Refining AI-based Approximations with hybrid CP solvers

Michel RUEHER

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

(Courtesy to Olivier Ponsini, Claude Michel)

June, 2011

Turunç

Outline

Problem: Programs with floating–point computations
- Context
- Problems with floating-point numbers
- Objective & Approach
- Example

Abstract interpretation, Fluctuat
- Static analyzer
- Zonotopes

Improving approximations with CP
- Approach
- Details

Experiments
- Programs
- Results over the reals
- Results over floating point numbers

Conclusion

Problem: Programs with floating-point computations

- **Embedded Systems** (transportation, nuclear energy, medicine, avionics...) rely more and more on floating-point computations
- **C language** is most used for such applications
- The implementation of floating-point arithmetic generally conforms to **IEEE 754 standard**

Problems with floating-point numbers

- **Floats** $\rightarrow$ **additional possible source of errors**
- **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 real numbers versus semantics of floating-point numbers

Programs are run on the floats but:

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

Abstract Interpretation
$\rightarrow$ **Approximations** over **floats** and over the **real numbers**

Objective & Approach

- **Goal: Refine the approximations** computed by abstract interpretation for domains of the program variables

- **Method:** Use **local consistencies** to **“shave”** the domains

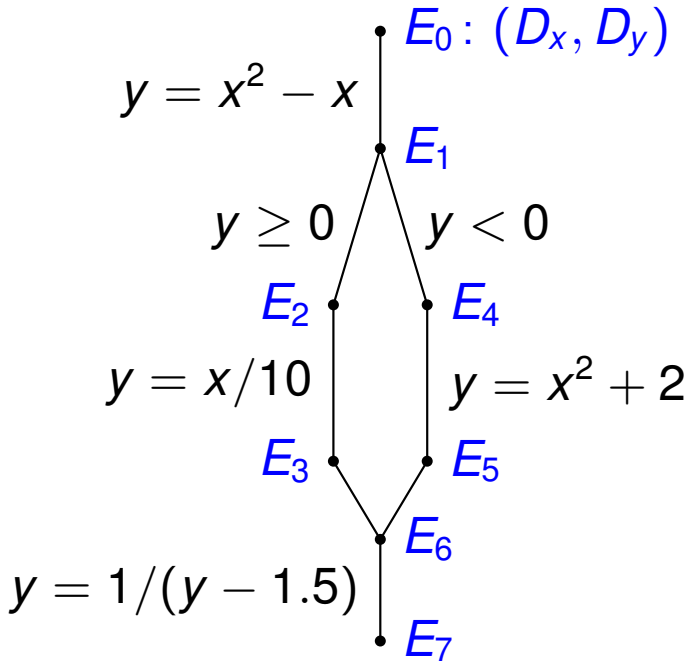
Abstract Interpretation, example

Abstract Interpretation requires:

1. **Semantics** to calculate the states of the program at various checkpoints
2. An **abstraction** to represent the states of the program
3. **Computation of a fixed point** of the equations of the semantics

Example (abstract domain of intervals)

```
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);
```



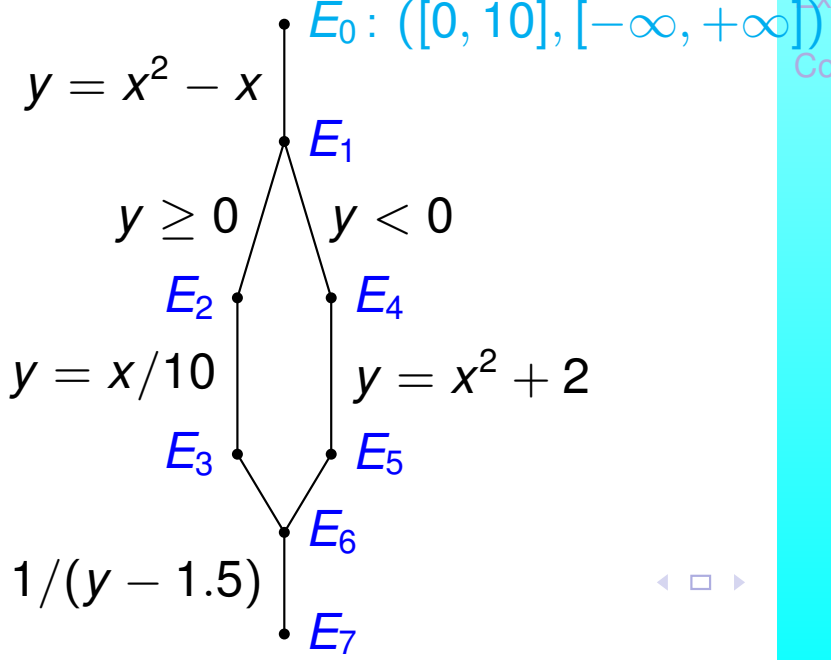
Abstract Interpretation, example

Abstract Interpretation requires:

1. **Semantics** to calculate the states of the program at various checkpoints
2. An **abstraction** to represent the states of the program
3. **Computation of a fixed point** of the equations of the semantics

Example (abstract domain of intervals)

```
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);
```



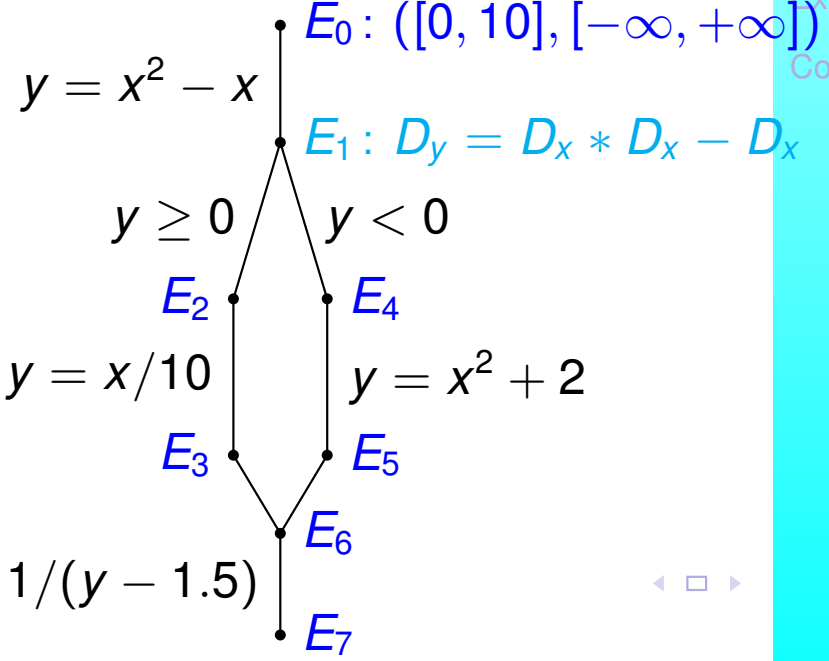
Abstract Interpretation, example

Abstract Interpretation requires:

1. **Semantics** to calculate the states of the program at various checkpoints
2. An **abstraction** to represent the states of the program
3. **Computation of a fixed point** of the equations of the semantics

Example (abstract domain of intervals)

```
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);
```



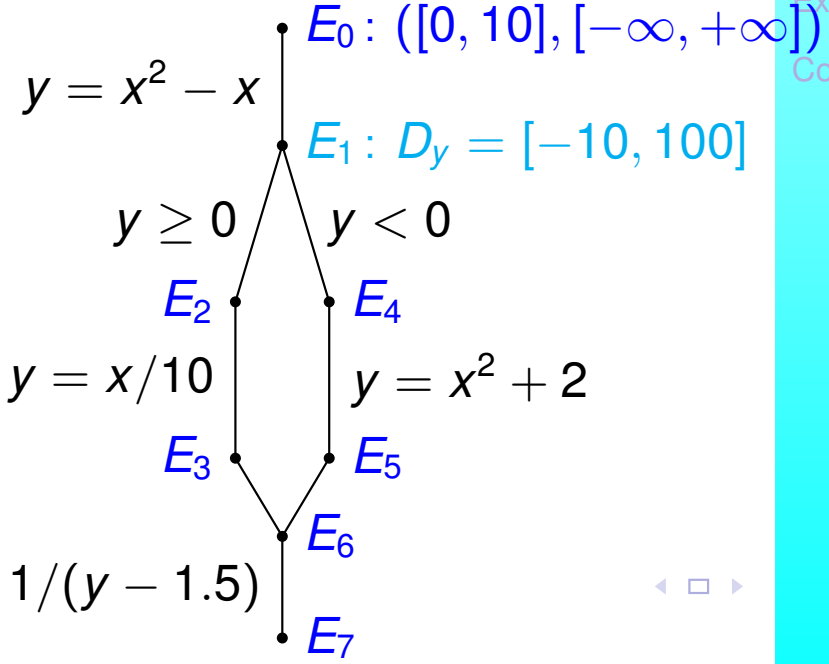
Abstract Interpretation, example

Abstract Interpretation requires:

1. **Semantics** to calculate the states of the program at various checkpoints
2. An **abstraction** to represent the states of the program
3. **Computation of a fixed point** of the equations of the semantics

Example (abstract domain of intervals)

```
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);
```



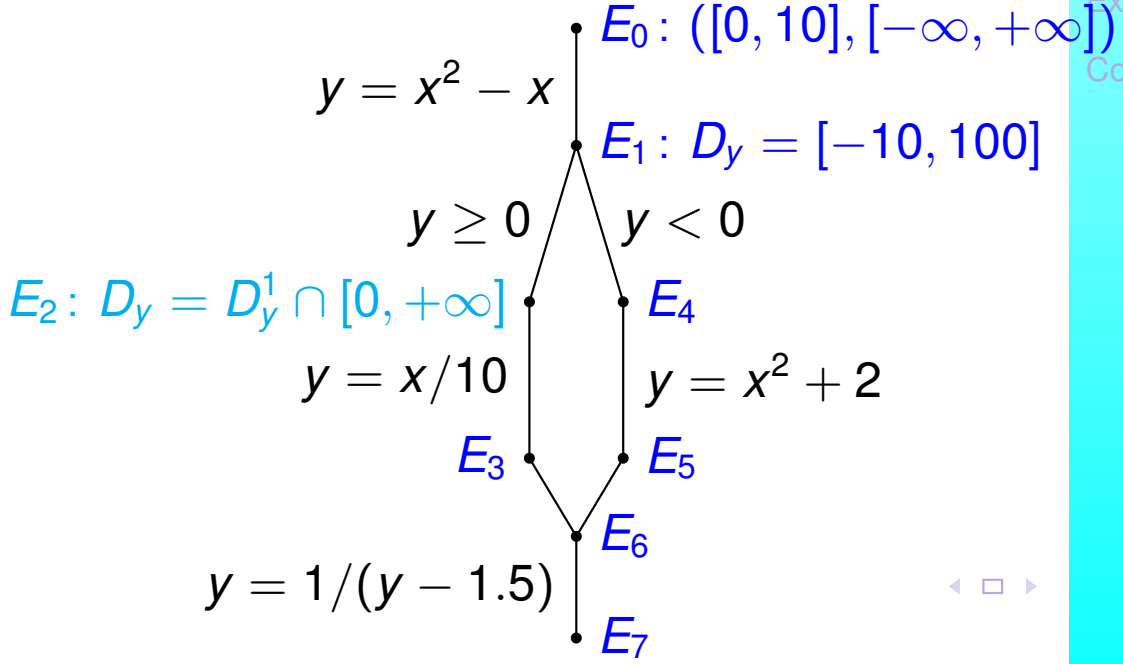
Abstract Interpretation, example

Abstract Interpretation requires:

