Stability: an Abstract Domain for the Trend of Variation of Variables

Luca Negrini[1], Sofia Presotto[1], Pietro Ferrara[1], Enea Zaffanella[2], Agostino Cortesi[1]

[1]Department of Environmental Sciences, Informatics and Statistics, Ca' Foscari University of Venice

[2]Department of Mathematics, Physics and Computer Science, University of Parma

luca.negrini@unive.it

NSAD 2024, Pasadena, California, United States

October 22nd, 2024

Abstract domains for numeric values

precision

Sign

Parity

Constants

```
x = input()

if (x > 10)

  y = 1 / x
```

Abstract domains for numeric values

Sign

Parity

Constants

```
x = input()
≀x → ⊤≀
if (x > 10)

  y = 1 / x
```

Abstract domains for numeric values

precision

Sign

Parity

Constants

```
x = input()
⟅x → ⊤⟆
if (x > 10)
  ⟅x → +⟆
  y = 1 / x
```

Abstract domains for numeric values

precision

Sign

Parity

Constants

```
x = input()
≀x → ⊤≀
if (x > 10)
  ≀x → +≀
  y = 1 / x ✓
```

Abstract domains for numeric values

precision

Sign

Parity

Constants

```
x = input()
⦃x → ⊤⦄
if (x > 10)
  ⦃x → +⦄
  y = 1 / x ✓
```

```
x = input()
⦃x → ⊤⦄
if (x > 10)
  ⦃x → +⦄
  y = 1 / (x - 1)
```

Abstract domains for numeric values

precision

Sign

Parity

Constants

```
x = input()
≀x → ⊤≀
if (x > 10)
  ≀x → +≀
  y = 1 / x ✓
```

```
x = input()
≀x → ⊤≀
if (x > 10)
  ≀x → +≀
  y = 1 / (x - 1) x - 1 = ⊤ ✗
```

Abstract domains for numeric values

precision

Sign
Parity
Constants
↓
Intervals
Congruences

```
x = input()
⦃x → [−∞, +∞]⦄
if (x > 10)
  ⦃x → [10, +∞]⦄
  y = 1 / x ✓
```

```
x = input()
⦃x → [−∞, +∞]⦄
if (x > 10)
  ⦃x → [10, +∞]⦄
  y = 1 / (x - 1)  x - 1 = [9, +∞] ✓
```

Abstract domains for numeric values

precision

Sign
Parity
Constants
↓
Intervals
Congruences

```
x = input()
⟅x → [−∞, +∞]⟆
if (x > 10)
  ⟅x → [10, +∞]⟆
  y = 1 / x ✓
```

```
x = input()
⟅x → [−∞, +∞]⟆
if (x > 10)
  ⟅x → [10, +∞]⟆
  y = 1 / (x - 1) x - 1 = [9, +∞] ✓
```

```
x = input_array()

for (int i = 0; i < x.length; i++)

  print(x[i])
```

Abstract domains for numeric values

precision

Sign
Parity
Constants

↓

Intervals
Congruences

```
x = input()
⦃x → [−∞, +∞]⦄
if (x > 10)
  ⦃x → [10, +∞]⦄
  y = 1 / x ✓
```

```
x = input()
⦃x → [−∞, +∞]⦄
if (x > 10)
  ⦃x → [10, +∞]⦄
  y = 1 / (x - 1) x - 1 = [9, +∞] ✓
```

```
x = input_array()
⦃x.length → [0, +∞]⦄
for (int i = 0; i < x.length; i++)

  print(x[i])
```

Abstract domains for numeric values

precision

Sign
Parity
Constants

↓

Intervals
Congruences

```
x = input()
⟅x → [−∞, +∞]⟆
if (x > 10)
  ⟅x → [10, +∞]⟆
  y = 1 / x ✓
```

```
x = input()
⟅x → [−∞, +∞]⟆
if (x > 10)
  ⟅x → [10, +∞]⟆
  y = 1 / (x - 1) x - 1 = [9, +∞] ✓
```

```
x = input_array()
⟅x.length → [0, +∞]⟆
for (int i = 0; i < x.length; i++)
  ⟅i → [0, +∞]⟆
  print(x[i])
```

Abstract domains for numeric values

precision

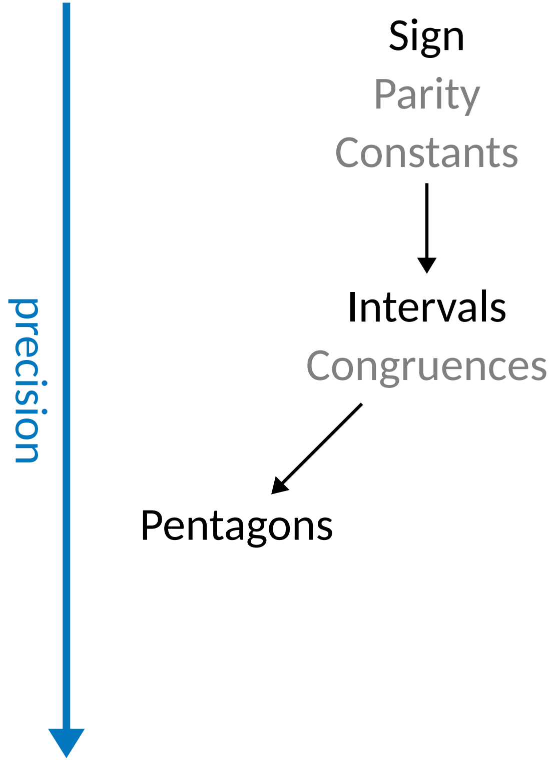
Sign
Parity
Constants
↓
Intervals
Congruences

```
x = input()
⟅x → [−∞, +∞]⟆
if (x > 10)
  ⟅x → [10, +∞]⟆
  y = 1 / x ✓
```

```
x = input()
⟅x → [−∞, +∞]⟆
if (x > 10)
  ⟅x → [10, +∞]⟆
  y = 1 / (x - 1) x - 1 = [9, +∞] ✓
```

```
x = input_array()
⟅x.length → [0, +∞]⟆
for (int i = 0; i < x.length; i++)
  ⟅i → [0, +∞]⟆
  print(x[i]) ✗
```

Abstract domains for numeric values

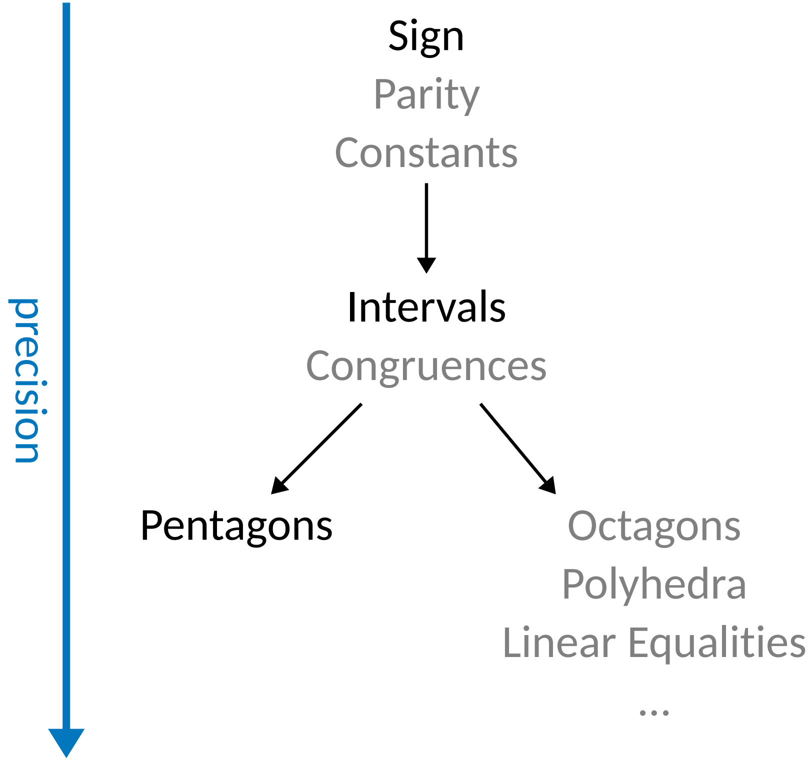



```
x = input()
⟅x → [−∞, +∞]⟆
if (x > 10)
  ⟅x → [10, +∞]⟆
  y = 1 / x ✓
```

```
x = input()
⟅x → [−∞, +∞]⟆
if (x > 10)
  ⟅x → [10, +∞]⟆
  y = 1 / (x - 1)  x - 1 = [9, +∞] ✓
```

```
x = input_array()
⟅x.length → [0, +∞]⟆
for (int i = 0; i < x.length; i++)
  ⟅i → [0, +∞] ∧ i < x.length⟆
  print(x[i]) ✓
```

Abstract domains for numeric values




```
x = input()
⟅x → [−∞, +∞]⟆
if (x > 10)
  ⟅x → [10, +∞]⟆
  y = 1 / x ✓
```

```
x = input()
⟅x → [−∞, +∞]⟆
if (x > 10)
  ⟅x → [10, +∞]⟆
  y = 1 / (x - 1) x - 1 = [9, +∞] ✓
```

```
x = input_array()
⟅x.length → [0, +∞]⟆
for (int i = 0; i < x.length; i++)
  ⟅i → [0, +∞] ∧ i < x.length⟆
  print(x[i]) ✓
```

Where are we aiming?

Non-relational domains

- Rough precision
- Good scalability
- Enough for simple properties

Relational domains

- Higher precision
- Higher computational cost
- Can express more complex invariants

Where are we aiming?

