	0.3 s
sqrt $\#_1$ $x \in [4.5\ ,\ 5.5]$	$[2.116, 2.354]$	0.1 s	$[2.121, 2.346]$	0.3 s
sqrt $\#_2$ $x \in [5\ ,\ 10]$	$[2.098, 3.435]$	0.1 s	$[2.232, 3.165]$	0.5 s

FPCS

Correct solver over the floats based on 2B-consistency: no solution lost

- *Projection functions* for floats:
  - Direct projection: straightforward adaptation of interval arithmetic
  - Inverse projection: uses a larger format than the system variables
- Handling of *rounding modes*, *nonlinear expressions* and the usual *mathematical functions* (trigonometric...)
- Handling of x86 architecture specifics

Results over the floats

	Fluctuat (AI)		FPCS (CP)	
	Domain	Time	Domain	Time
quadratic $\#_1$				
$a \in [-1\ ,\ 1]$ x_0	$[-\infty, \infty]$	0.1 s	$[-\infty, 0]$	0.3 s
$b \in [0.5\ ,\ 1]$ x_1	$[-\infty, \infty]$	0.1 s	$[-8.064, \infty]$	0.3 s
$c \in [0\ ,\ 2]$				
quadratic $\#_2$				
$a \in [1\ ,\ 1e6]$ x_0	$[-2e6, 0]$	0.1 s	$[-2e6, 0]$	0.3 s
$b \in [1\ ,\ 1e6]$ x_1	$[-1e6, 0]$	0.1 s	$[-2503.8, 0]$	0.3 s
$c \in [1\ ,\ 1e6]$				
rump				
$x \in [7e4\ ,\ 8e4]$	$[-1e37, 2e37]$	0.1 s	$[-1e37, 2e37]$	0.2 s
$y \in [3e4\ ,\ 4e4]$				
sinus7 $x \in [-1\ ,\ 1]$	$[-1.009, 1.009]$	0.1 s	$[-0.853, 0.852]$	0.2 s
sqrt $\#_1$ $x \in [4.5\ ,\ 5.5]$	$[2.116, 2.354]$	0.1 s	$[2.120, 2.347]$	1 s
sqrt $\#_2$ $x \in [5\ ,\ 10]$	$[-\infty, \infty]$	0.1 s	$[2.232, 3.168]$	1.6 s

Conclusion

- CP
  - Good *refutation* capabilities
  - Distinct exploration of each executable path is a *critical issue*
- AI
  - Good *scaling* capabilities
  - *Over-approximations* can be very rough

$\rightarrow$ *Complementary* techniques $\rightarrow$ *hybrid* approach

- Greater domain reduction achieved when combining both approaches
- On going work: *tight cooperation*
  $\rightarrow$ refining domains at each jonction point