1. **Semantics** to calculate the states of the program at various checkpoints
2. An **abstraction** to represent the states of the program
3. **Computation of a fixed point** of the equations of the semantics

Example (abstract domain of intervals)

```
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);
```



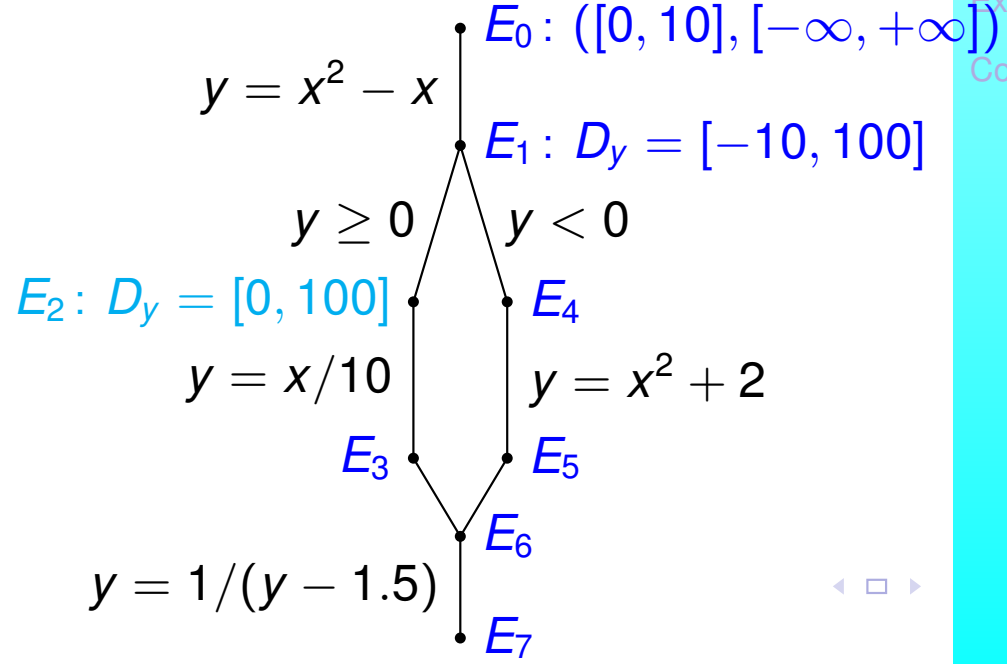
Abstract Interpretation, example

Abstract Interpretation requires:

1. **Semantics** to calculate the states of the program at various checkpoints
2. An **abstraction** to represent the states of the program
3. **Computation of a fixed point** of the equations of the semantics

Example (abstract domain of intervals)

```
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);
```



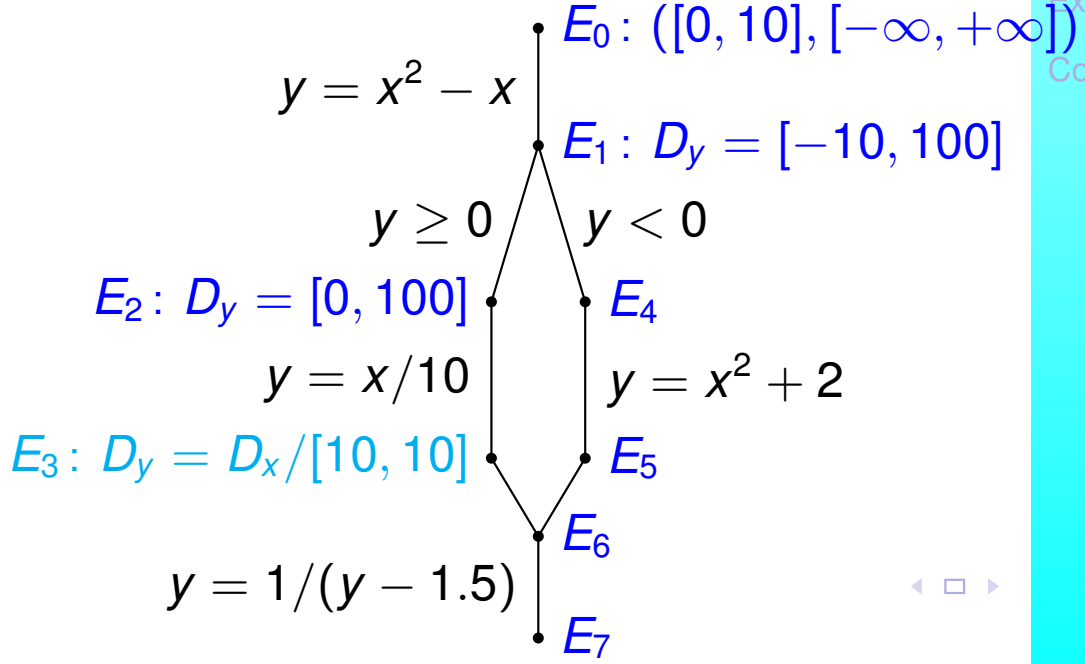
Abstract Interpretation, example

Abstract Interpretation requires:

1. **Semantics** to calculate the states of the program at various checkpoints
2. An **abstraction** to represent the states of the program
3. **Computation of a fixed point** of the equations of the semantics

Example (abstract domain of intervals)

```
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);
```



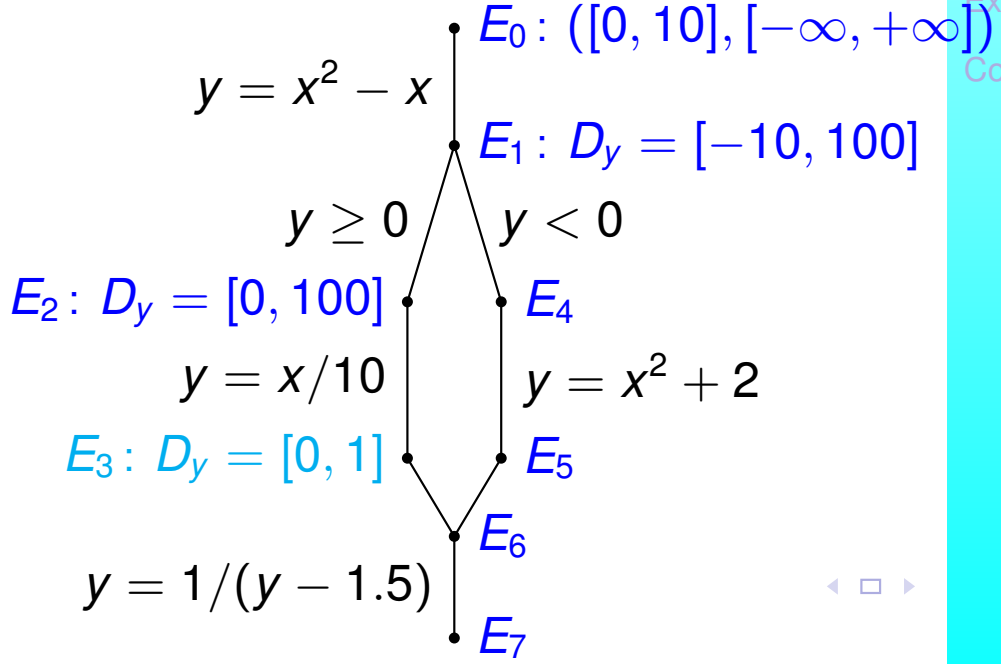
Abstract Interpretation, example

Abstract Interpretation requires:

1. **Semantics** to calculate the states of the program at various checkpoints
2. An **abstraction** to represent the states of the program
3. **Computation of a fixed point** of the equations of the semantics

Example (abstract domain of intervals)

```
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);
```



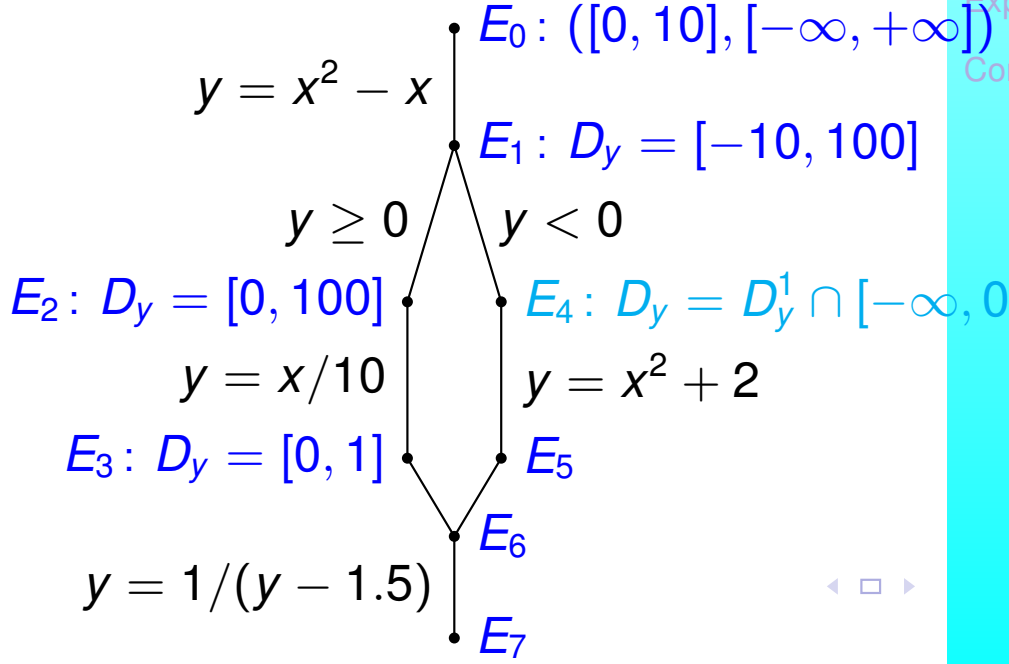
Abstract Interpretation, example

Abstract Interpretation requires:

1. **Semantics** to calculate the states of the program at various checkpoints
2. An **abstraction** to represent the states of the program
3. **Computation of a fixed point** of the equations of the semantics

Example (abstract domain of intervals)

```
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);
```



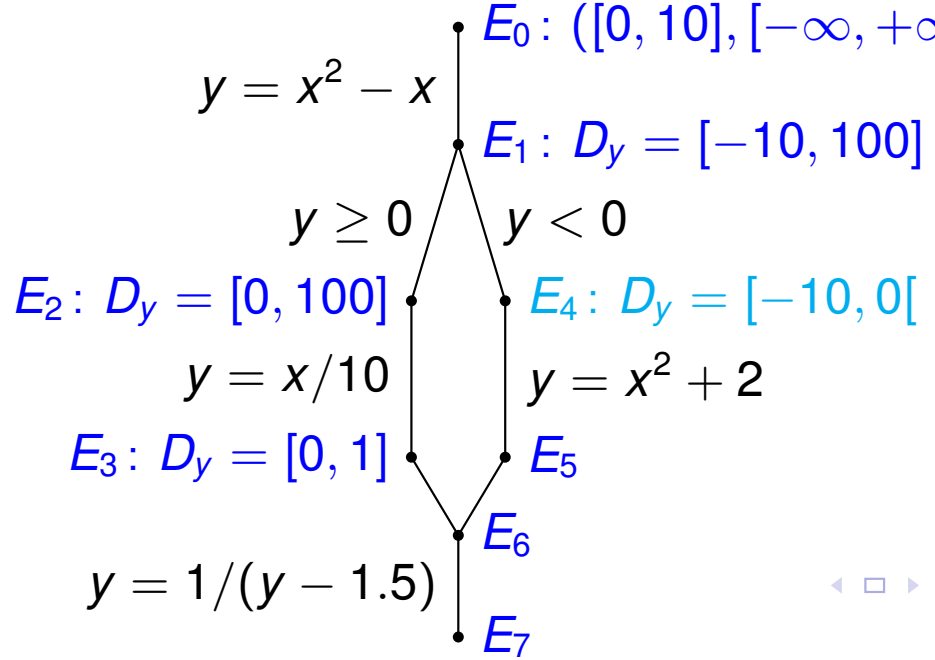
Abstract Interpretation, example

Abstract Interpretation requires:

1. **Semantics** to calculate the states of the program at various checkpoints
2. An **abstraction** to represent the states of the program
3. **Computation of a fixed point** of the equations of the semantics

Example (abstract domain of intervals)

```
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);
```



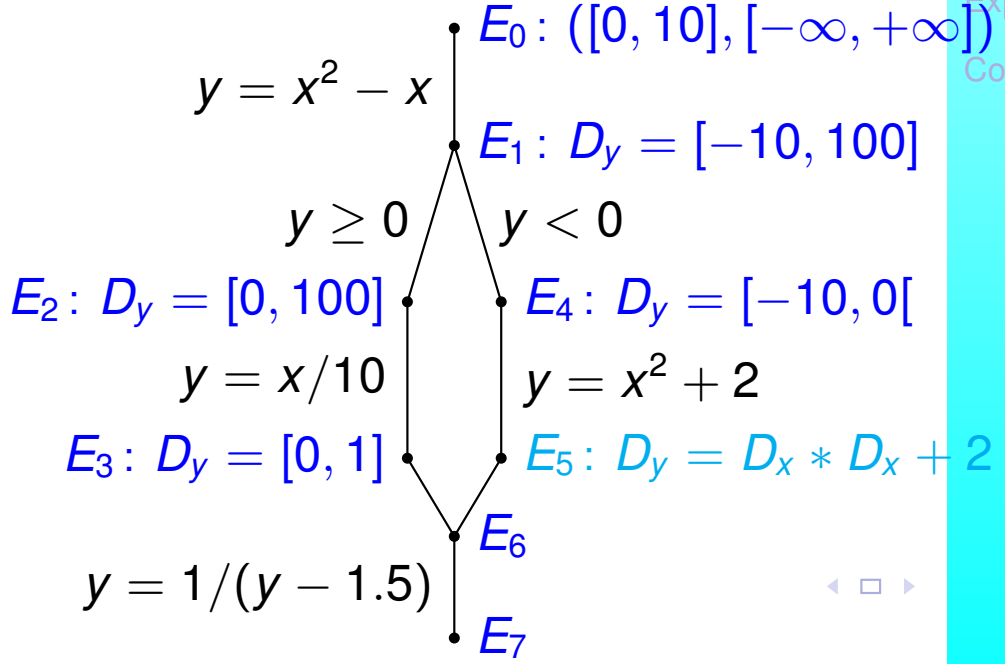
Abstract Interpretation, example

Abstract Interpretation requires:

1. **Semantics** to calculate the states of the program at various checkpoints
2. An **abstraction** to represent the states of the program
3. **Computation of a fixed point** of the equations of the semantics

Example (abstract domain of intervals)

```
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);
```



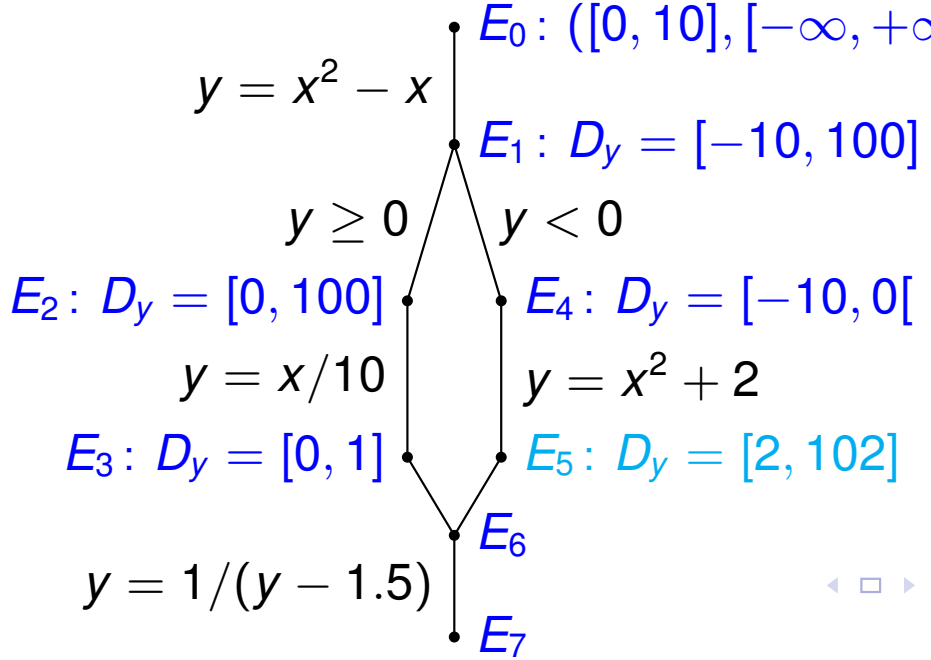
Abstract Interpretation, example

Abstract Interpretation requires:

1. **Semantics** to calculate the states of the program at various checkpoints
2. An **abstraction** to represent the states of the program
3. **Computation of a fixed point** of the equations of the semantics

Example (abstract domain of intervals)

```
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);
```



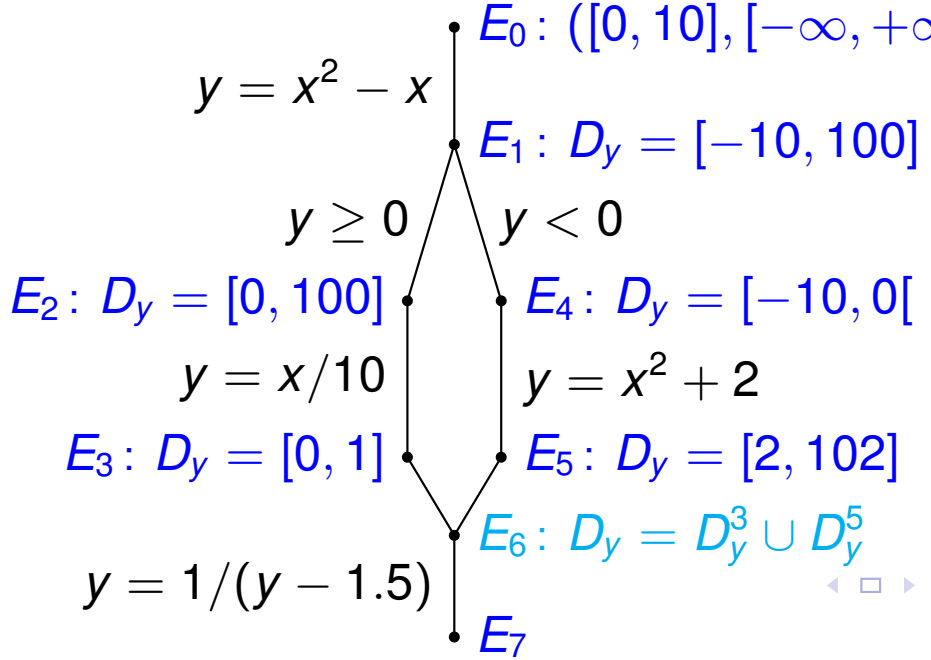
Abstract Interpretation, example

Abstract Interpretation requires:

1. **Semantics** to calculate the states of the program at various checkpoints
2. An **abstraction** to represent the states of the program
3. **Computation of a fixed point** of the equations of the semantics

Example (abstract domain of intervals)

```
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);
```



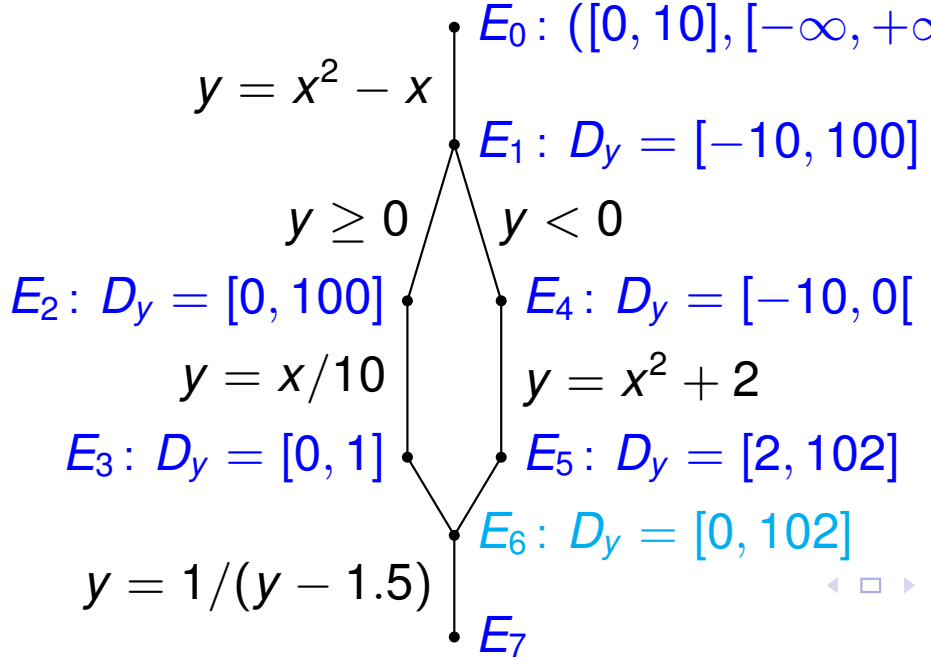
Abstract Interpretation, example

Abstract Interpretation requires:

1. **Semantics** to calculate the states of the program at various checkpoints
2. An **abstraction** to represent the states of the program
3. **Computation of a fixed point** of the equations of the semantics

Example (abstract domain of intervals)

```
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);
```



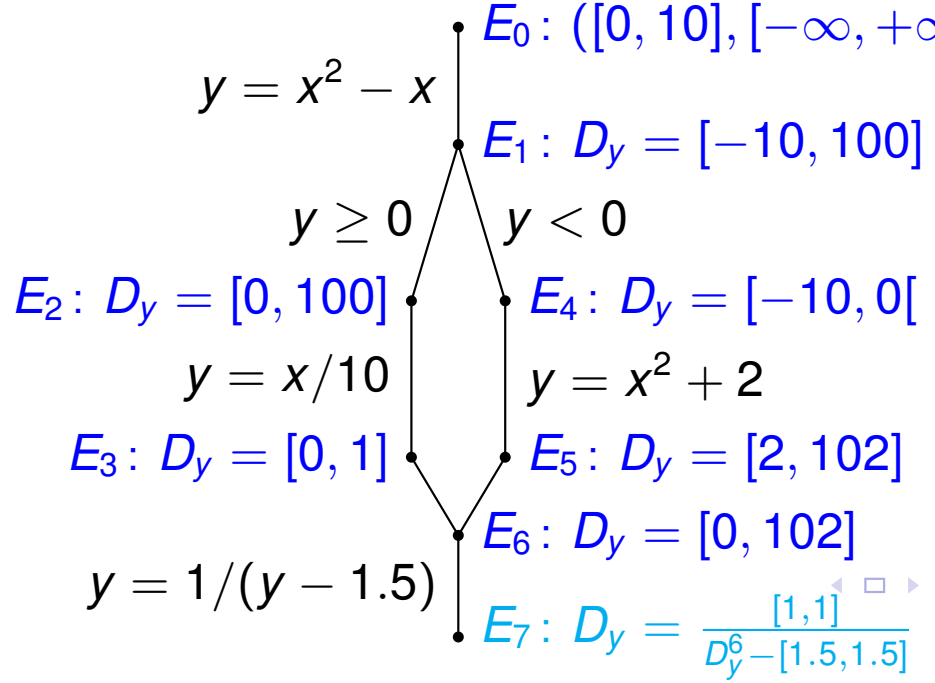
Abstract Interpretation, example

Abstract Interpretation requires:

1. **Semantics** to calculate the states of the program at various checkpoints
2. An **abstraction** to represent the states of the program
3. **Computation of a fixed point** of the equations of the semantics

Example (abstract domain of intervals)

```
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);
```



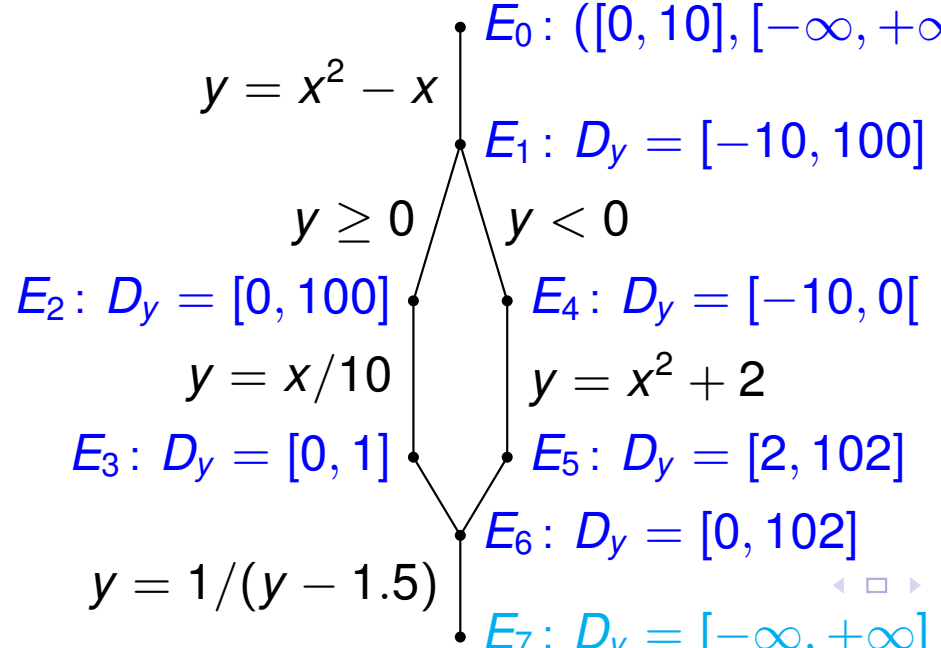
Abstract Interpretation, example

Abstract Interpretation requires:

1. **Semantics** to calculate the states of the program at various checkpoints
2. An **abstraction** to represent the states of the program
3. **Computation of a fixed point** of the equations of the semantics

Example (abstract domain of intervals)

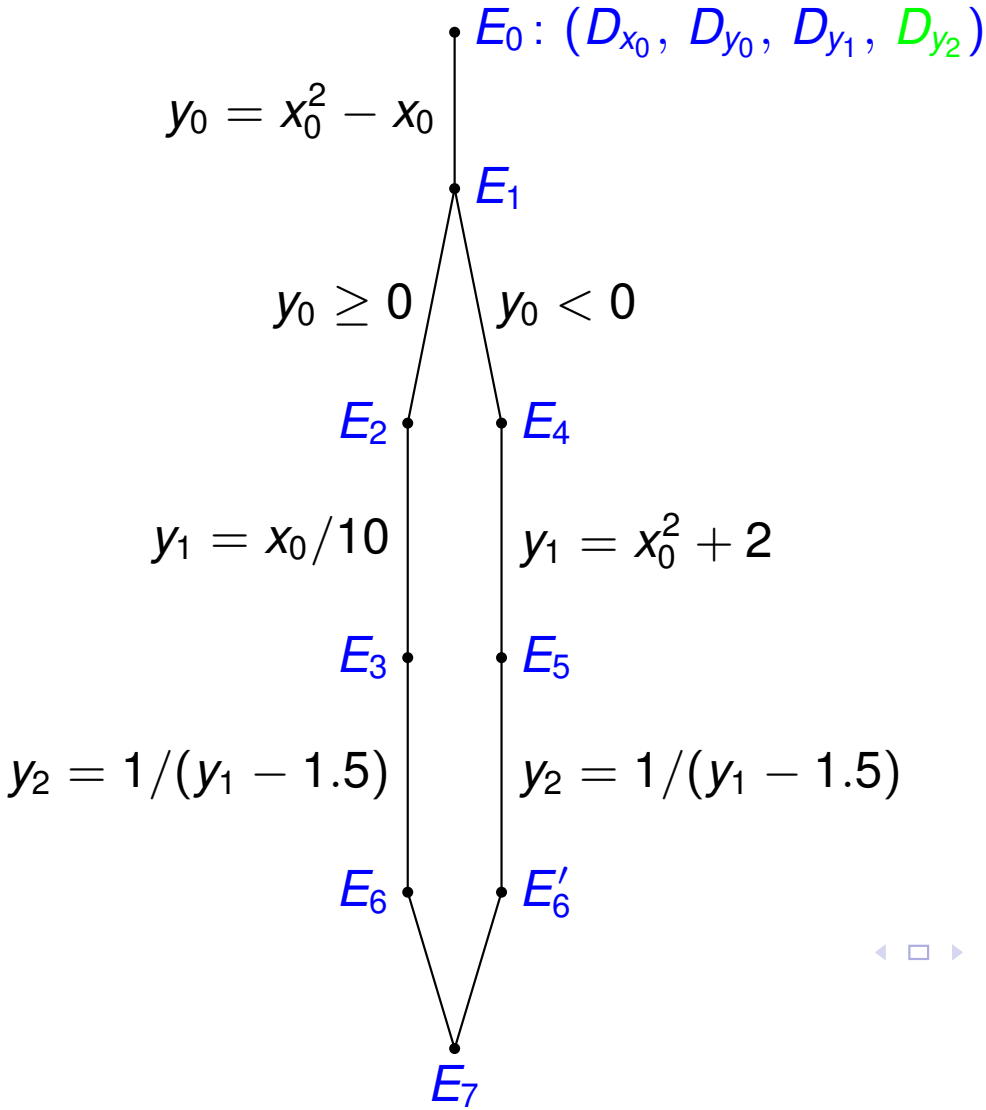
```
float x = [0,10];
float y = x*x - x;
if (y >= 0)
  y = x/10;
else
  y = x*x + 2;
y = 1 / (y-1.5);
```



Refining AI-based approximations, example

Using CP-based local consistencies

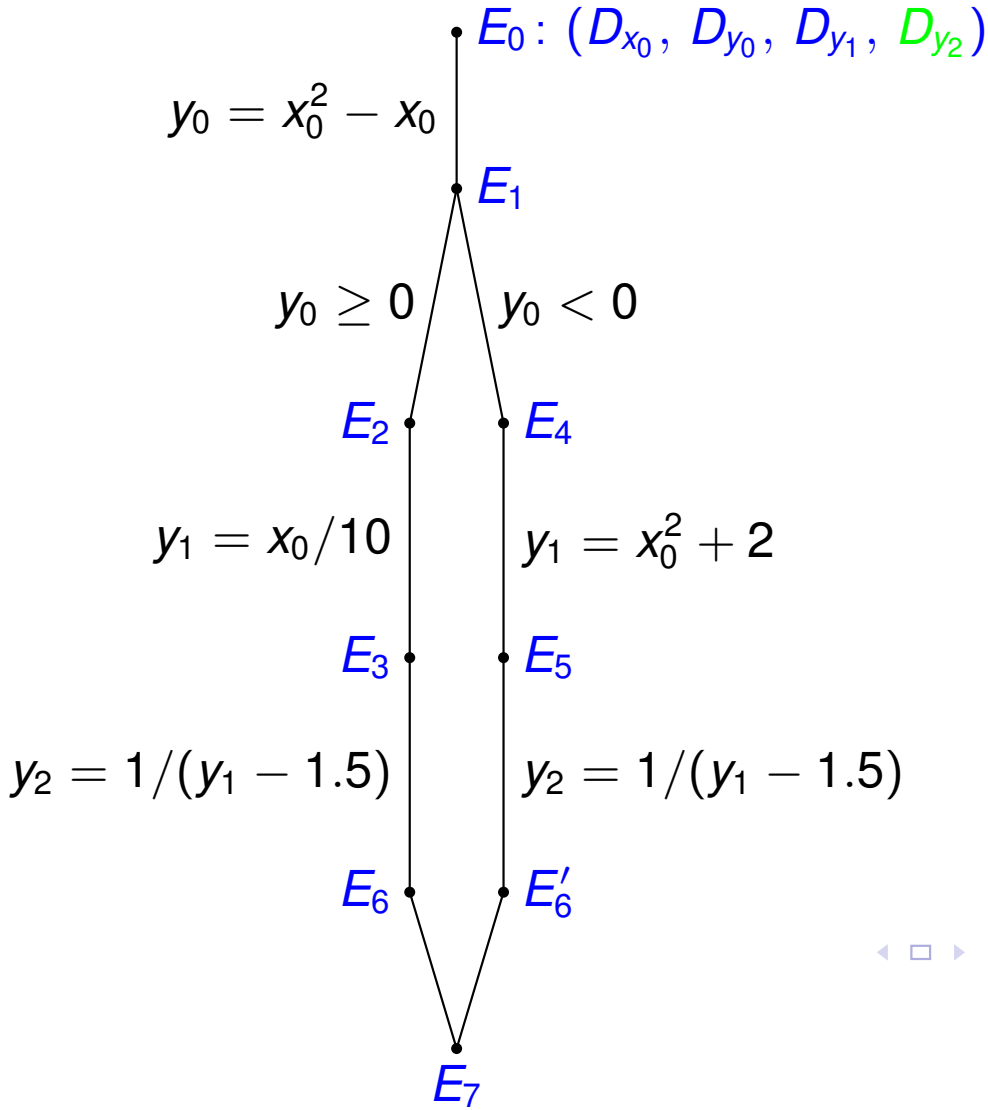
```
float x  = [0,10];
float y  = x *x  - x ;
if (y  >= 0)
  y  = x /10;
else
  y  = x *x  + 2;
 y  = 1 / (y -1.5);
```