Non-relational domains

- Rough precision
- Good scalability
- Enough for simple properties

Relational domains

- Higher precision
- Higher computational cost
- Can express more complex invariants

Stability

- Non-relational (fast)
- Some relational reasoning
- Able to tell something about unknown values

The starting point: a simple Solidity contract

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount;
    balances[dest] += amount;
  }
}
```

The starting point: a simple Solidity contract

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount;
    balances[dest] += amount;
  }
}
```

▷ Target invariant: `balances[msg.sender]` decreases, `balances[dest]` increases

The starting point: a simple Solidity contract

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount;
    balances[dest] += amount;
  }
}
```

- Target invariant: `balances[msg.sender]` decreases, `balances[dest]` increases
- Problem: `balances[msg.sender]`, `balances[dest]` and `amount` are unknown

The starting point: a simple Solidity contract

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount;
    balances[dest] += amount;
  }
}
```

- Target invariant: `balances[msg.sender]` decreases, `balances[dest]` increases
- Problem: `balances[msg.sender]`, `balances[dest]` and `amount` are unknown
- No-go for non-relational analyses: everything is $\top$

The starting point: a simple Solidity contract

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount;
    balances[dest] += amount;
  }
}
```

- Target invariant: `balances[msg.sender]` decreases, `balances[dest]` increases
- Problem: `balances[msg.sender]`, `balances[dest]` and `amount` are unknown
- No-go for non-relational analyses: everything is $\top$
- Relational analyses *might* learn something (but not in general)

Our idea: tracking per-variable trends

Keep relations between different values of the same variable:

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount;
    balances[dest] += amount;
  }
}
```

Our idea: tracking per-variable trends

Keep relations between different values of the same variable:

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount; amount > 0 ⟹ balances[msg.sender] decreased
    balances[dest] += amount;
  }
}
```

Our idea: tracking per-variable trends

Keep relations between different values of the same variable:

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount; amount > 0 ⟹ balances[msg.sender] decreased
    balances[dest] += amount; amount > 0 ⟹ balances[dest] increased
  }
}
```

Our idea: tracking per-variable trends

Keep relations between different values of the same variable:

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount; amount > 0 ⟹ balances[msg.sender] decreased
    balances[dest] += amount; amount > 0 ⟹ balances[dest] increased
  }
}
```

▷ No (direct) knowledge on the values of variables needed

Our idea: tracking per-variable trends

Keep relations between different values of the same variable:

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount; amount > 0 ⟹ balances[msg.sender] decreased
    balances[dest] += amount; amount > 0 ⟹ balances[dest] increased
  }
}
```

▷ No (direct) knowledge on the values of variables needed

▷ Not tailored to specific variable relations

Our idea: tracking per-variable trends

Keep relations between different values of the same variable:

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount; amount > 0 ⟹ balances[msg.sender] decreased
    balances[dest] += amount; amount > 0 ⟹ balances[dest] increased
  }
}
```

▷ No (direct) knowledge on the values of variables needed

▷ Not tailored to specific variable relations

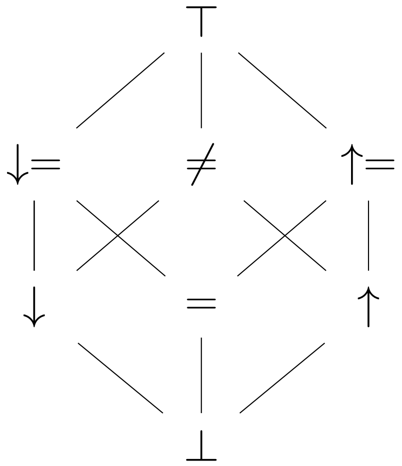
▷ Needs information on some expressions (e.g., sign)

The Stability abstract domain

The Stability abstract domain

The lattice of per-variable trends:

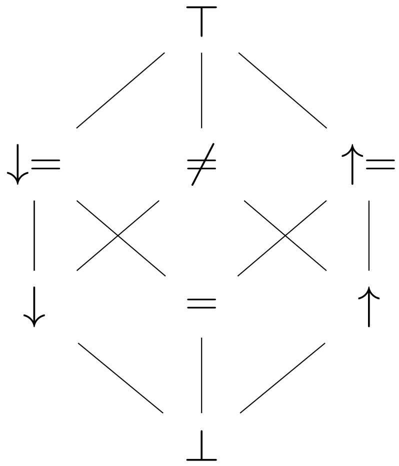
$\mathsf{Trn} \triangleq$



The Stability abstract domain

The lattice of per-variable trends:

$\mathsf{Trn} \triangleq$



The stability lattice:

$$\mathsf{Stb} \triangleq \langle \mathbb{V}\mathsf{ar} \to \mathsf{Trn}, \dot{\sqsubseteq}_{\mathsf{Trn}}, \dot{\sqcup}_{\mathsf{Trn}}, \dot{\sqcap}_{\mathsf{Trn}}, \bot_{\mathsf{Stb}}, \top_{\mathsf{Stb}} \rangle$$

Stb gives trends to variables w.r.t. **the previous instruction**

The Stability abstract domain (contd.)

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {

    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount;

    balances[dest] += amount;

  }
}
```

The Stability abstract domain (contd.)

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    ≀dest → =, amount → =, balances[msg.sender] → =, balances[dest] → =≀
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount;

    balances[dest] += amount;

  }
}
```

The Stability abstract domain (contd.)

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    ⦃dest → =, amount → =, balances[msg.sender] → =, balances[dest] → =⦄
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount;
    ⦃dest → =, amount → =, balances[msg.sender] → ↓, balances[dest] → =⦄
    balances[dest] += amount;

  }
}
```

The Stability abstract domain (contd.)

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    ≀dest → =, amount → =, balances[msg.sender] → =, balances[dest] → =∫
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount;
    ≀dest → =, amount → =, balances[msg.sender] → ↓, balances[dest] → =∫
    balances[dest] += amount;
    ≀dest → =, amount → =, balances[msg.sender] → =, balances[dest] → ↑∫
  }
}
```

Chaining Stability information

Stb instances can be combined to obtain trends between two arbitrary program points:

Chaining Stability information

Stb instances can be combined to obtain trends between two arbitrary program points:

1. Element-wise trend combination:

t_1 \ t_2	$\bot$	$\uparrow=$	$\uparrow$	$=$	$\neq$	$\downarrow$	$\downarrow=$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\uparrow=$	$\bot$	$\uparrow=$	$\uparrow$	$\uparrow=$	$\top$	$\top$	$\top$	$\top$
$\uparrow$	$\bot$	$\uparrow$	$\uparrow$	$\uparrow$	$\top$	$\top$	$\top$	$\top$
$=$	$\bot$	$\uparrow=$	$\uparrow$	$=$	$\neq$	$\downarrow$	$\downarrow=$	$\top$
$\neq$	$\bot$	$\top$	$\top$	$\neq$	$\top$	$\top$	$\top$	$\top$
$\downarrow$	$\bot$	$\top$	$\top$	$\downarrow$	$\top$	$\downarrow$	$\downarrow$	$\top$
$\downarrow=$	$\bot$	$\top$	$\top$	$\downarrow=$	$\top$	$\downarrow$	$\downarrow=$	$\top$
$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$

Chaining Stability information

Stb instances can be combined to obtain trends between two arbitrary program points:

1. Element-wise trend combination:

t_1 \ t_2	$\bot$	$\uparrow=$	$\uparrow$	$=$	$\neq$	$\downarrow$	$\downarrow=$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\uparrow=$	$\bot$	$\uparrow=$	$\uparrow$	$\uparrow=$	$\top$	$\top$	$\top$	$\top$
$\uparrow$	$\bot$	$\uparrow$	$\uparrow$	$\uparrow$	$\top$	$\top$	$\top$	$\top$
$=$	$\bot$	$\uparrow=$	$\uparrow$	$=$	$\neq$	$\downarrow$	$\downarrow=$	$\top$
$\neq$	$\bot$	$\top$	$\top$	$\neq$	$\top$	$\top$	$\top$	$\top$
$\downarrow$	$\bot$	$\top$	$\top$	$\downarrow$	$\top$	$\downarrow$	$\downarrow$	$\top$
$\downarrow=$	$\bot$	$\top$	$\top$	$\downarrow=$	$\top$	$\downarrow$	$\downarrow=$	$\top$
$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$

2. Combine over different paths with $\dot{\sqcup}_{\text{Trn}}$

Chaining Stability information (contd.)

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    ≀dest → =, amount → =, balances[msg.sender] → =, balances[dest] → =≀
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount;
    ≀dest → =, amount → =, balances[msg.sender] → ↓, balances[dest] → =≀

    balances[dest] += amount;
    ≀dest → =, amount → =, balances[msg.sender] → =, balances[dest] → ↑≀

  }
}
```

Chaining Stability information (contd.)

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    ≀dest → =, amount → =, balances[msg.sender] → =, balances[dest] → =∫
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount;
    ≀dest → =, amount → =, balances[msg.sender] → ↓, balances[dest] → =∫
    ≀dest → =, amount → =, balances[msg.sender] → ↓, balances[dest] → =∫
    balances[dest] += amount;
    ≀dest → =, amount → =, balances[msg.sender] → =, balances[dest] → ↑∫

  }
}
```

Chaining Stability information (contd.)

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    ≀dest → =, amount → =, balances[msg.sender] → =, balances[dest] → =⟆
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount;
    ≀dest → =, amount → =, balances[msg.sender] → ↓, balances[dest] → =⟆
    ≀dest → =, amount → =, balances[msg.sender] → ↓, balances[dest] → =⟆
    balances[dest] += amount;
    ≀dest → =, amount → =, balances[msg.sender] → =, balances[dest] → ↑⟆
    ≀dest → =, amount → =, balances[msg.sender] → ↓, balances[dest] → ↑⟆
  }
}
```

The need for additional information

Stability cannot be inferred through stability alone:

- Does `x` increase after `x = x - y`?
- Does `x` increase after `x = 99999`?

The need for additional information

Stability cannot be inferred through stability alone:

- Does `x` increase after `x = x - y`?
- Does `x` increase after `x = 99999`?

An oracle that knows at least signs of expressions is needed!

The need for additional information

Stability cannot be inferred through stability alone:

- Does `x` increase after `x = x - y`?
- Does `x` increase after `x = 99999`?

