An Order Theory Framework of Recurrence Equations for Static Cost Analysis

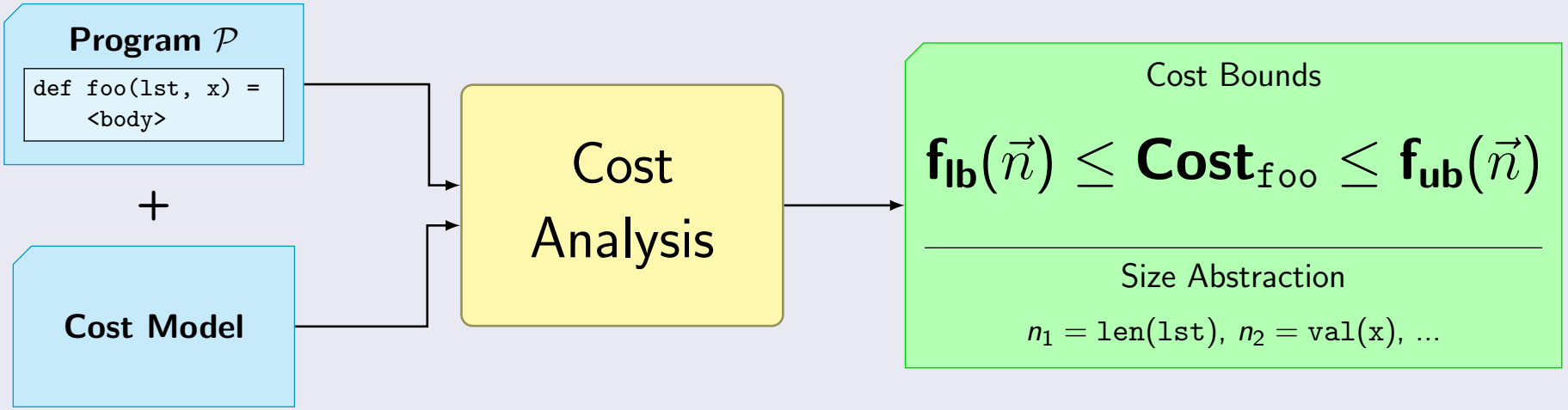
Dynamic Inference of Non-Linear Inequality Invariants

Louis Rustenholz[1,3], Pedro López-García[2,3], José F. Morales[1,3] and Manuel V. Hermenegildo[1,3]

[1]Universidad Politécnica de Madrid (UPM), Spain
[2]Spanish Council for Scientific Research (CSIC), Spain
[3]IMDEA Software Institute, Spain