Refining AI-based approximations, example

Using CP-based local consistencies

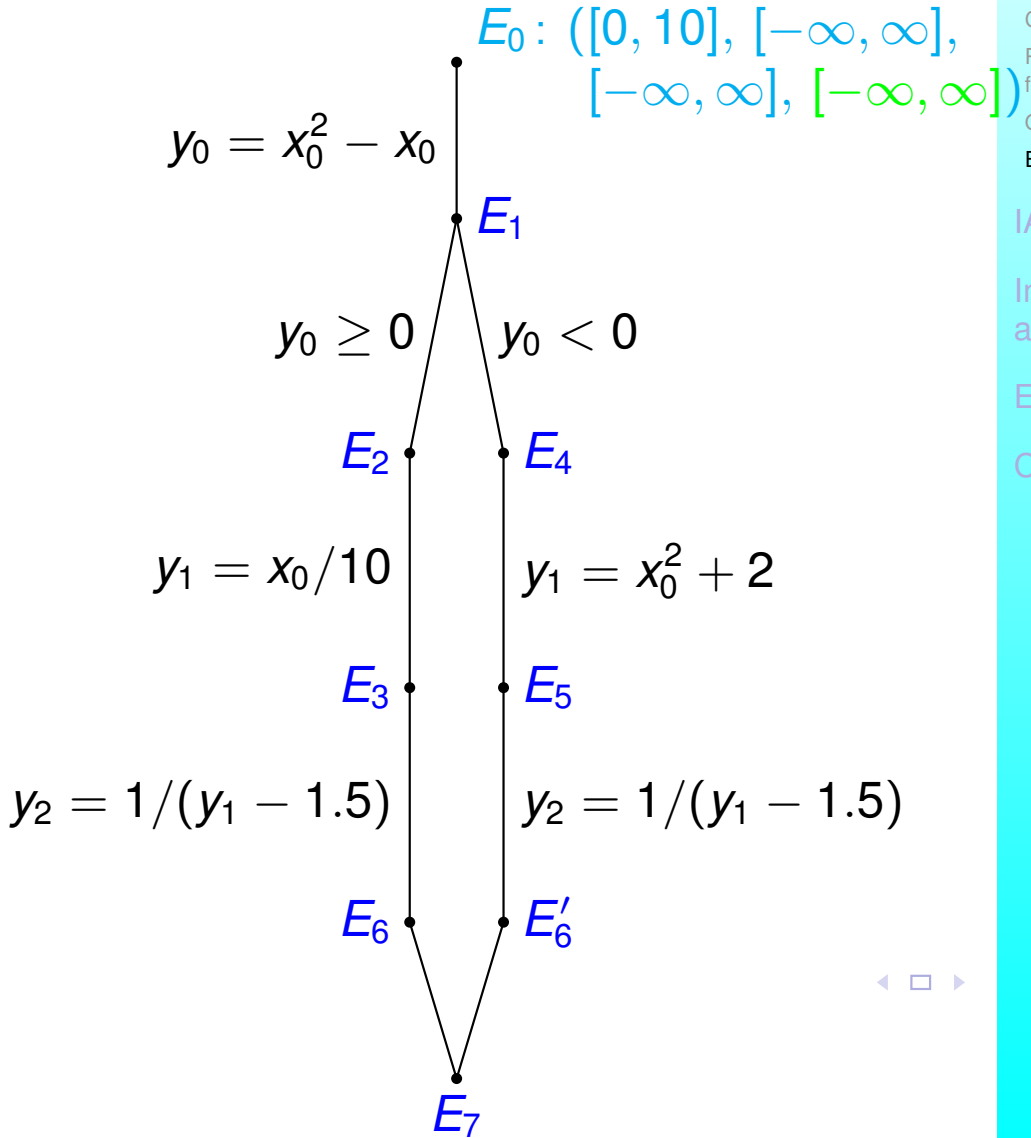
```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



Refining AI-based approximations, example

Using CP-based local consistencies

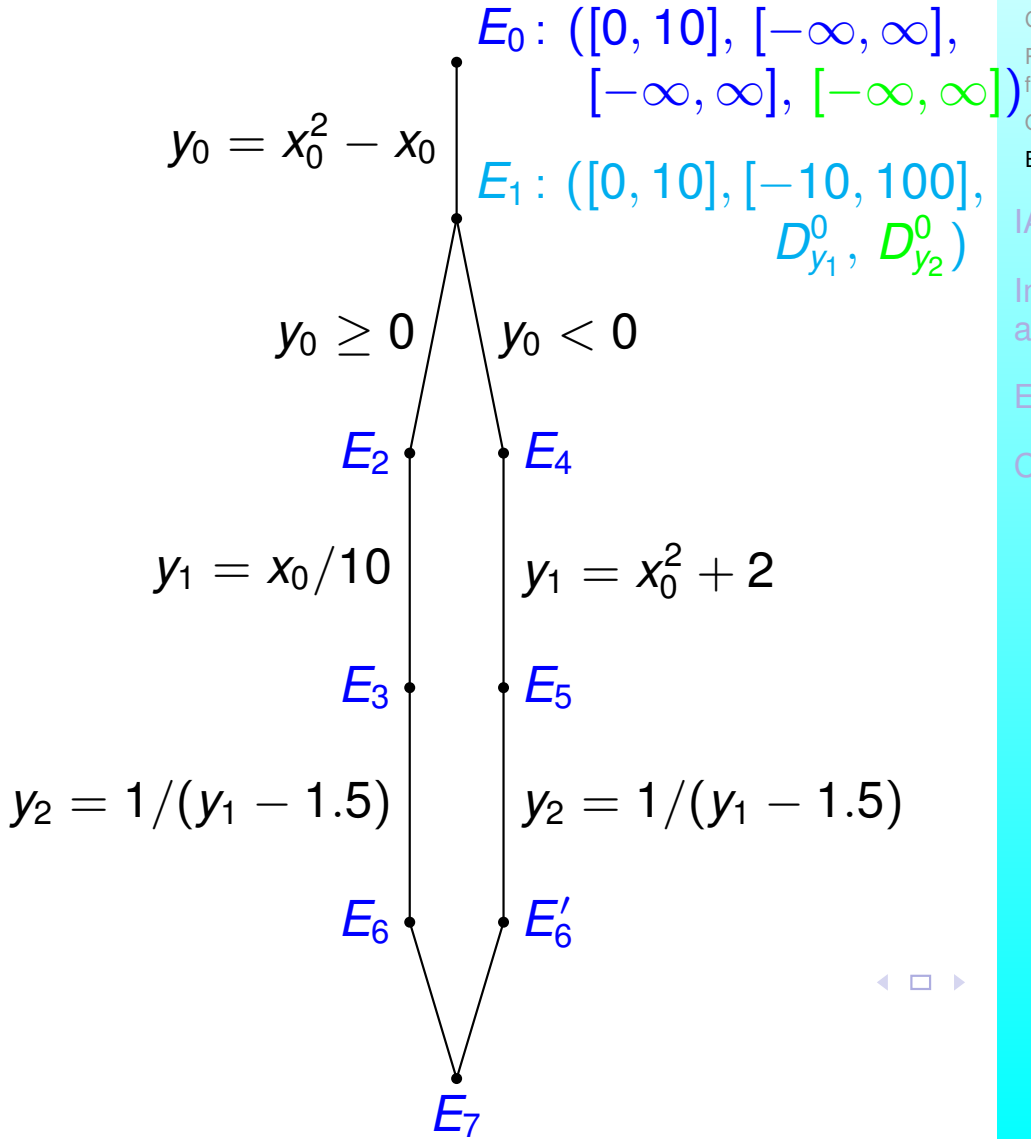
```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



Refining AI-based approximations, example

Using CP-based local consistencies

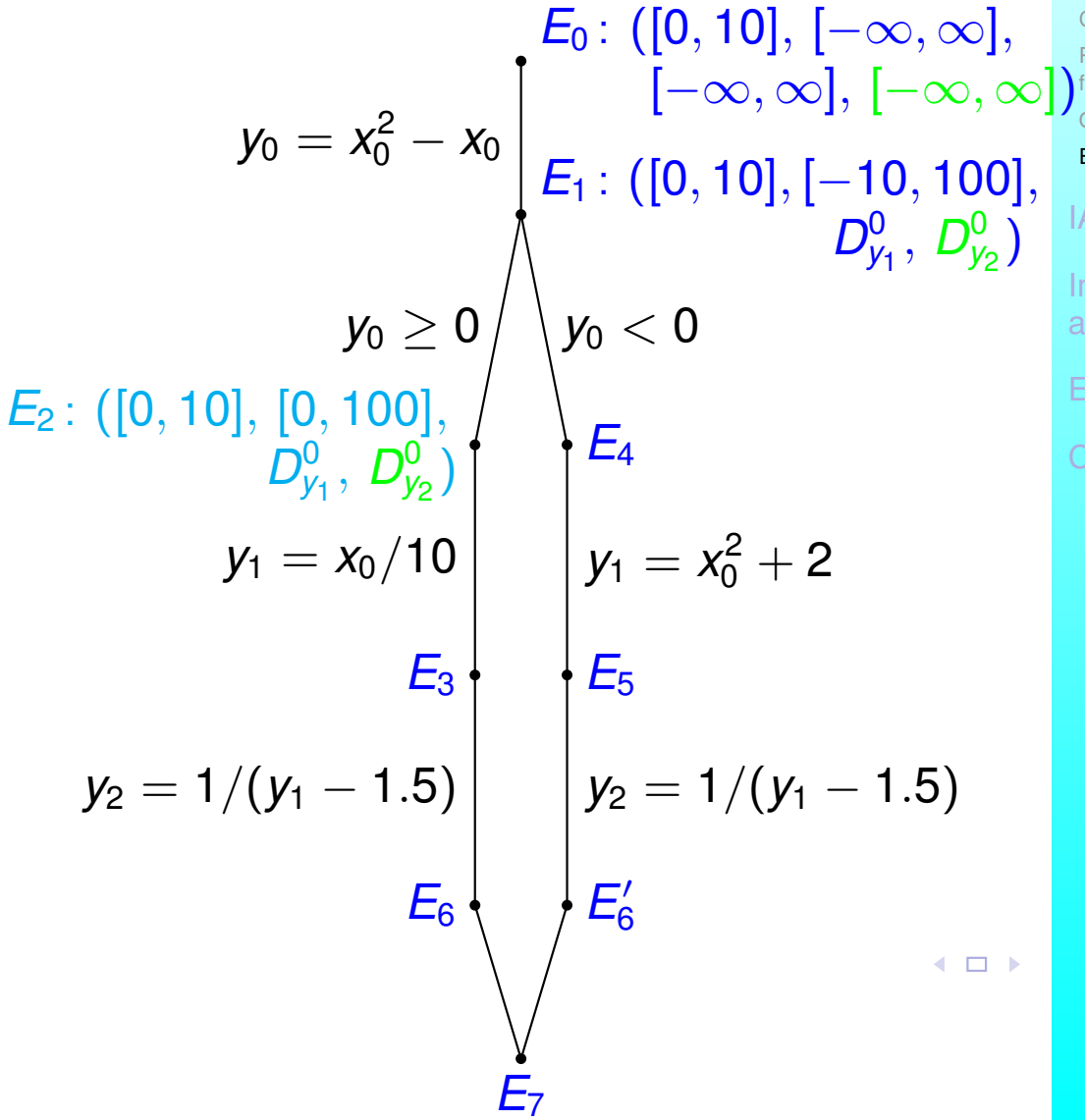
```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



Refining AI-based approximations, example

Using CP-based local consistencies

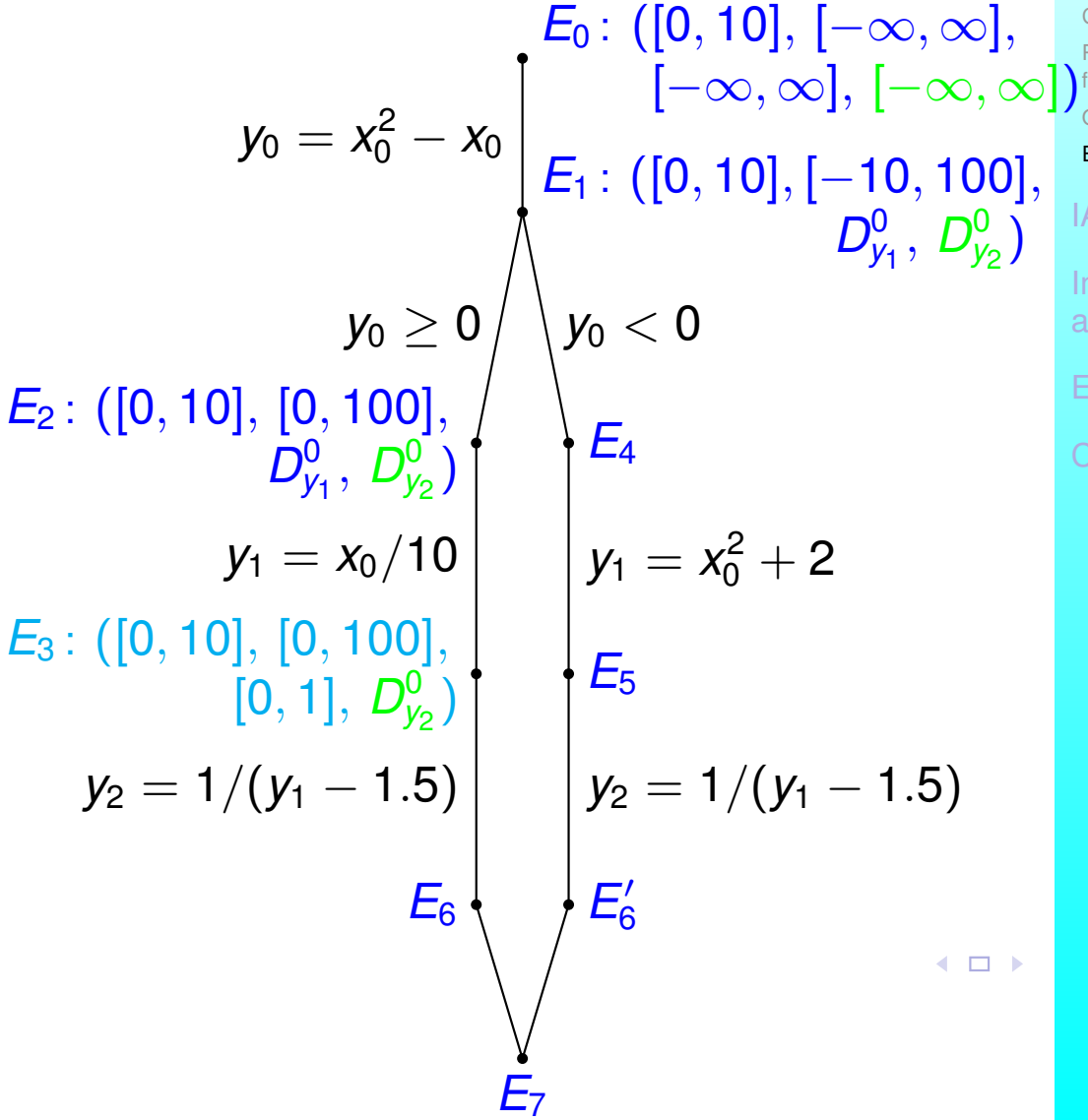
```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