An oracle that knows at least signs of expressions is needed!

Stability works in **open product** with a numeric domain $\mathcal{A}$:

$$\mathbb{S}^\sharp[\![\ldots]\!] \triangleq \begin{cases} \ldots & \text{if } \mathrm{Q}_{\mathcal{A}}(\mathsf{e}_1) \\ \ldots & \text{if } \mathrm{Q}_{\mathcal{A}}(\mathsf{e}_2) \\ \ldots & \end{cases}$$

The need for additional information

Stability cannot be inferred through stability alone:

- Does `x` increase after `x = x - y`?
- Does `x` increase after `x = 99999`?

An oracle that knows at least signs of expressions is needed!

Stability works in **open product** with a numeric domain $\mathcal{A}$:

$$\mathbb{S}^\sharp[\![\ldots]\!] \triangleq \begin{cases} \ldots & \text{if } \mathrm{Q}_{\mathcal{A}}(\mathsf{e}_1) \\ \ldots & \text{if } \mathrm{Q}_{\mathcal{A}}(\mathsf{e}_2) \\ \ldots & \end{cases}$$

Practical note: no product needed, only a pre-analysis

Examples

...

$\text{Sign} = \{\!| x \to -, y \to 0^{+} |\!\}$ $\text{Stb} = \{\!| x \to =, y \to \downarrow= |\!\}$

```
x = x + 3 * y
```

$\text{Sign} = \{\!| x \to \top, y \to 0^{+} |\!\}$

...

Examples

...

$\text{Sign} = ⟅x \to -, y \to 0^{+}⟆ \quad \text{Stb} = ⟅x \to =, y \to {\downarrow}{=}⟆$

```
x = x + 3 * y
```

$\text{Sign} = ⟅x \to \top, y \to 0^{+}⟆ \quad \text{Stb} = ⟅x \to =, y \to =⟆$

...

1. Reset all variables to $=$

Examples

...

$\text{Sign} = \llparenthesis x \rightarrow -, y \rightarrow 0^{+} \rrparenthesis \quad \text{Stb} = \llparenthesis x \rightarrow =, y \rightarrow \downarrow= \rrparenthesis$

```
x = x + 3 * y
```

$\text{Sign} = \llparenthesis x \rightarrow \top, y \rightarrow 0^{+} \rrparenthesis \quad \text{Stb} = \llparenthesis x \rightarrow =, y \rightarrow = \rrparenthesis$

...

1. Reset all variables to $=$
2. Query Sign repeatedly

Examples

```
...
Sign = ≀x → −, y → 0⁺)∫   Stb = ≀x → =, y → ↓=∫
x = x + 3 * y
Sign = ≀x → ⊤, y → 0⁺)∫   Stb = ≀x →=, y → =∫
...
```

1. Reset all variables to $=$
2. Query Sign repeatedly
   2.1. $Q_{\mathcal{A}}(3 * y > 0)$? ✗

Examples

...

$\text{Sign} = \lbag x \to -, y \to 0^{+} \rbag \quad \text{Stb} = \lbag x \to =, y \to \downarrow= \rbag$

```
x = x + 3 * y
```

$\text{Sign} = \lbag x \to \top, y \to 0^{+} \rbag \quad \text{Stb} = \lbag x \to =, y \to = \rbag$

...

1. Reset all variables to $=$
2. Query Sign repeatedly
   2.1. $\mathbb{Q}_{\mathcal{A}}(3 \,*\, y > 0)$? ✗
   2.2. $\mathbb{Q}_{\mathcal{A}}(3 \,*\, y < 0)$? ✗

Examples

...

$\text{Sign} = \{x \to -, y \to 0^+\}$ $\text{Stb} = \{x \to =, y \to \downarrow=\}$

```
x = x + 3 * y
```

$\text{Sign} = \{x \to \top, y \to 0^+\}$ $\text{Stb} = \{x \to =, y \to =\}$

...

1. Reset all variables to $=$
2. Query Sign repeatedly

   2.1. $\mathbb{Q}_{\mathcal{A}}(3 * y > 0)$? ✗

   2.2. $\mathbb{Q}_{\mathcal{A}}(3 * y < 0)$? ✗

   2.3. $\mathbb{Q}_{\mathcal{A}}(3 * y == 0)$? ✗

Examples

...

$\text{Sign} = \{\!\!\{ x \to -, y \to 0^+ \}\!\!\}$ $\text{Stb} = \{\!\!\{ x \to =, y \to \downarrow= \}\!\!\}$

```
x = x + 3 * y
```

$\text{Sign} = \{\!\!\{ x \to \top, y \to 0^+ \}\!\!\}$ $\text{Stb} = \{\!\!\{ x \to \uparrow=, y \to = \}\!\!\}$

...

1. Reset all variables to $=$
2. Query Sign repeatedly

   2.1. $Q_{\mathcal{A}}(3 * y > 0)$? ✗

   2.2. $Q_{\mathcal{A}}(3 * y < 0)$? ✗

   2.3. $Q_{\mathcal{A}}(3 * y == 0)$? ✗

   2.4. $Q_{\mathcal{A}}(3 * y \geq 0)$? ✓

Examples

```
...
Sign = ⦃x → −, y → 0⁺⦄   Stb = ⦃x → =, y → ↓=⦄
x = x + 3 * y
Sign = ⦃x → ⊤, y → 0⁺⦄   Stb = ⦃x → ↑=, y → =⦄
...
```

1. Reset all variables to $=$

2. Query Sign repeatedly

   2.1. $\mathbb{Q}_{\mathcal{A}}(3 * y > 0)$? ✗

   2.2. $\mathbb{Q}_{\mathcal{A}}(3 * y < 0)$? ✗

   2.3. $\mathbb{Q}_{\mathcal{A}}(3 * y == 0)$? ✗

   2.4. $\mathbb{Q}_{\mathcal{A}}(3 * y \geq 0)$? ✓

   2.5. ...

Examples (contd.)

...

$\text{Sign} = \lbag x \to -, y \to + \rbag \quad \text{Stb} = \lbag x \to =, y \to \downarrow= \rbag$

```
x = x * (y + 2)
```

$\text{Sign} = \lbag x \to -, y \to + \rbag \quad \text{Stb} = \lbag x \to =, y \to = \rbag$

...

1. Reset all variables to $=$
2. Query Sign repeatedly

Examples (contd.)

```
...
Sign = ⦃x → −, y → +⦄  Stb = ⦃x → =, y → ↓=⦄
x = x * (y + 2)
Sign = ⦃x → −, y → +⦄  Stb = ⦃x → =, y → =⦄
...
```

1. Reset all variables to $=$

2. Query Sign repeatedly

   2.1. $Q_{\mathcal{A}}(\mathtt{x} == 0) \vee$
   $Q_{\mathcal{A}}(\mathtt{(y + 2)} == 1)$? ✗

Examples (contd.)

```
...
Sign = ⦇x → −, y → +⦈   Stb = ⦇x → =, y → ↓=⦈
x = x * (y + 2)
Sign = ⦇x → −, y → +⦈   Stb = ⦇x → =, y → =⦈
...
```

1. Reset all variables to $=$
2. Query Sign repeatedly

   2.1. $\mathrm{Q}_{\mathcal{A}}(\mathtt{x == 0}) \vee$
   $\mathrm{Q}_{\mathcal{A}}(\mathtt{(y + 2) == 1})$? ✗

   2.2. $\mathrm{Q}_{\mathcal{A}}(\mathtt{x < 0 \&\& (y + 2) > 1}) \vee$
   $\mathrm{Q}_{\mathcal{A}}(\mathtt{x > 0 \&\& (y + 2) < 1})$? ✗

Examples (contd.)

```
...
Sign = ⦃x → −, y → +⦄   Stb = ⦃x → =, y → ↓=⦄
x = x * (y + 2)
Sign = ⦃x → −, y → +⦄   Stb = ⦃x → =, y → =⦄
...
```

1. Reset all variables to $=$
2. Query Sign repeatedly

   2.1. $Q_{\mathcal{A}}(\texttt{x == 0}) \vee Q_{\mathcal{A}}(\texttt{(y + 2) == 1})$? ✗

   2.2. $Q_{\mathcal{A}}(\texttt{x < 0 \&\& (y + 2) > 1}) \vee Q_{\mathcal{A}}(\texttt{x > 0 \&\& (y + 2) < 1})$? ✗

   2.3. $Q_{\mathcal{A}}(\texttt{x < 0 \&\& (y + 2) < 1}) \vee Q_{\mathcal{A}}(\texttt{x > 0 \&\& (y + 2) > 1})$? ✗

Examples (contd.)

```
...
Sign = ⦇x → −, y → +⦈   Stb = ⦇x → =, y → ↓=⦈
x = x * (y + 2)
Sign = ⦇x → −, y → +⦈   Stb = ⦇x → ⊤, y → =⦈
...
```

1. Reset all variables to $=$
2. Query Sign repeatedly

   2.1. $Q_{\mathcal{A}}(\texttt{x == 0}) \lor Q_{\mathcal{A}}(\texttt{(y + 2) == 1})$? ✗

   2.2. $Q_{\mathcal{A}}(\texttt{x < 0 \&\& (y + 2) > 1}) \lor Q_{\mathcal{A}}(\texttt{x > 0 \&\& (y + 2) < 1})$? ✗

   2.3. $Q_{\mathcal{A}}(\texttt{x < 0 \&\& (y + 2) < 1}) \lor Q_{\mathcal{A}}(\texttt{x > 0 \&\& (y + 2) > 1})$? ✗

   2.4. ...