SAS, October 21st, 2024

Cost Analysis: Bounds on Resource Consumption



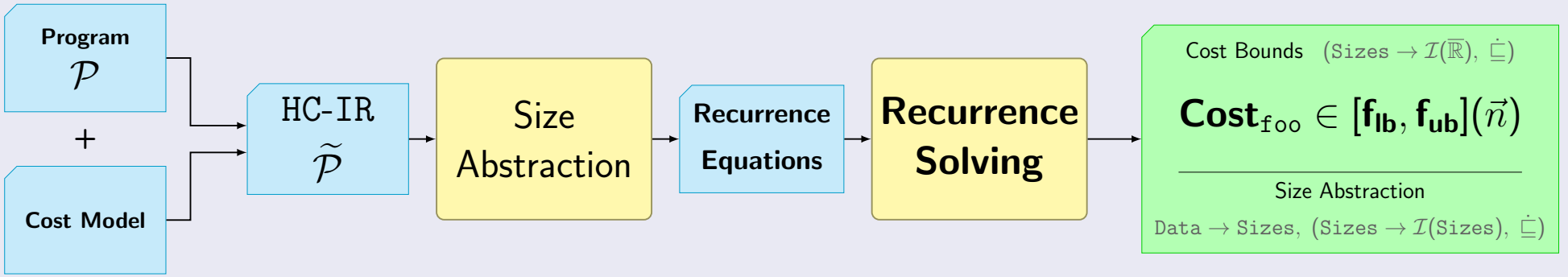
WCET,
Side-Channel

Parallelisation,
Scheduling

Transparency, Optimisation

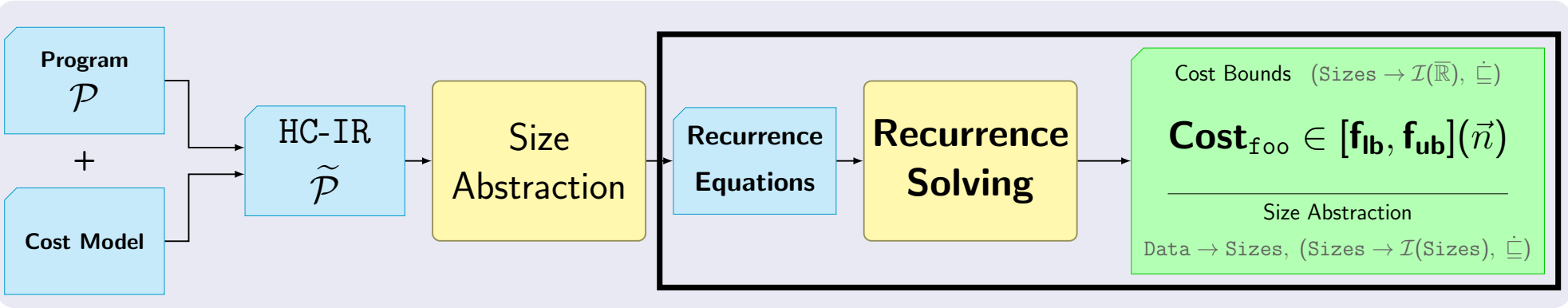
Recurrence-Based Cost Analysis

- Pipeline implemented in CiaoPP (and other analysers)



Recurrence-Based Cost Analysis

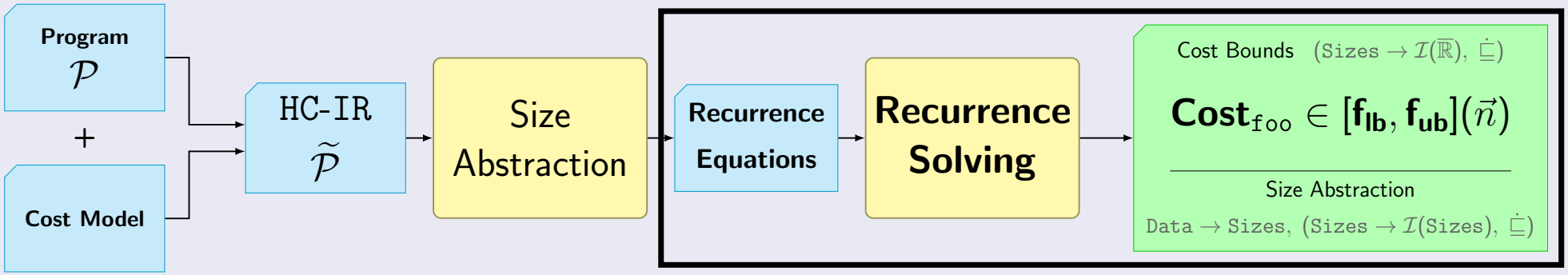
- Pipeline implemented in CiaoPP (and other analysers)



- Bottleneck: SotA in recurrence solving

Recurrence-Based Cost Analysis

- Pipeline implemented in CiaoPP (and other analysers)



- Bottleneck: SotA in recurrence solving

$$f(x) = \begin{cases} f(x+1)+1 & \text{if } x < 10 \\ 1 & \text{if } x \geq 10 \end{cases}$$

$f_{sol}(x) = \max(1, 11 - x)$

imperative loops

$$f(x,y) = \begin{cases} \max(f(x-1,y), f(x,y-1)) + 1 & \\ \quad \text{if } x > 0 \wedge y > 0 & \\ 0 \quad \text{if } x = 0 \vee y = 0 & \end{cases}$$

$$f_{sol}(x,y) = \begin{cases} x+y-1 & \text{if } x > 0 \wedge y > 0 \\ 0 & \text{if } x = 0 \vee y = 0 \end{cases}$$

conditionals

$$f_{(wc)}(n) = \begin{cases} 0 \quad \text{if } n \leq 2 \\ n + \max\limits_{1 \leq k \leq n-1} f(k) + f(n-k) & \\ \quad \text{if } n > 2 & \end{cases}$$

$f_{sol}(n) = ? \quad (\Theta(n^2))$

nondeterminism

$$f(n,c) = \begin{cases} f(n-1,0) + n + 300 & \\ \quad \text{if } n > 0 \wedge c \geq 100 & \\ f(n-1,c+1) + n & \\ \quad \text{if } n > 0 \wedge c < 100 & \\ c \quad \text{if } n = 0 & \end{cases}$$