Refining AI-based approximations, example

Using CP-based local consistencies

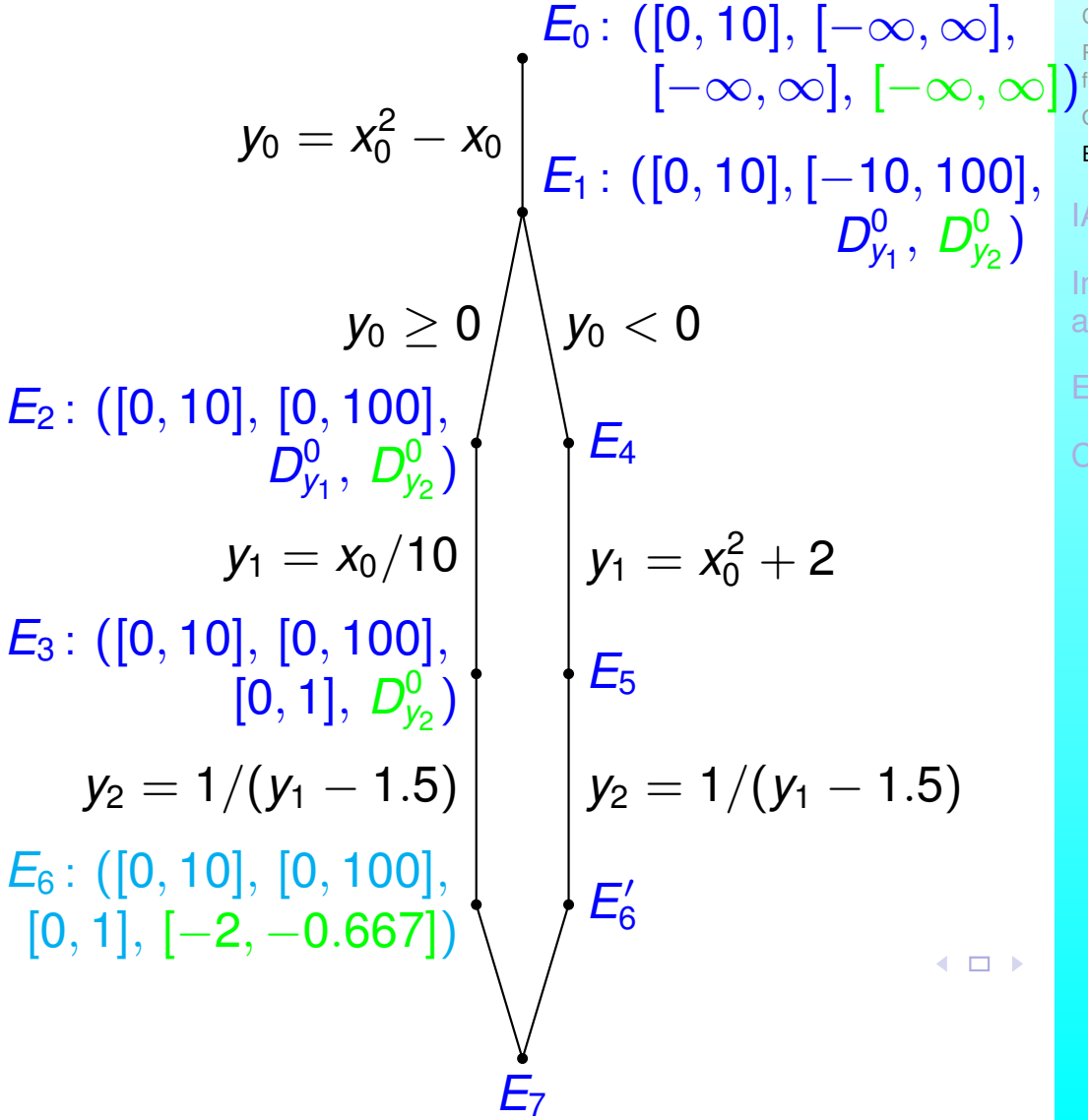
```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



Refining AI-based approximations, example

Using CP-based local consistencies

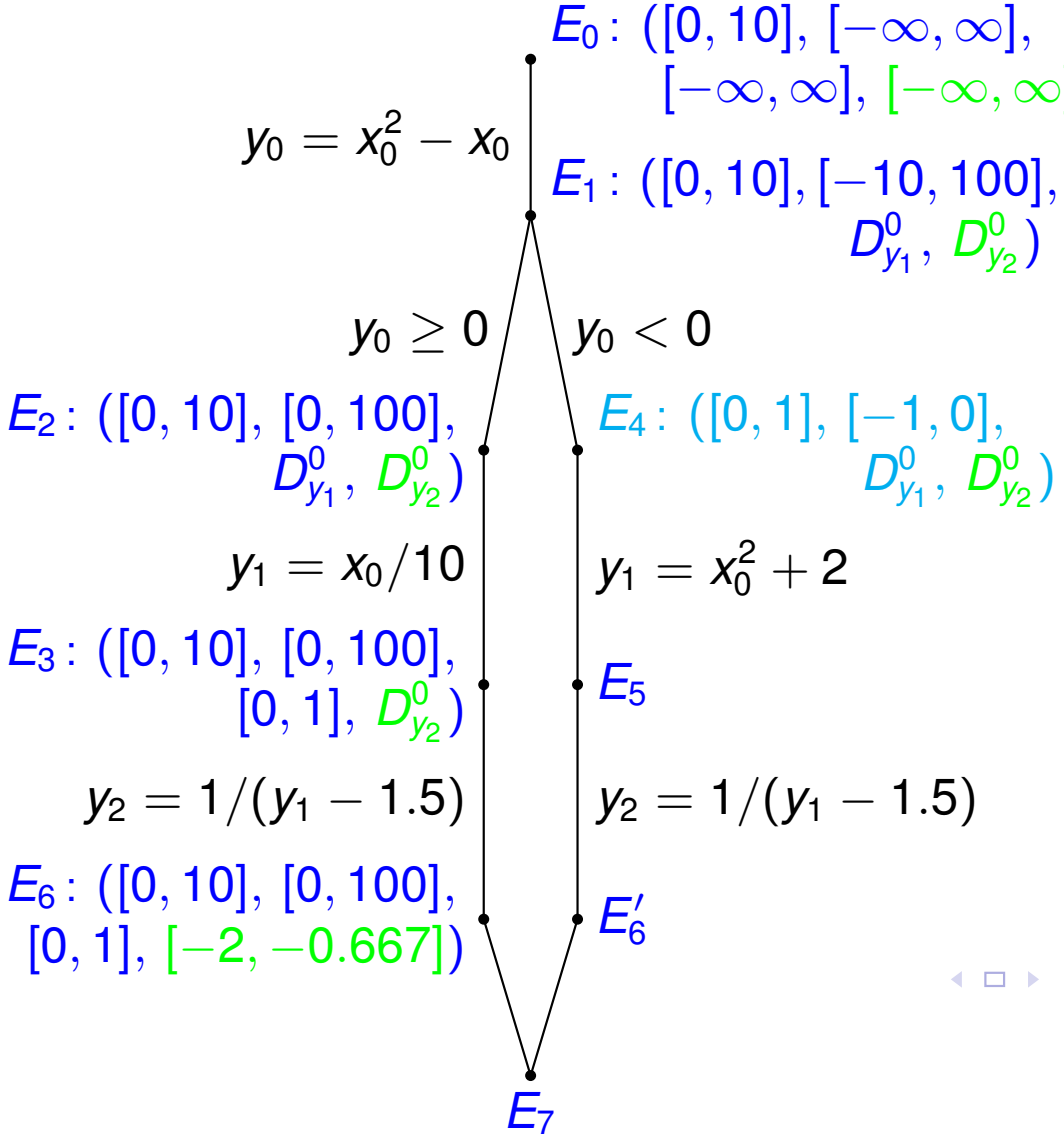
```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