Examples (contd.)

```
...
Intv = ⦃x → [−12, −3], y → [1, +∞]⦄  Stb = ⦃x → =, y → ↓=⦄
x = x * (y + 2)
Intv = ⦃x → [−∞, −9], y → [1, +∞]⦄  Stb = ⦃x → ↓, y → =⦄
...
```

1. Reset all variables to $=$
2. Query Intv repeatedly

   2.1. $Q_{\mathcal{A}}(\texttt{x == 0}) \vee Q_{\mathcal{A}}(\texttt{(y + 2) == 1})$? ✗

   2.2. $Q_{\mathcal{A}}(\texttt{x < 0 \&\& (y + 2) > 1}) \vee Q_{\mathcal{A}}(\texttt{x > 0 \&\& (y + 2) < 1})$? ✓

   2.3. $Q_{\mathcal{A}}(\texttt{x < 0 \&\& (y + 2) < 1}) \vee Q_{\mathcal{A}}(\texttt{x > 0 \&\& (y + 2) > 1})$? ✗

   2.4. ...

Where can this be applied?

Covariance and Contravariance

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount;
    balances[dest] += amount;
    ≀dest → =, amount → =, balances[msg.sender] → ↓, balances[dest] → ↑≀
  }
}
```

▷ Weak relation between pairs/groups of variables

Covariance and Contravariance

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount;
    balances[dest] += amount;
    ≀dest → =, amount → =, balances[msg.sender] → ↓, balances[dest] → ↑≀
  }
}
```

▷ Weak relation between pairs/groups of variables

▷ Byproduct of the chained Stability

Covariance and Contravariance

```
contract Coin {
  mapping (address => uint) public balances;
  // [...]
  function send(address dest, uint amount) public {
    require(amount > 0);
    require(amount <= balances[msg.sender]);
    balances[msg.sender] -= amount;
    balances[dest] += amount;
    ≀dest → =, amount → =, balances[msg.sender] → ↓, balances[dest] → ↑≀
  }
}
```

- ▷ Weak relation between pairs/groups of variables
- ▷ Byproduct of the chained Stability
- ▷ Can be used to prove functional requirements

Termination (light)

```
int foo(int x, int y) {

  while (x > 0 && y > 0) {

    y = 2 * y;

    x = x - 1;


  }

  return x + y;

}
```

▷ Analyze the program with Stability

Termination (light)

```
int foo(int x, int y) {
  ⦃{x ↦ =, y ↦ =}⦄
  while (x > 0 && y > 0) {

    y = 2 * y;

    x = x - 1;


  }

  return x + y;

}
```

▷ Analyze the program with Stability

Termination (light)

```
int foo(int x, int y) {
  ⟅{x ↦ =, y ↦ =}⟆
  while (x > 0 && y > 0) {
    ⟅{x ↦ =, y ↦ =}⟆
    y = 2 * y;

    x = x - 1;


  }

  return x + y;

}
```

▷ Analyze the program with Stability

Termination (light)

```
int foo(int x, int y) {
  ≀{x ↦ =, y ↦ =}≀
  while (x > 0 && y > 0) {
    ≀{x ↦ =, y ↦ =}≀
    y = 2 * y;
    ≀{x ↦ =, y ↦ ↑}≀
    x = x - 1;


  }

  return x + y;

}
```

▷ Analyze the program with Stability

Termination (light)

```
int foo(int x, int y) {
  ⟅{x ↦ =, y ↦ =}⟆
  while (x > 0 && y > 0) {
    ⟅{x ↦ =, y ↦ =}⟆
    y = 2 * y;
    ⟅{x ↦ =, y ↦ ↑}⟆
    x = x - 1;
    ⟅{x ↦ ↓, y ↦ =}⟆

  }

  return x + y;

}
```

▷ Analyze the program with Stability

Termination (light)

```
int foo(int x, int y) {
  ⦃{x ↦ =, y ↦ =}⦄
  while (x > 0 && y > 0) {
    ⦃{x ↦ ↓=, y ↦ =}⦄
    y = 2 * y;
    ⦃{x ↦ =, y ↦ ↑}⦄
    x = x - 1;
    ⦃{x ↦ ↓, y ↦ =}⦄

  }

  return x + y;

}
```

▷ Analyze the program with Stability

Termination (light)

```
int foo(int x, int y) {
  ≀{x ↦ =, y ↦ =}≀
  while (x > 0 && y > 0) {
    ≀{x ↦ ↓=, y ↦ =}≀
    y = 2 * y;
    ≀{x ↦ =, y ↦ ↑}≀
    x = x - 1;
    ≀{x ↦ ↓, y ↦ =}≀

  }
  ≀{x ↦ =, y ↦ =}≀
  return x + y;

}
```

▷ Analyze the program with Stability

Termination (light)

```
int foo(int x, int y) {
  ⟅{x ↦ =, y ↦ =}⟆
  while (x > 0 && y > 0) {
    ⟅{x ↦ ↓=, y ↦ =}⟆
    y = 2 * y;
    ⟅{x ↦ =, y ↦ ↑}⟆
    x = x - 1;
    ⟅{x ↦ ↓, y ↦ =}⟆

  }
  ⟅{x ↦ =, y ↦ =}⟆
  return x + y;
  ⟅{x ↦ =, y ↦ =}⟆
}
```

▷ Analyze the program with Stability

Termination (light)

```
int foo(int x, int y) {
  ≀{x ↦ =, y ↦ =}∫
  while (x > 0 && y > 0) {
    ≀{x ↦ ↓=, y ↦ =}∫
    y = 2 * y;
    ≀{x ↦ =, y ↦ ↑}∫
    x = x - 1;
    ≀{x ↦ ↓, y ↦ =}∫
    ≀{x ↦ ↓, y ↦ ↑}∫
  }
  ≀{x ↦ =, y ↦ =}∫
  return x + y;
  ≀{x ↦ =, y ↦ =}∫
}
```

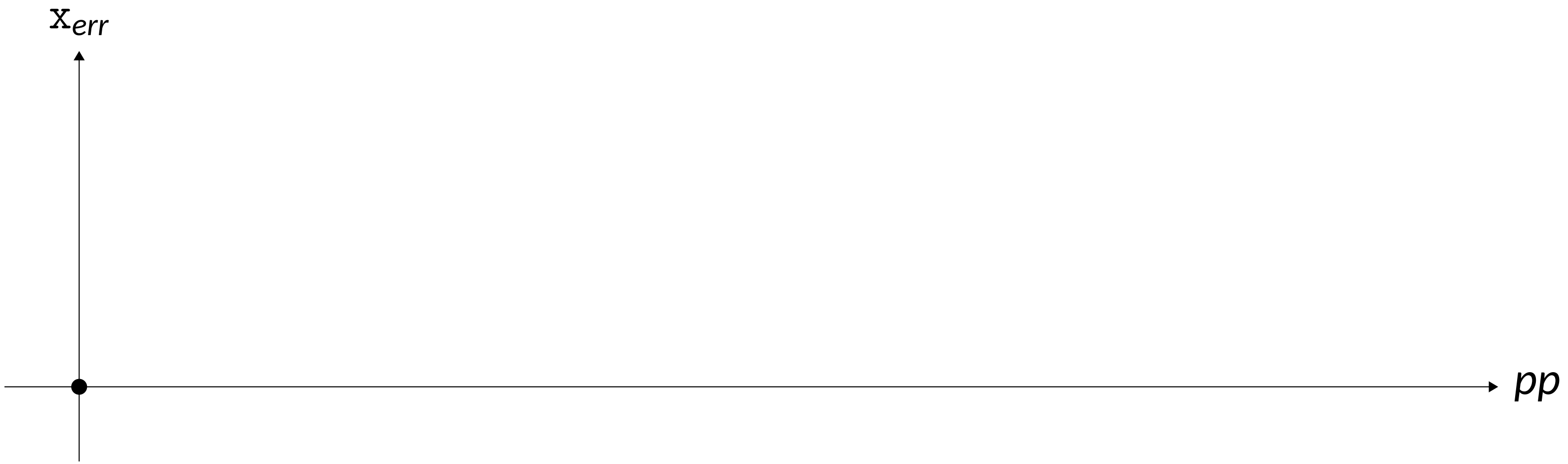
- ▷ Analyze the program with Stability
- ▷ Chain Stability info at end of loops

Termination (light)

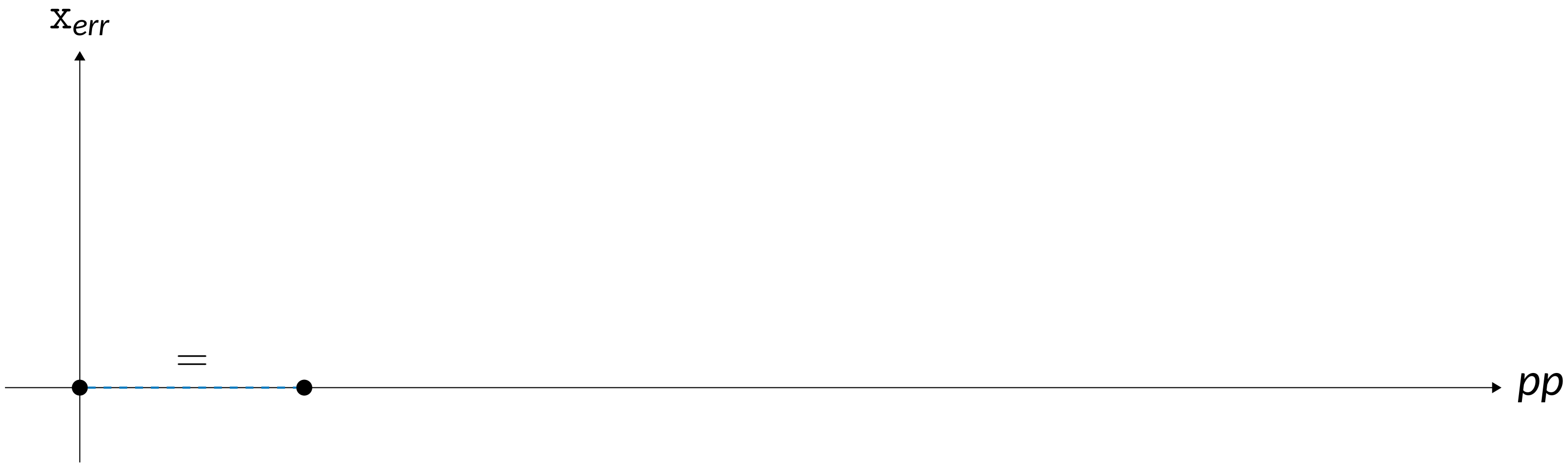
```
int foo(int x, int y) {
  ⦃{x ↦ =, y ↦ =}⦄
  while (x > 0 && y > 0) {
    ⦃{x ↦ ↓=, y ↦ =}⦄
    y = 2 * y;
    ⦃{x ↦ =, y ↦ ↑}⦄
    x = x - 1;
    ⦃{x ↦ ↓, y ↦ =}⦄
    ⦃{x ↦ ↓, y ↦ ↑}⦄
  }
  ⦃{x ↦ =, y ↦ =}⦄
  return x + y;
  ⦃{x ↦ =, y ↦ =}⦄
}
```