$f_{sol}(n,c) = ite(c < 100,$
$n^2 + 199\lfloor \frac{n+c}{101} \rfloor + ..., \ ...)$

cost accidents

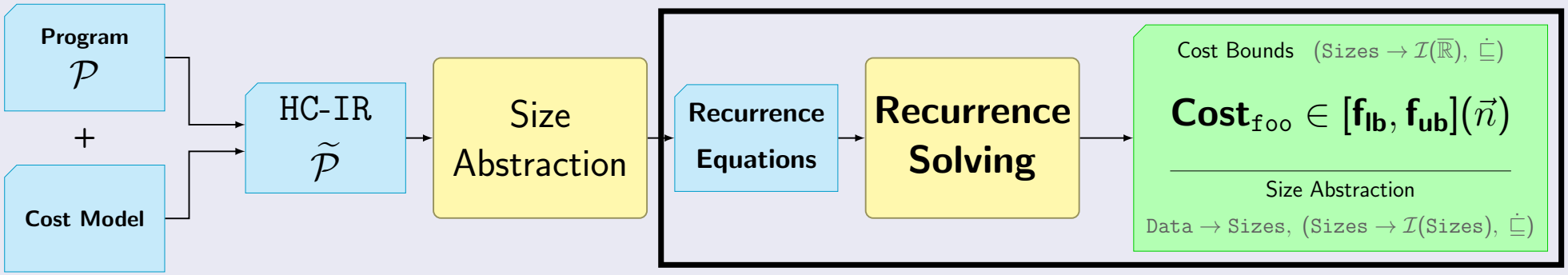
$$f(n) = \begin{cases} f(f(n-1)) + 1 & \text{if } n > 0 \\ 0 & \text{if } n = 0 \end{cases}$$

$f_{sol}(n) = n$

nested calls

Recurrence-Based Cost Analysis

- Pipeline implemented in CiaoPP (and other analysers)



- Bottleneck: SotA in recurrence solving

$$f(x) = \begin{cases} f(x+1)+1 & \text{if } x < 10 \\ 1 & \text{if } x \geq 10 \end{cases}$$

$f_{sol}(x) = \max(1, 11 - x)$

imperative loops

$$f(x,y) = \begin{cases} \max(f(x-1,y), f(x,y-1)) + 1 & \\ \quad \text{if } x > 0 \wedge y > 0 & \\ 0 \quad \text{if } x = 0 \vee y = 0 & \end{cases}$$

$$f_{sol}(x,y) = \begin{cases} x+y-1 & \text{if } x > 0 \wedge y > 0 \\ 0 & \text{if } x = 0 \vee y = 0 \end{cases}$$

conditionals

$$f_{(wc)}(n) = \begin{cases} 0 \quad \text{if } n \leq 2 \\ n + \max\limits_{1 \leq k \leq n-1} f(k) + f(n-k) & \\ \quad \text{if } n > 2 & \end{cases}$$

$f_{sol}(n) = ? \quad (\Theta(n^2))$

nondeterminism

$$f(n,c) = \begin{cases} f(n-1,0) + n + 300 & \\ \quad \text{if } n > 0 \wedge c \geq 100 & \\ f(n-1,c+1) + n & \\ \quad \text{if } n > 0 \wedge c < 100 & \\ c \quad \text{if } n = 0 & \end{cases}$$

$f_{sol}(n,c) = ite(c < 100,$
$n^2 + 199\lfloor \frac{n+c}{101} \rfloor + ..., ...)$

cost accidents

$$f(n) = \begin{cases} f(f(n-1)) + 1 & \text{if } n > 0 \\ 0 & \text{if } n = 0 \end{cases}$$

$f_{sol}(n) = n$

nested calls

- Applications beyond cost analysis

This Talk

Problem context

- Cost Analysis – Via (Generalised) Recurrence Equations
- Limitations of Existing Solvers

Our solution

- New framework/approaches to obtain (bounds on) solutions
- Order-theoretical and geometric viewpoint
- Dynamic/static cooperation

This Talk

Problem context

- Cost Analysis – Via (Generalised) Recurrence Equations
- Limitations of Existing Solvers

Our solution

- New framework/approaches to obtain (bounds on) solutions
- Order-theoretical and geometric viewpoint
- Dynamic/static cooperation
  - ➔ Tight, *non-linear* invariants on programs with *complex control flows* and *fine-grained cost models*