Refining AI-based approximations, example

Using CP-based local consistencies

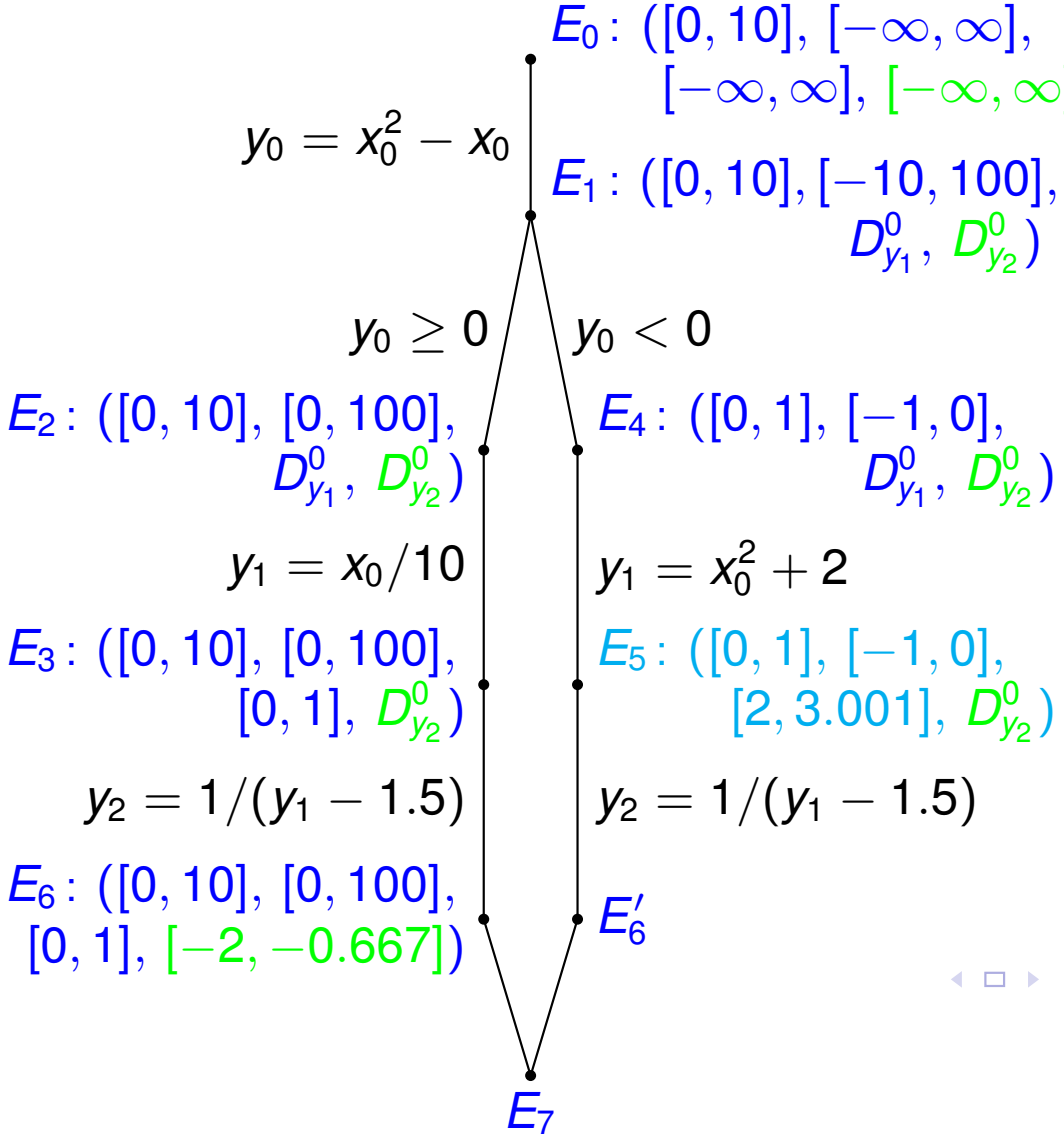
```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



Refining AI-based approximations, example

Using CP-based local consistencies

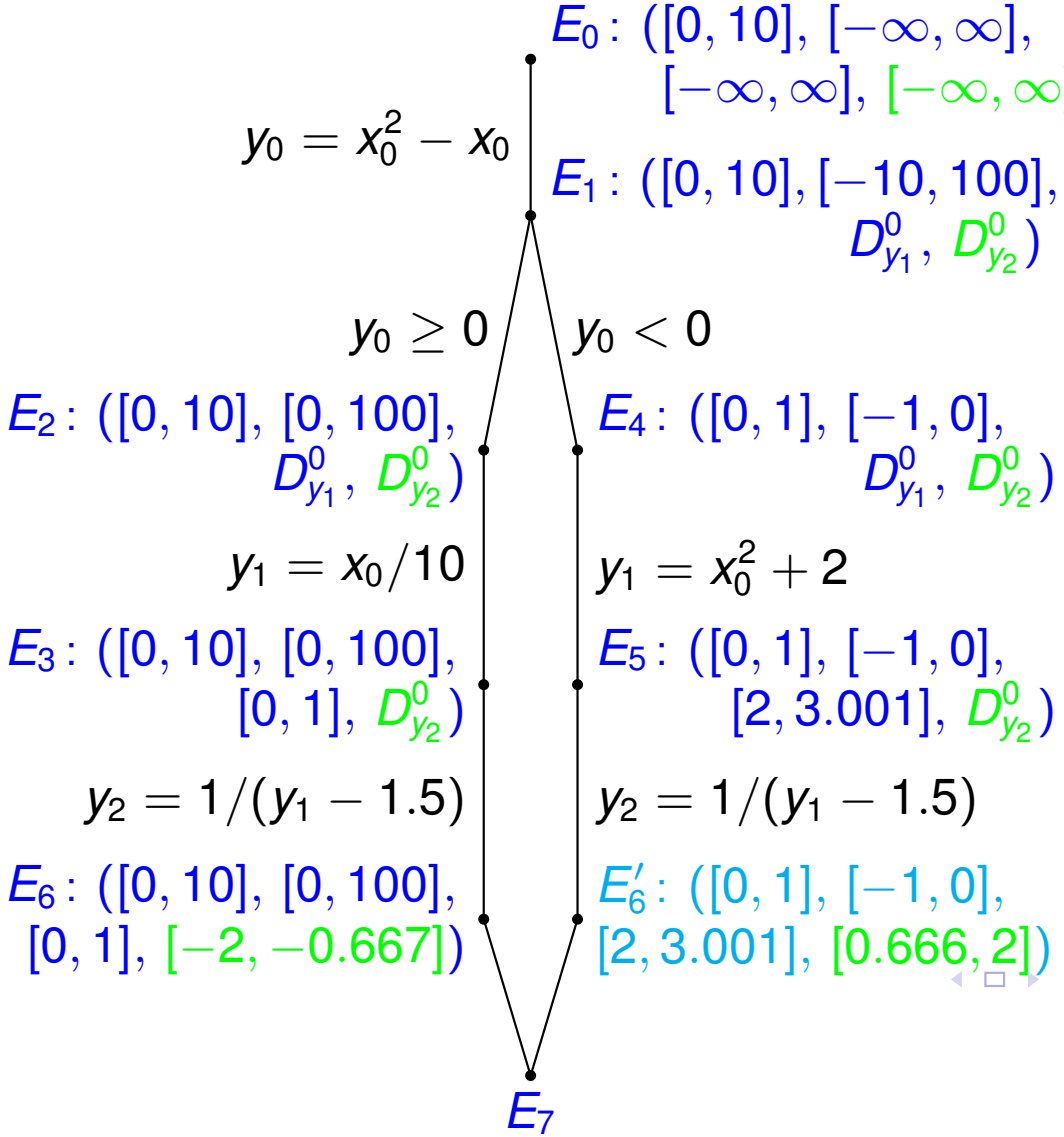
```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