- ▷ Analyze the program with Stability
- ▷ Chain Stability info at end of loops
- ▷ Chained information can sometimes prove termination

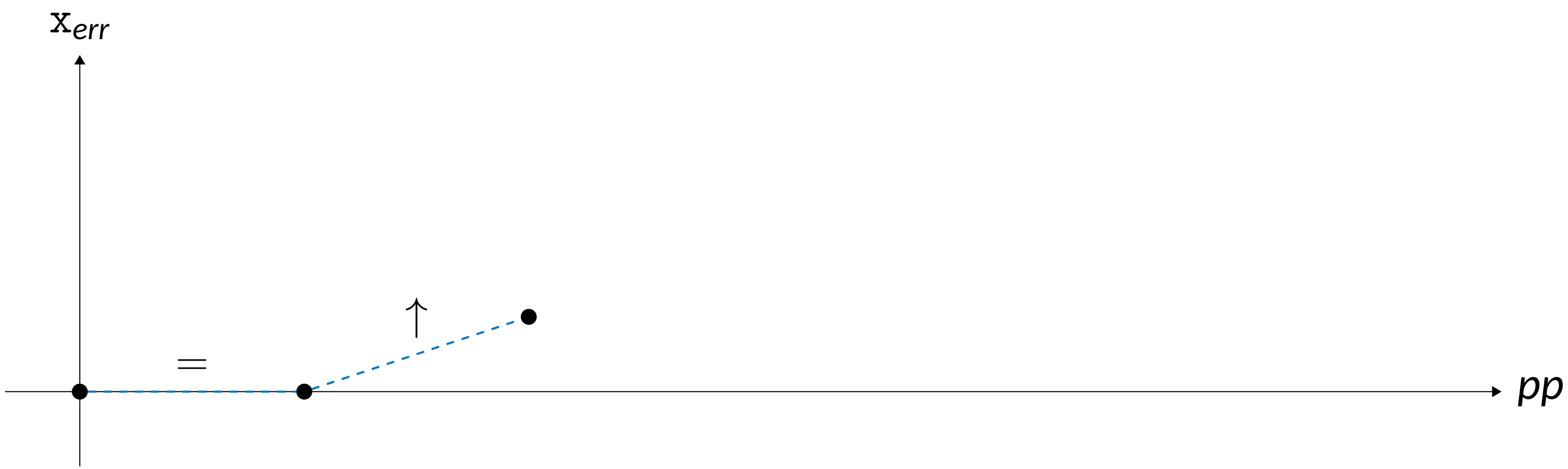
Floating point error



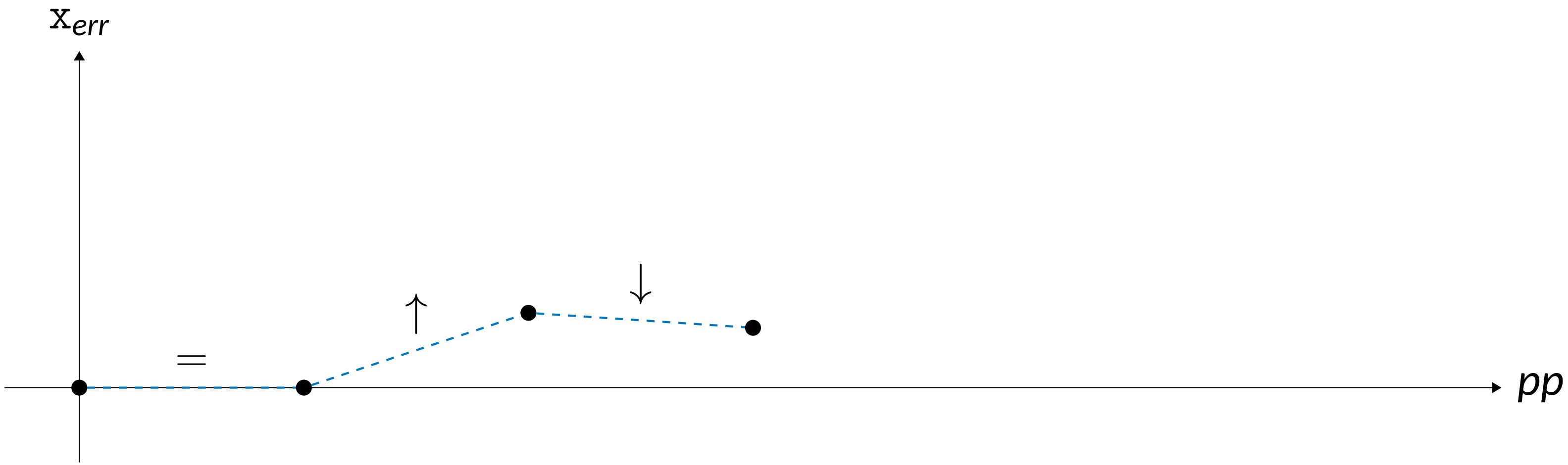
Floating point error



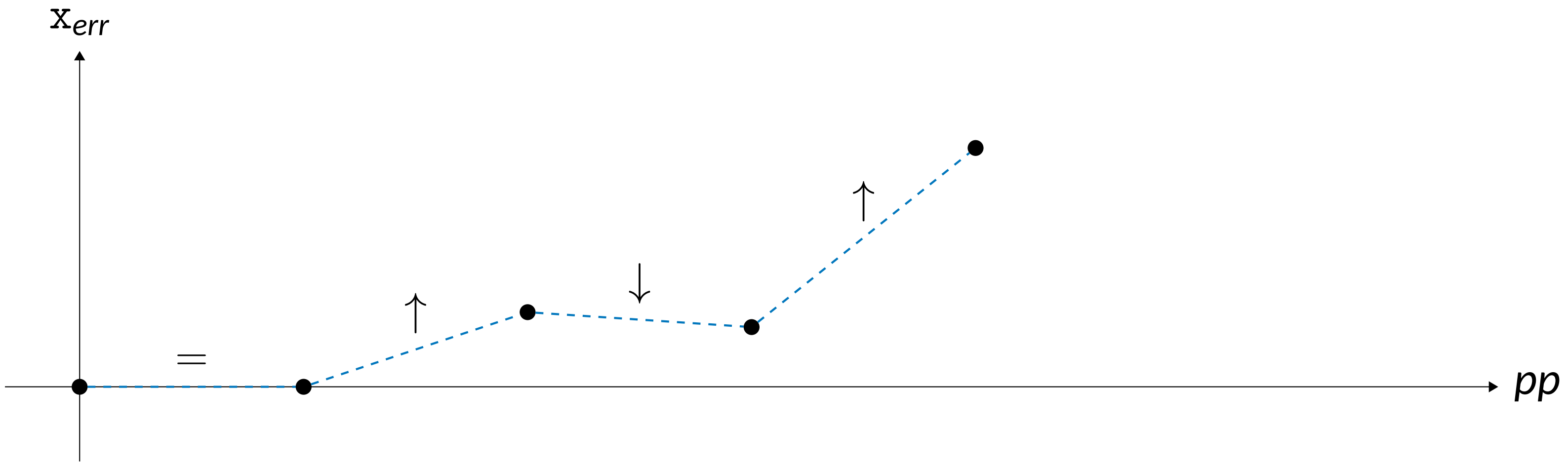
Floating point error



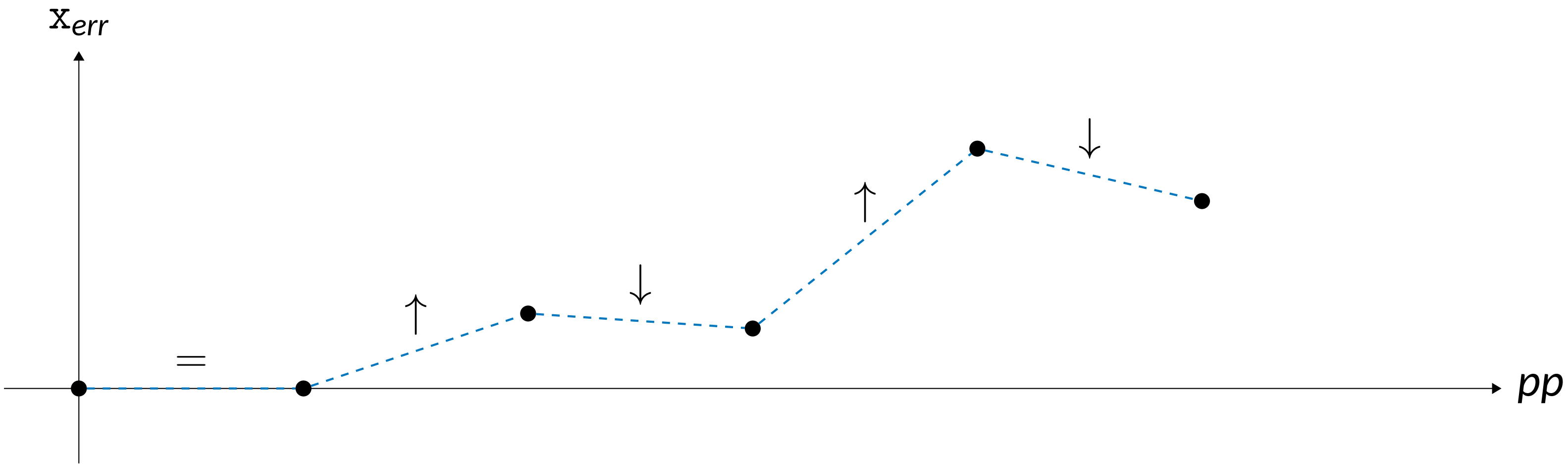