This Talk

Problem context

- Cost Analysis – Via (Generalised) Recurrence Equations
- Limitations of Existing Solvers

Our solution

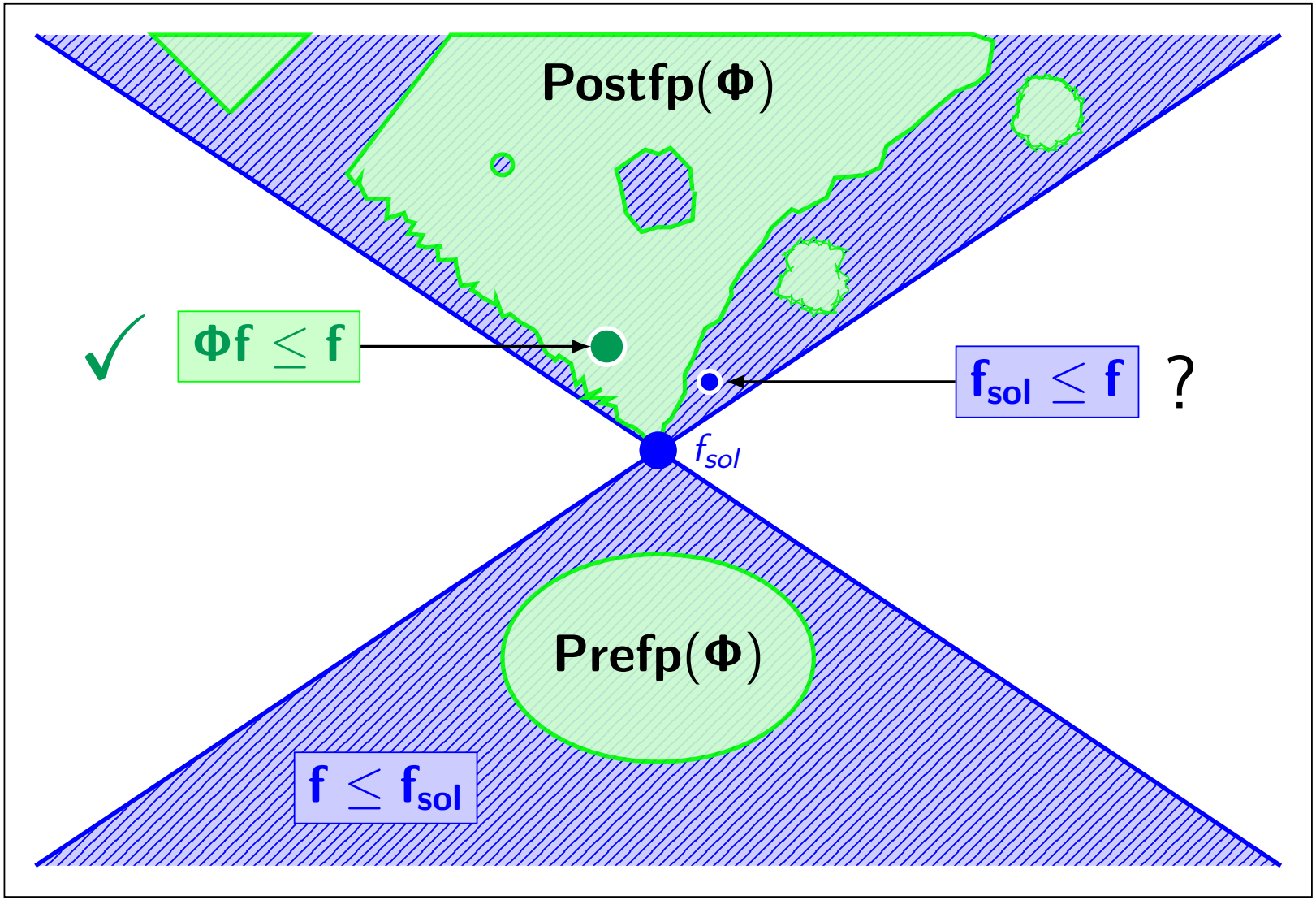
- New framework/approaches to obtain (bounds on) solutions
- Order-theoretical and geometric viewpoint
- Dynamic/static cooperation
  - ➔ Tight, *non-linear* invariants on programs with *complex control flows* and *fine-grained cost models*

Open the discussion

- Applications beyond cost analysis
- Sprinkle some take-home messages:
  general static analysis ideas, open to consideration

Where we are going

Equations $\leftrightarrow$ Operators Solutions $\leftrightarrow$ Fixpoints Bounds $\leftarrow$ Pre/Postfixp



Recurrence Equations for Non-Linear Inequality Invariants

Programs as Recurrence Equations

Recurrence Equations as { Invariants, Programs, Operators }

Recurrences: A Specialised Numerical Analysis Technique

Why study recurrences?