Refining AI-based approximations, example

Using CP-based local consistencies

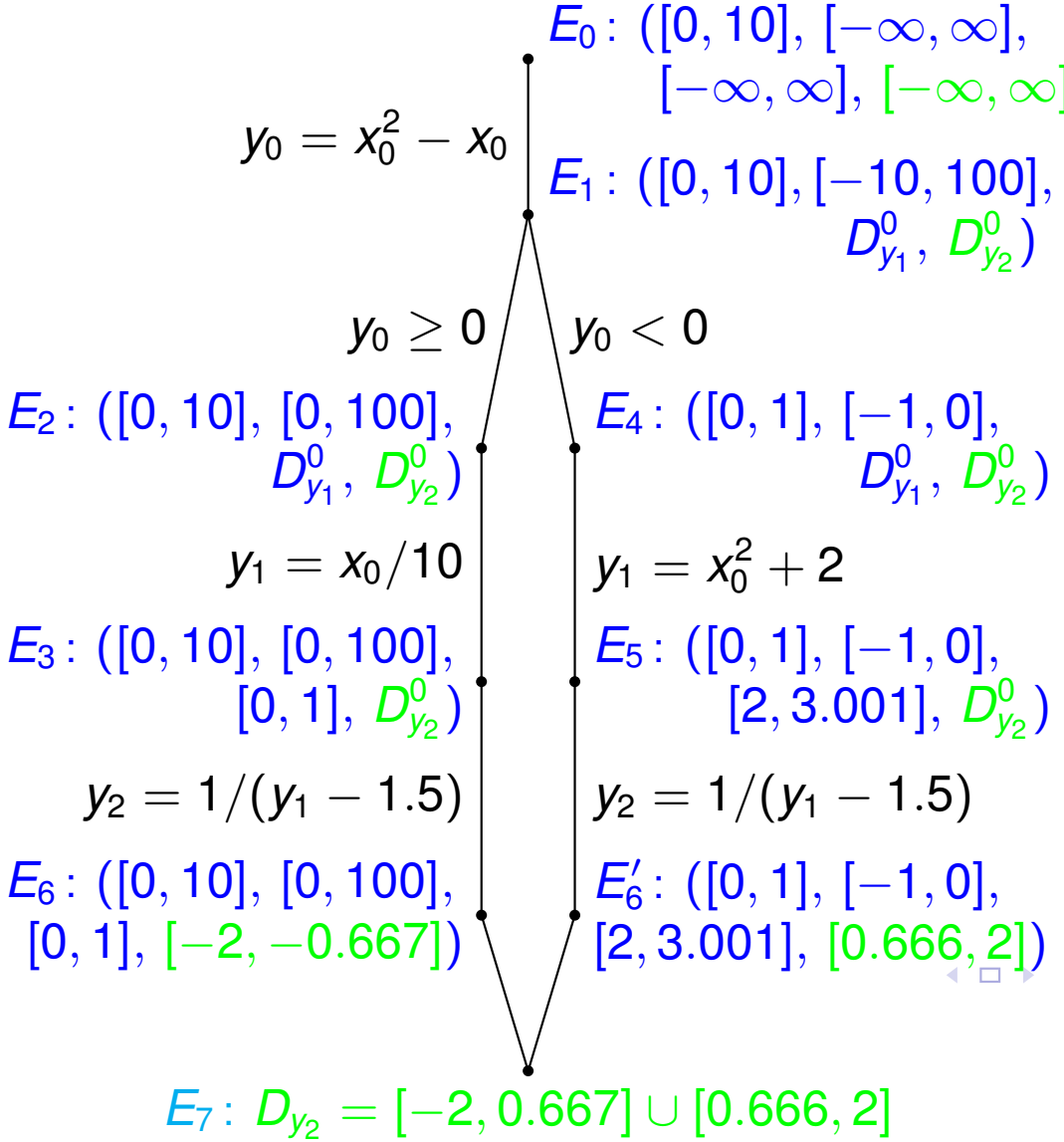
```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



Refining AI-based approximations, example

Using CP-based local consistencies

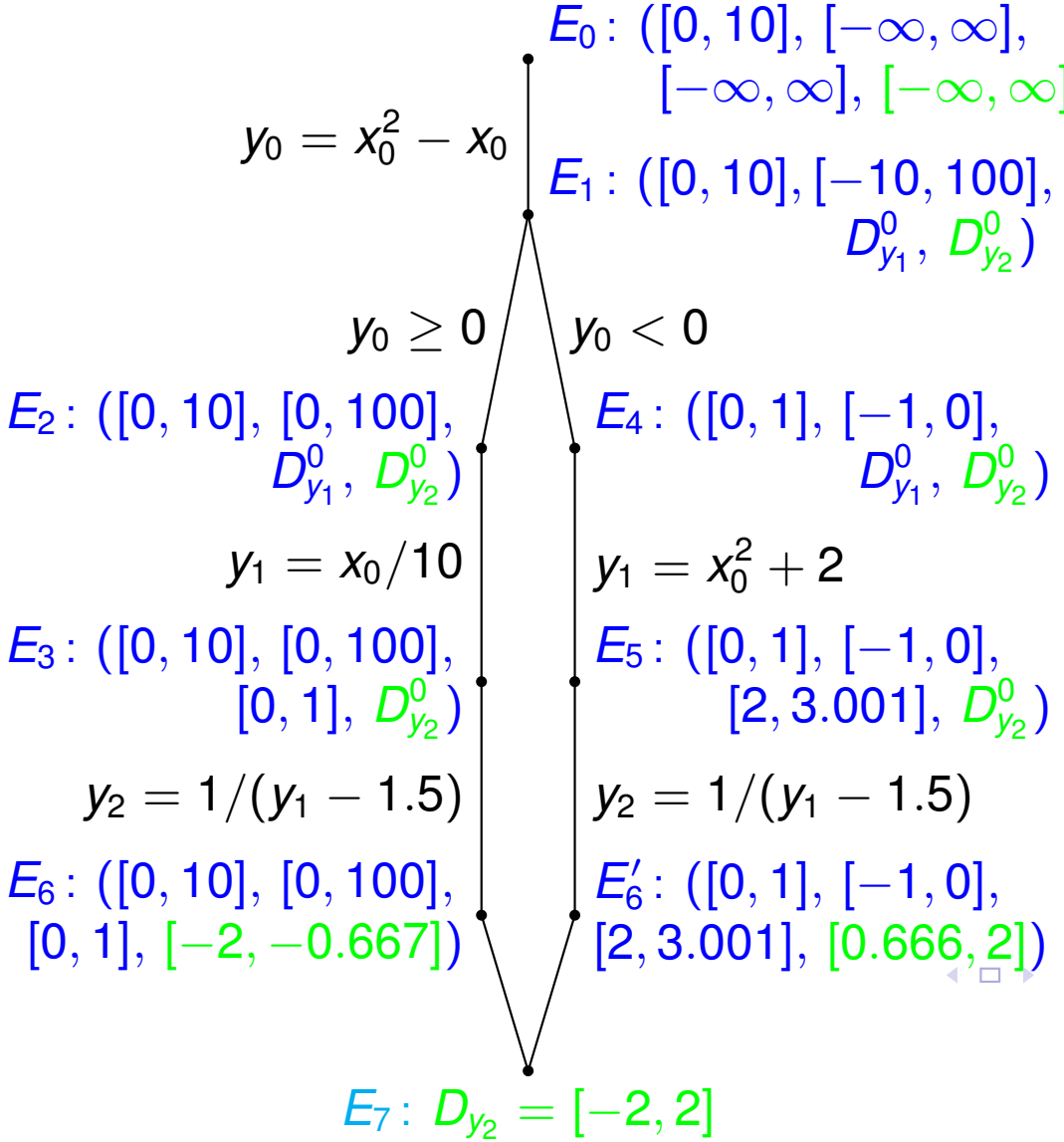
```
float x0 = [0,10];
float y0 = x0*x0 - x0;
if (y0 >= 0)
  y1 = x0/10;
else
  y1 = x0*x0 + 2;
y2 = 1 / (y1-1.5);
```



Refining AI-based approximations, example

Using CP-based local consistencies

```
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);
```



Fluctuat

Static analyzer for C programs based on **Abstract Interpretation**

$\longrightarrow$ **estimating rounding errors and their propagation**

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

Zonotopes

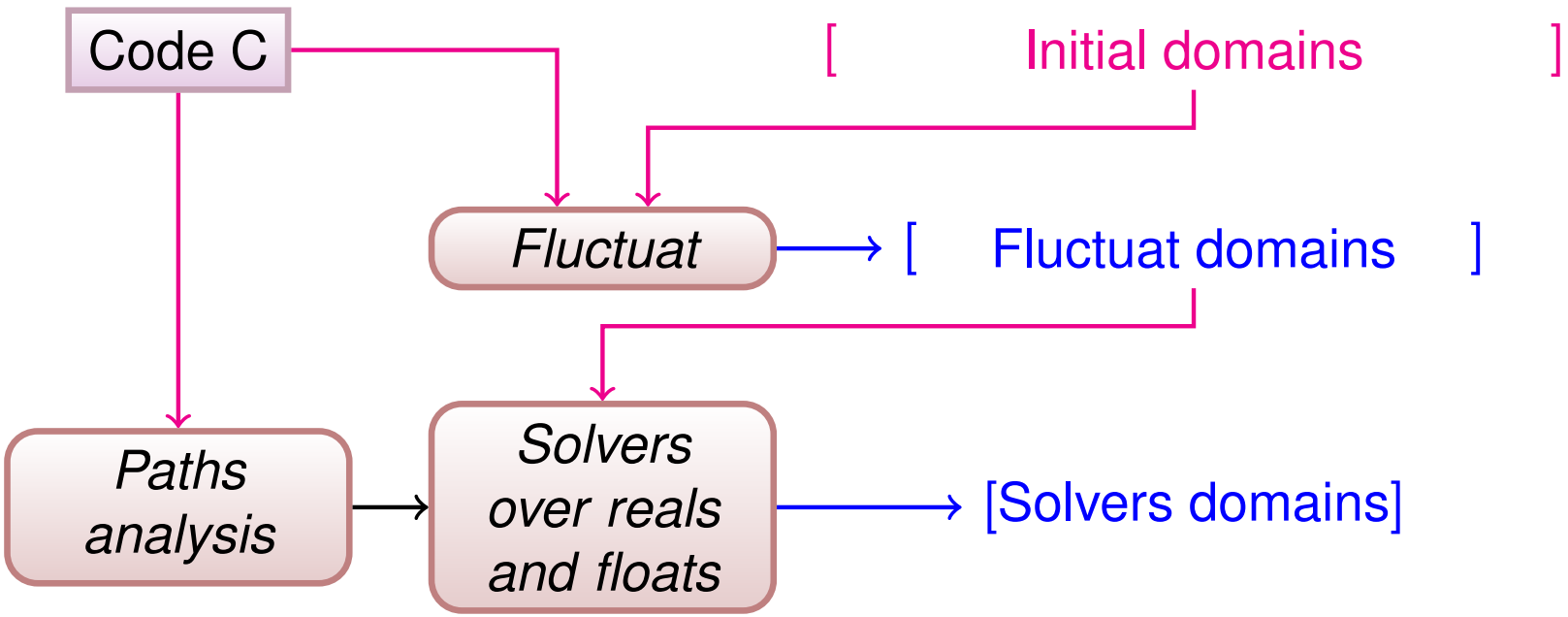
- **Intuition** : **convex polytopes** with a **central symmetry**

  $\rightarrow$ 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, worse than polyhedra
  - Not very accurate for nonlinear terms
  - Not accurate on very common program constructions as **if**

Proposed approach

Overview

We use partial consistencies to reduce the domains of variables calculated by Fluctuat



Proposed approach
Details

- **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

- **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

Experiments

- **Programs...**
  - `quadratic`: computing the roots of a quadratic equation (GSL library) – **conditionals**
  - `sinus7`: program of the 7th-order Taylor series of function sinus – **nonlinearity**
  - `rump`: polynomial of Rump – **pathological cancellation phenomenon**
  - `sqrt`: square root of a number greater than 4 (Babylonian method) – **iterative program**
- **Loss of accuracy of Fluctuat when:**
  - **Union at the earliest** of program states (join operator): `quadratic`, `sqrt`;
  - Domains intersection due to **conditional instructions** (meet operator): `quadratic`, `sqrt`;
  - Interpolation by expansion of approximations in presence of loops (widening operator): `sqrt`;
  - Approximation of **nonlinear expressions**: `quadratic`, `sinus7`, `rump`, `sqrt`.

Results over the reals

	Fluctuat		RealPaver	
	Domain	Time	Domain	Time
quadratic$_1$ x_0	$[-\infty, \infty]$	0.1 s	$[-\infty, 0]$	1.5 s
quadratic$_1$ x_1	$[-\infty, \infty]$	0.1 s	$[-8.011, \infty]$	1.5 s
quadratic$_2$ x_0	$[-2e6, 0]$	0.1 s	$[-1e6, 0]$	0.5 s
quadratic$_2$ x_1	$[-1e6, 0]$	0.1 s	$[-5.186e5, 0]$	0.5 s
sinus7	$[-1.009, 1.009]$	0.1 s	$[-0.842, 0.843]$	0.3 s
rump	$[-1e37, 2e37]$	0.1 s	$[-1e36, 1.7e37]$	1.2 s
sqrt$_1$	$[2.116, 2.354]$	0.1 s	$[2.121, 2.346]$	0.3 s
sqrt$_2$	$[2.098, 3.435]$	0.1 s	$[2.232, 3.165]$	0.5 s

FPCS

Correct solver over floating-point numbers based on 2B-consistency

$\longrightarrow$ preserves all the solutions

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

Results over the floats

	Fluctuat		FPCS	
	Domain	Time	Domain	Time
quadratic$_1$ x_0	$[-\infty, \infty]$	0.1 s	$[-\infty, 0]$	0.3 s
quadratic$_1$ x_1	$[-\infty, \infty]$	0.1 s	$[-8.064, \infty]$	0.3 s
quadratic$_2$ x_0	$[-2e6, 0]$	0.1 s	$[-2e6, 0]$	0.3 s
quadratic$_2$ x_1	$[-1e6, 0]$	0.1 s	$[-2503.8, 0]$	0.3 s
sinus7	$[-1.009, 1.009]$	0.1 s	$[-0.853, 0.852]$	0.2 s
rump	$[-1e37, 2e37]$	0.1 s	$[-1e37, 2e37]$	0.2 s
sqrt$_1$	$[2.116, 2.354]$	0.1 s	$[2.120, 2.347]$	1 s
sqrt$_2$	$[-\infty, \infty]$	0.1 s	$[2.232, 3.168]$	1.6 s

Conclusion

- CP
  - **Advantages:**
    → Good **refutation** capabilities
    → Handling nonlinear constraints
  - **Limits:** Distinct exploration of each executable path is a **critical issue**
- AI
  - **Advantages:**
    → Good **scaling** capabilities
    → Zonotopes are better approximations of linear constraints than boxes
  - **Limits:** **Over-approximations** can be very rough

→ **Complementary** techniques

. . . **The hybrid approach is the best :)**