Floating point error



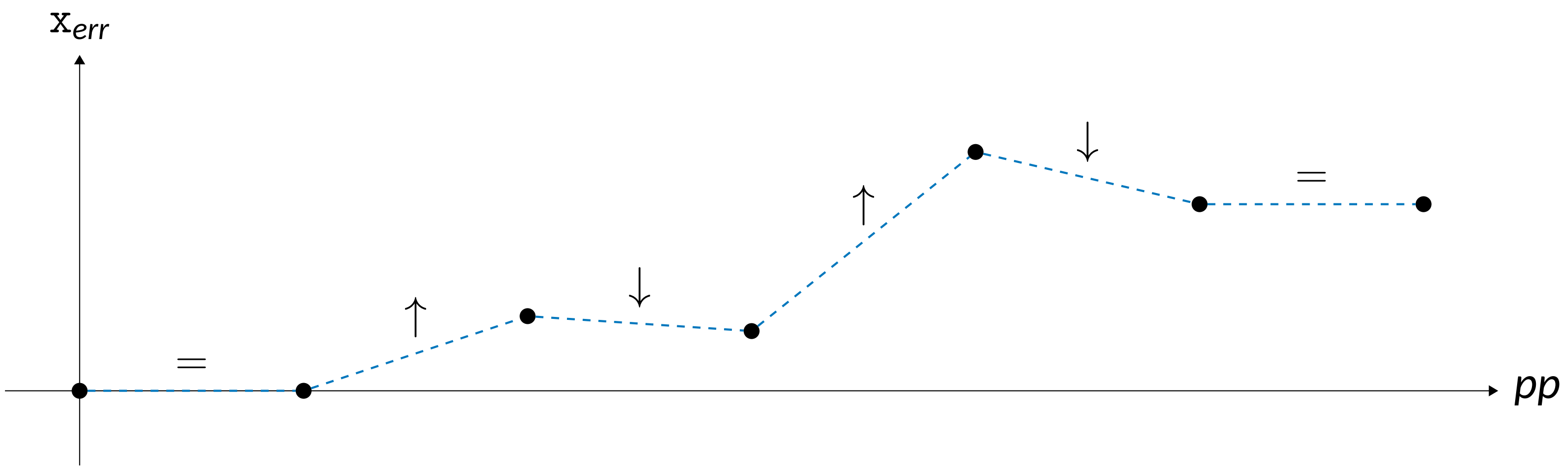
Floating point error



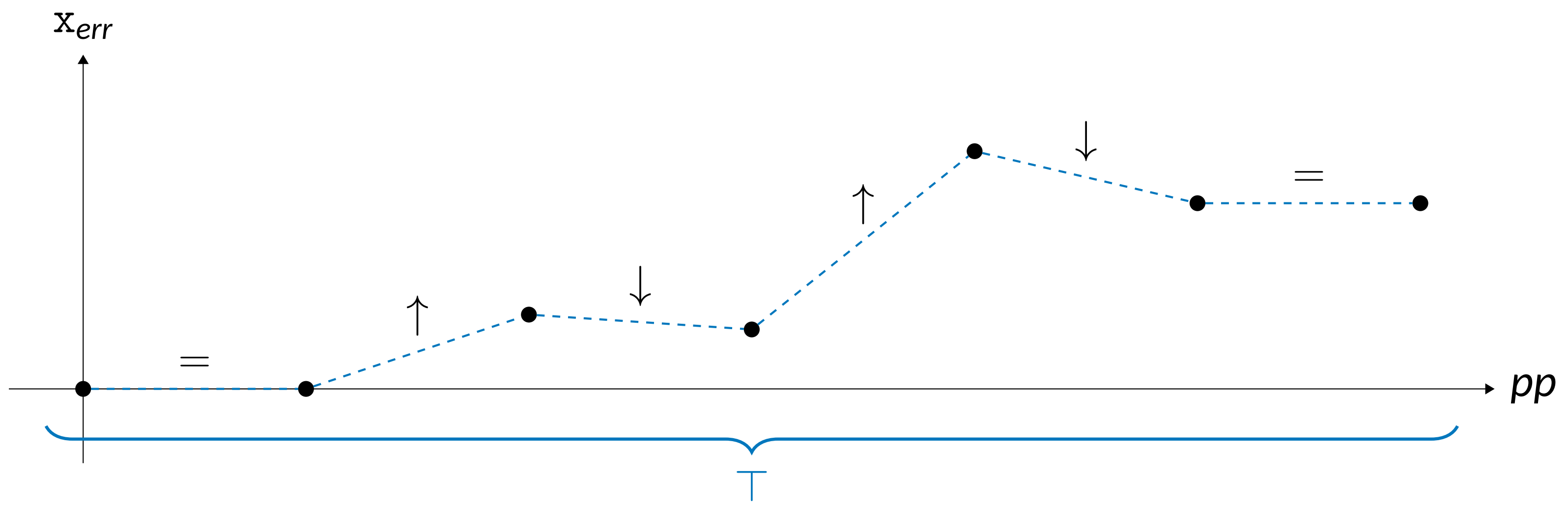
Floating point error



Floating point error



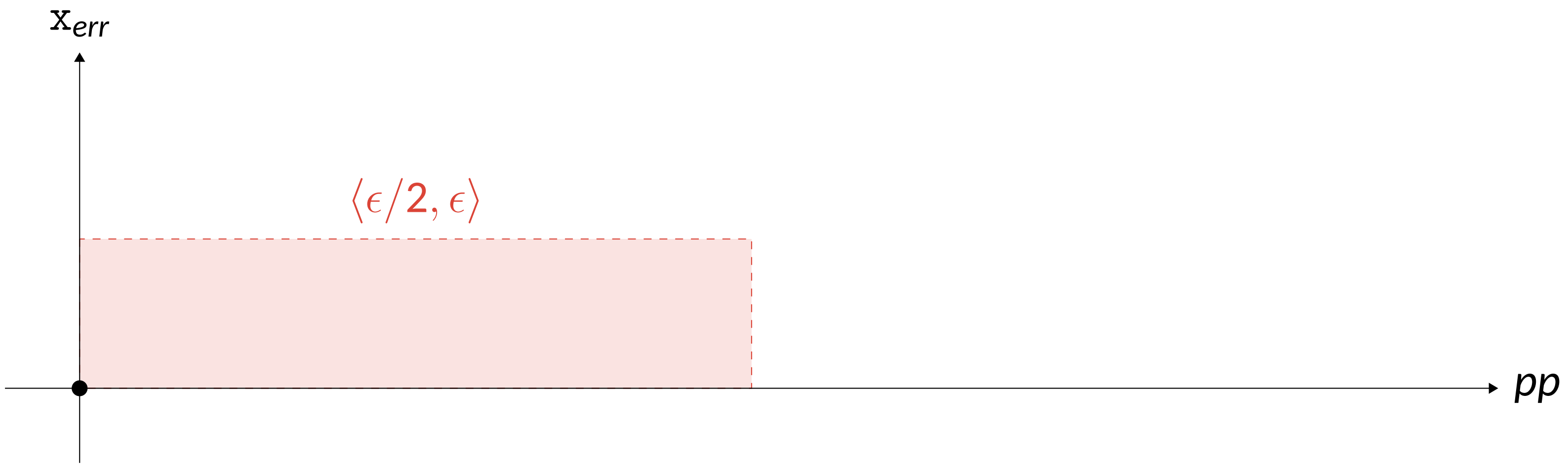
Floating point error



Tiny fluctuations cause the trend to be unstable too quickly

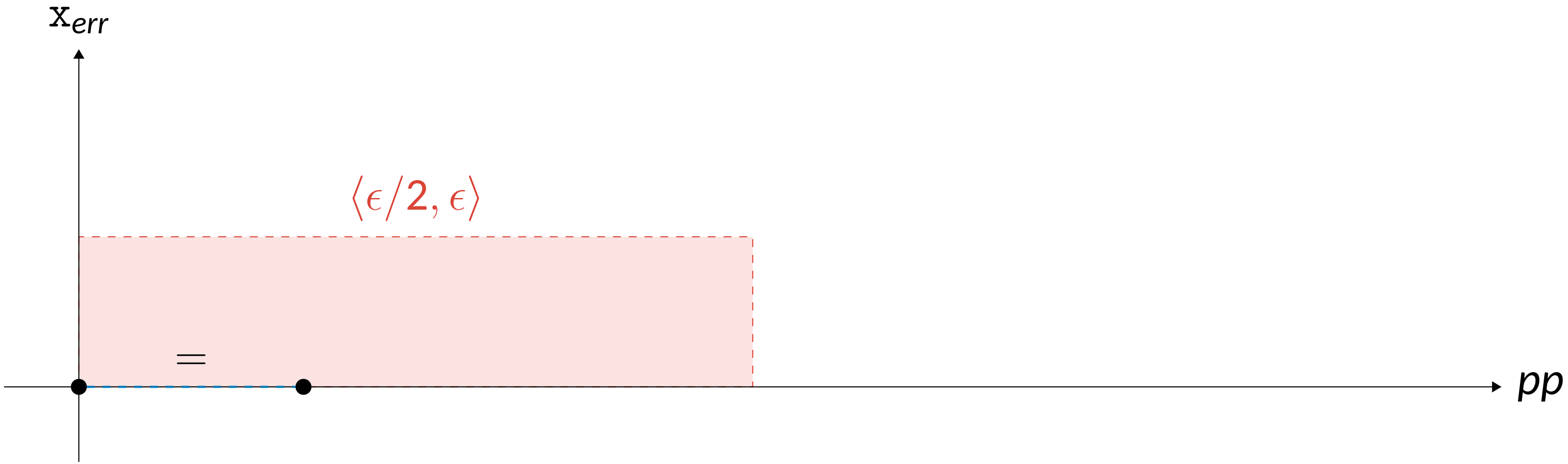
Floating point error (contd.)