- Well-known methods produce very non-linear invariants
  - Logarithms, polynomials, exponentials, hypergeometric...
  - A challenge for classical numerical analyses

➔ **Deduce global non-linear invariants from local linear invariants**

Recurrences: A Specialised Numerical Analysis Technique

Why study recurrences?

- Well-known methods produce very non-linear invariants
  - Logarithms, polynomials, exponentials, hypergeometric...
  - A challenge for classical numerical analyses

➔ **Deduce global non-linear invariants from local linear invariants**

- Cost analysis may be viewed as a numerical analysis (`tick` variable), but a very specialised one, with particular control flows
  - *cost does not evolve like just another numerical variable*

Recurrences: A Specialised Numerical Analysis Technique

Why study recurrences?

- Well-known methods produce very non-linear invariants
  - Logarithms, polynomials, exponentials, hypergeometric...
  - A challenge for classical numerical analyses

➔ **Deduce global non-linear invariants from local linear invariants**

- Cost analysis may be viewed as a numerical analysis (`tick` variable), but a very specialised one, with particular control flows
  - *cost does not evolve like just another numerical variable*
  - Here recurrences applied to cost, but increasingly used elsewhere for non-linear invariants (summarisation of loops and beyond, etc.)
- Recurrence equations as abstractions of programs
- Maybe recurrences are for you!

Challenge and Opportunity

- Program Equations are not (should not be) classical CAS Recurrences

$$f(n) = a(n) \cdot f(\mathbf{n - 1}) + b(n)$$

complex factors, simple recursive calls

(classical computer algebra)

Challenge and Opportunity

- Program Equations are not (should not be) classical CAS Recurrences

$$f(n) = a(n) \cdot f(\mathbf{n - 1}) + b(n)$$

complex factors, simple recursive calls

(classical computer algebra)

$$f(\vec{n}) = \begin{cases} \dots \\ a \cdot f(\phi(\vec{n})) + b(\vec{n}) \\ \dots \end{cases}$$

simple factors, complex recursive calls
complex control flow
(program analysis)

Challenge and Opportunity – **Bounds!**

- Program Equations are not (should not be) classical CAS Recurrences

$$f(n) = a(n) \cdot f(\mathbf{n - 1}) + b(n)$$

complex factors, simple recursive calls

(classical computer algebra)

$$f(\vec{n}) = \begin{cases} \dots \\ a \cdot f(\phi(\vec{n})) + b(\vec{n}) \\ \dots \end{cases}$$

simple factors, complex recursive calls
complex control flow
(program analysis)

- Key insight: for us, **(good) bounds are good enough**
  $\rightarrow$ understudied problem

Dynamic/Static Cooperation (TPLP24)

- Specialised static cost analysers tackle very complex features
- But sometimes, problems look just too hard...

$$f(n) = \mathrm{ite}(n > 0,\ f(f(n-1)) + 1,\ 0)$$ **UNSUPPORTED**

Dynamic/Static Cooperation (TPLP24)

- Specialised static cost analysers tackle very complex features
- But sometimes, problems look just too hard...

$$f(n) = \mathrm{ite}(n > 0,\ f(f(n-1)) + 1,\ 0)$$ **UNSUPPORTED**

➔ Do not think too hard: observe first! (syntax vs semantics)

$$f_{sol}(0) = 0,\ f_{sol}(1) = 1,\ f_{sol}(7) = 7,\ f_{sol}(42) = 42...$$

- **Guess and Check:** infer "likely invariants" and attempt to prove them

$\hat{f} : n \mapsto n$,

(needs templates, optimisation methods, etc.)

Dynamic/Static Cooperation (TPLP24)

- Specialised static cost analysers tackle very complex features
- But sometimes, problems look just too hard...

$$f(n) = \mathrm{ite}(n > 0,\ f(f(n-1)) + 1,\ 0)$$ **UNSUPPORTED**

➔ Do not think too hard: observe first! (syntax vs semantics)

$$f_{sol}(0) = 0,\ f_{sol}(1) = 1,\ f_{sol}(7) = 7,\ f_{sol}(42) = 42...$$

- **Guess and Check:** infer "likely invariants" and attempt to prove them

$$\hat{f} : n \mapsto n, \quad \hat{f}(n) = \mathrm{ite}(n > 0,\ \hat{f}(\hat{f}(n-1)) + 1,\ 0), \quad \mathbf{f_{sol} = \hat{f}}\ \checkmark$$

(needs templates, optimisation methods, etc.)