Main idea: introduce a sliding tolerance window $\langle \rho, \epsilon \rangle$:



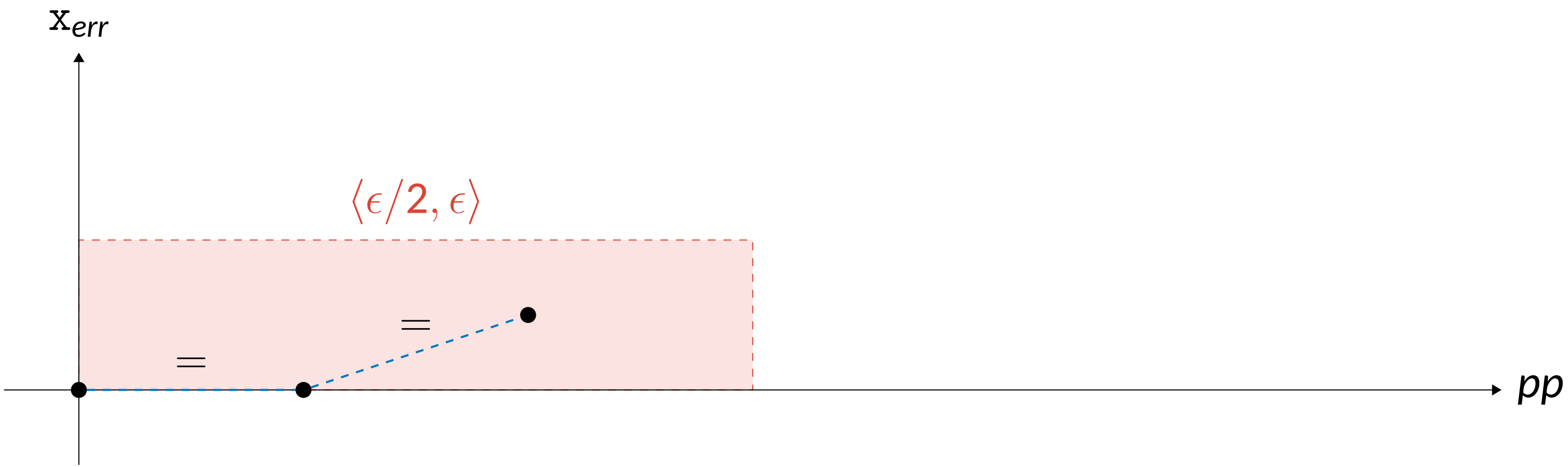
Floating point error (contd.)

Main idea: introduce a sliding tolerance window $\langle \rho, \epsilon \rangle$:



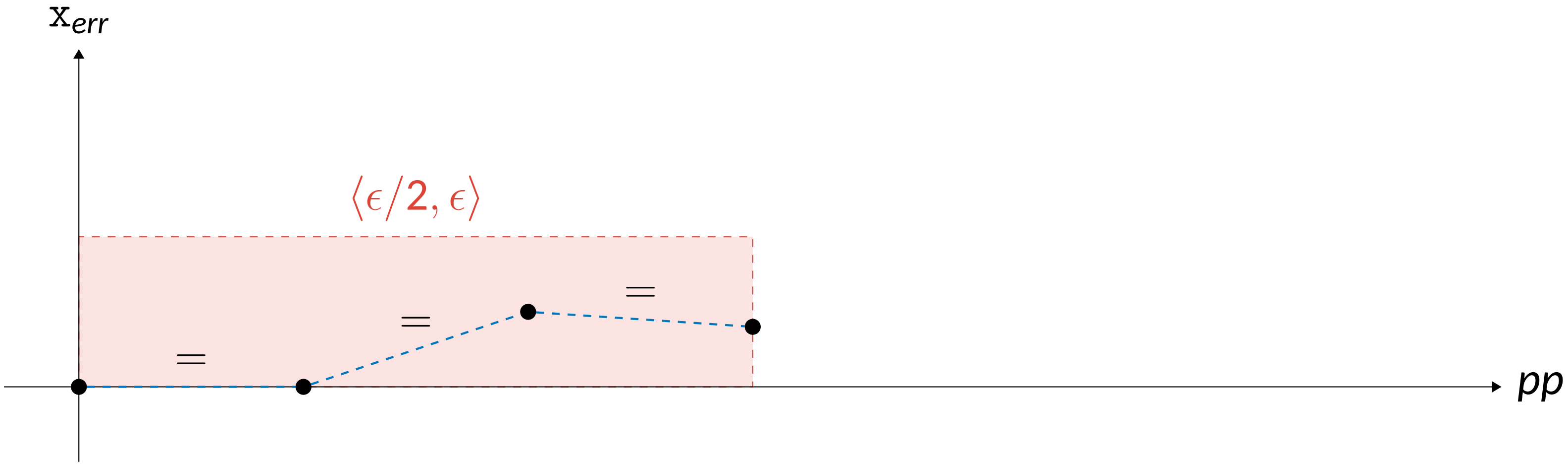
Floating point error (contd.)

Main idea: introduce a sliding tolerance window $\langle \rho, \epsilon \rangle$:



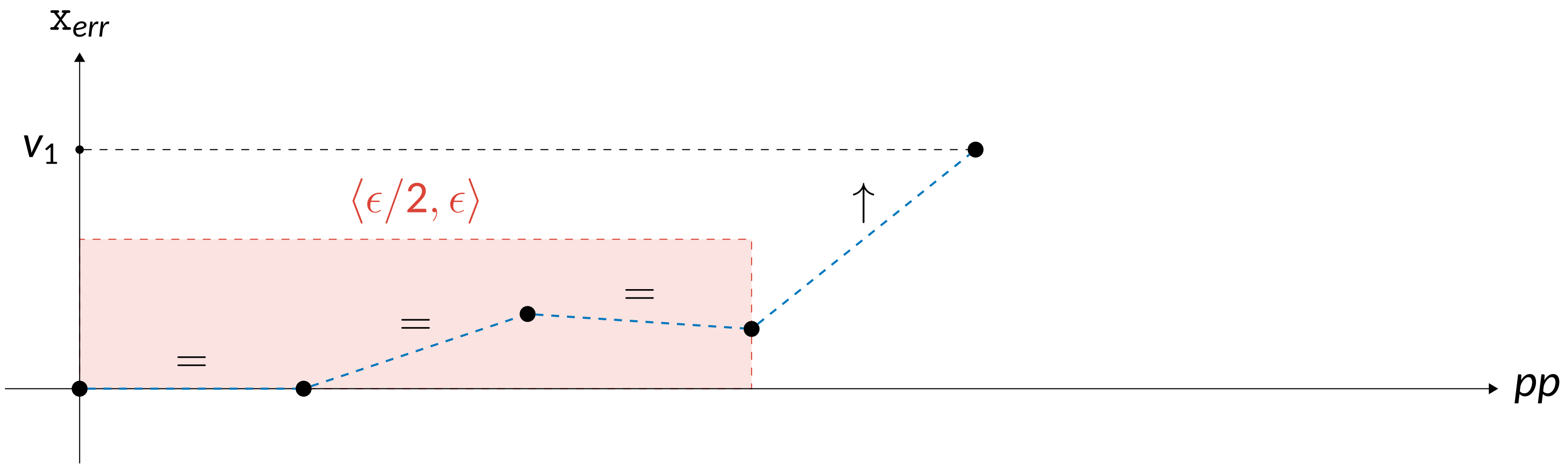
Floating point error (contd.)

Main idea: introduce a sliding tolerance window $\langle \rho, \epsilon \rangle$:



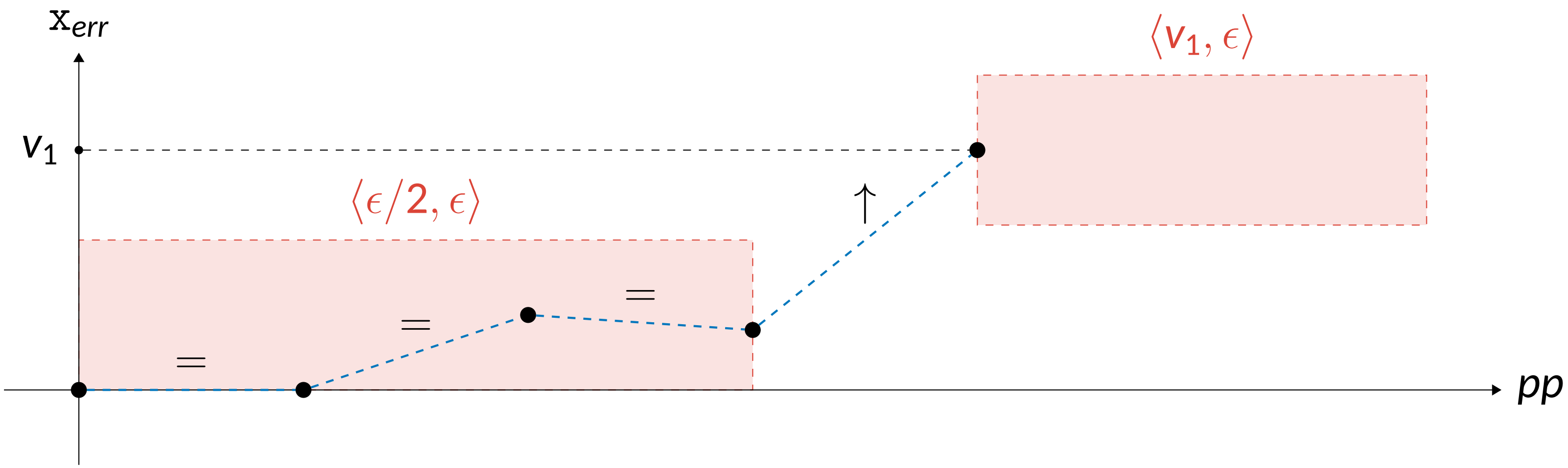
Floating point error (contd.)

Main idea: introduce a sliding tolerance window $\langle \rho, \epsilon \rangle$:



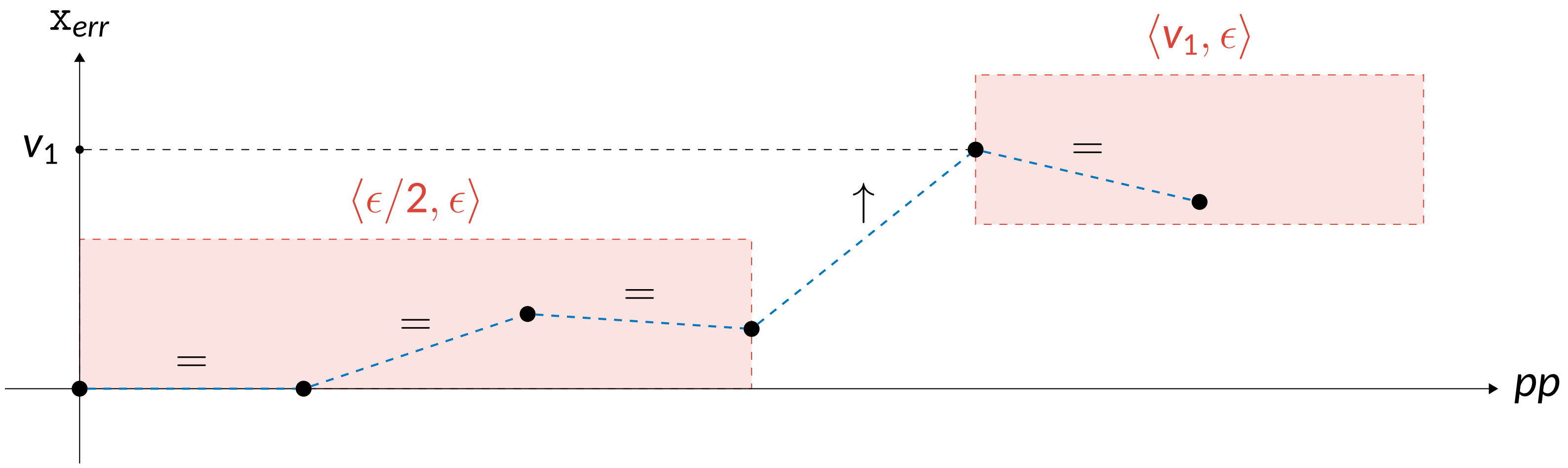
Floating point error (contd.)

Main idea: introduce a sliding tolerance window $\langle \rho, \epsilon \rangle$:



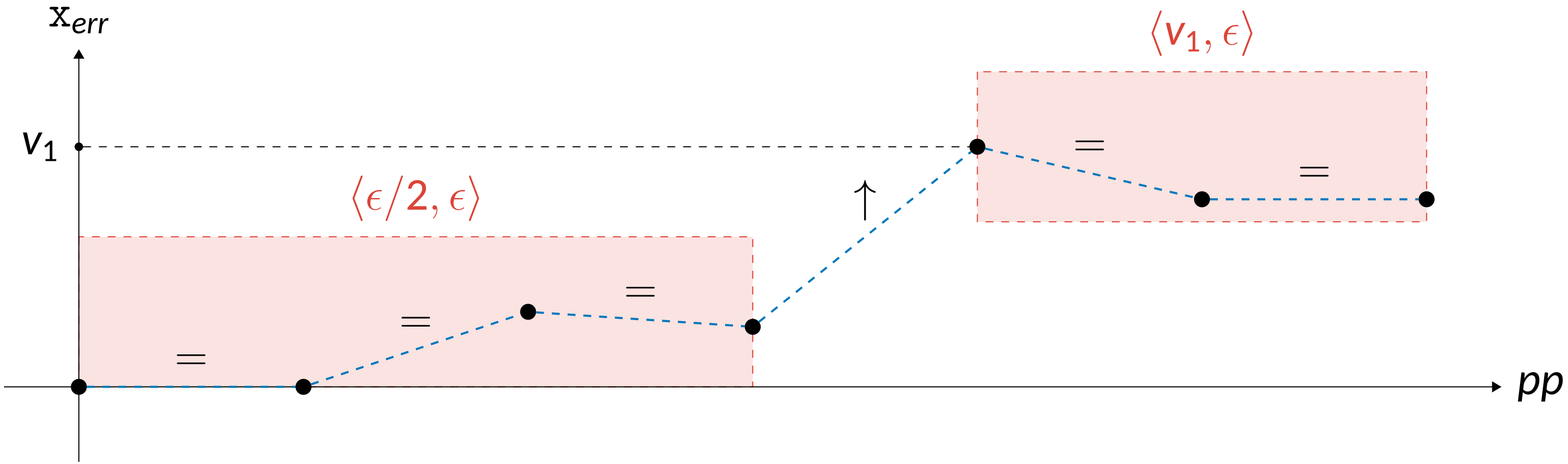
Floating point error (contd.)

Main idea: introduce a sliding tolerance window $\langle \rho, \epsilon \rangle$:



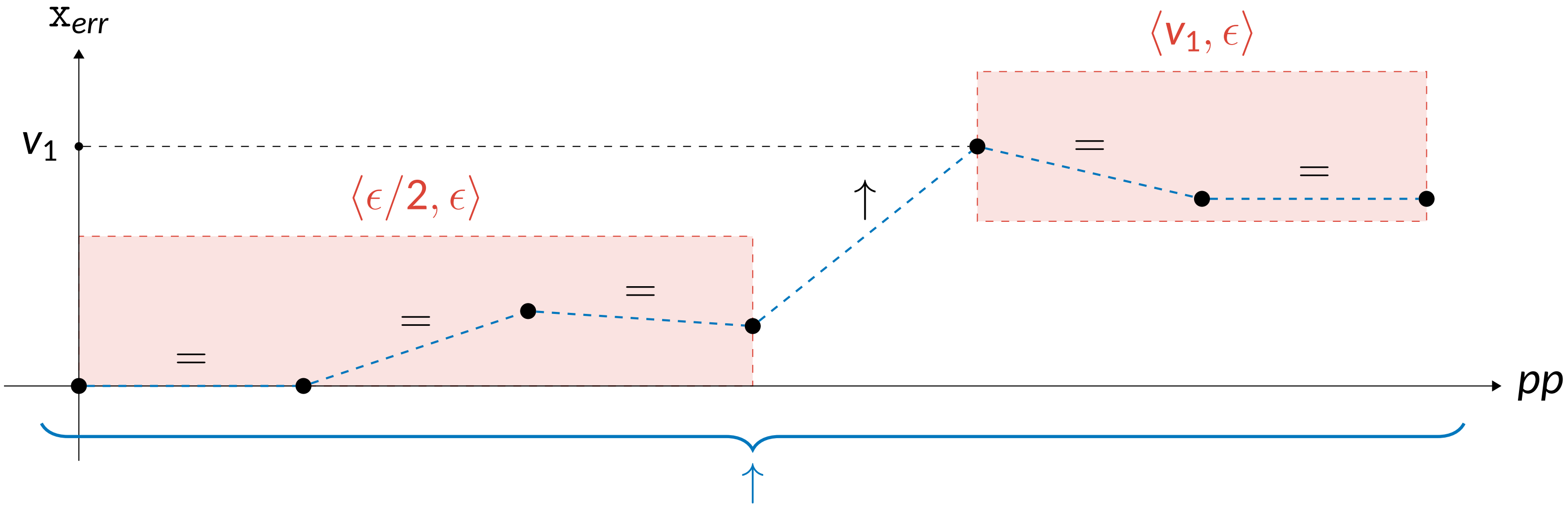
Floating point error (contd.)

Main idea: introduce a sliding tolerance window $\langle \rho, \epsilon \rangle$:



Floating point error (contd.)

Main idea: introduce a sliding tolerance window $\langle \rho, \epsilon \rangle$:



We can show stability up to ϵ!

Summing up

Stability is a new abstract domain for numerical trends

Summing up

Stability is a new abstract domain for numerical trends

- Relates values of the same variable at different program points

Summing up

Stability is a new abstract domain for numerical trends

- Relates values of the same variable at different program points
- Able to track information on $\top$ values for other non-relational domains

Summing up

Stability is a new abstract domain for numerical trends

- Relates values of the same variable at different program points
- Able to track information on $\top$ values for other non-relational domains
- Fast: non-relational and finite

Summing up

Stability is a new abstract domain for numerical trends

- Relates values of the same variable at different program points
- Able to track information on $\top$ values for other non-relational domains
- Fast: non-relational and finite

Stability is a work in progress

Summing up

Stability is a new abstract domain for numerical trends

- Relates values of the same variable at different program points
- Able to track information on $\top$ values for other non-relational domains
- Fast: non-relational and finite

Stability is a work in progress

- Still searching for a target application

Summing up

Stability is a new abstract domain for numerical trends

- Relates values of the same variable at different program points
- Able to track information on $\top$ values for other non-relational domains
- Fast: non-relational and finite

Stability is a work in progress

- Still searching for a target application
- Benchmarks needed to confirm scalability

Summing up

Stability is a new abstract domain for numerical trends

- Relates values of the same variable at different program points
- Able to track information on $\top$ values for other non-relational domains
- Fast: non-relational and finite

Stability is a work in progress

- Still searching for a target application
- Benchmarks needed to confirm scalability
- Proofs needed once we settle on the semantics

Thanks!

luca.negrini@unive.it