- Hard problem, simple solution ✓

Dynamic/Static Cooperation (TPLP24)

- Specialised static cost analysers tackle very complex features
- But sometimes, problems look just too hard...

$$f(n) = \mathrm{ite}(n > 0,\ f(f(n-1)) + 1,\ 0)$$ **UNSUPPORTED**

➔ Do not think too hard: observe first! (syntax vs semantics)

$$f_{sol}(0) = 0,\ f_{sol}(1) = 1,\ f_{sol}(7) = 7,\ f_{sol}(42) = 42...$$

- **Guess and Check:** infer "likely invariants" and attempt to prove them

$$\hat{f} : n \mapsto n, \quad \hat{f}(n) = \mathrm{ite}(n > 0,\ \hat{f}(\hat{f}(n-1)) + 1,\ 0), \quad \mathbf{f_{sol} = \hat{f}}\ \checkmark$$

(needs templates, optimisation methods, etc.)

- Hard problem, simple solution ✓
- Hard problem, complex exact solution ×

Dynamic/Static Cooperation – **Bounds!**

- Specialised static cost analysers tackle very complex features
- But sometimes, problems look just too hard...

$$f(n) = \mathrm{ite}(n > 0,\ f(f(n-1)) + 1,\ 0)$$ **UNSUPPORTED**

➔ Do not think too hard: observe first! (syntax vs semantics)

$$f_{sol}(0) = 0,\ f_{sol}(1) = 1,\ f_{sol}(7) = 7,\ f_{sol}(42) = 42...$$

- **Guess and Check:** infer "likely invariants" and attempt to prove them

$$\hat{f} : n \mapsto n, \quad \hat{f}(n) = \mathrm{ite}(n > 0,\ \hat{f}(\hat{f}(n-1)) + 1,\ 0), \quad \mathbf{f_{sol} = \hat{f}}\ \checkmark$$

(needs templates, optimisation methods, etc.)

- Hard problem, simple solution ✓
- Hard problem, complex exact solution, **simple bounds ?**
- Advantages for guessing, but poses a new problem: **checking**

$$\mathbf{f_{sol} \leq \hat{f}} \quad \mathbf{??}$$

Bounds? Tarski saves the day: **Equations as Operators**

Equations $\leftrightarrow$ Operators Solutions $\leftrightarrow$ Fixpoints Bounds $\leftarrow$ Pre/Postfixp

Example

Search $f : \mathbb{N}^2 \to \mathbb{R}$ such that

$$f(n,c) = \begin{cases} f(n-1,0) + n + 300 & \text{if } n > 0 \text{ and } c \geq 100, \\ f(n-1,c+1) + n & \text{if } n > 0 \text{ and } c < 100, \\ c & \text{if } n = 0, \end{cases}$$

Example

Search $f : \mathbb{N}^2 \to \mathbb{R}$ such that $f = \Phi f$, i.e. $f \in \mathrm{Fixp}(\Phi)$, where

$\Phi : (\mathbb{N}^2 \to \mathbb{R}) \to (\mathbb{N}^2 \to \mathbb{R})$

$$f \mapsto (n,c) \mapsto \begin{cases} f(n-1,0) + n + 300 & \text{if } n > 0 \text{ and } c \geq 100, \\ f(n-1,c+1) + n & \text{if } n > 0 \text{ and } c < 100, \\ c & \text{if } n = 0. \end{cases}$$

Bounds? Tarski saves the day: **Equations as Operators**

Equations $\leftrightarrow$ Operators Solutions $\leftrightarrow$ Fixpoints Bounds $\leftarrow$ Pre/Postfixp

Example

Search $f : \mathbb{N}^2 \to \mathbb{R}$ such that $f = \Phi f$, i.e. $f \in \mathrm{Fixp}(\Phi)$, where

$\Phi : (\mathbb{N}^2 \to \mathbb{R}) \to (\mathbb{N}^2 \to \mathbb{R})$

$$f \mapsto (n, c) \mapsto \begin{cases} f(n-1, 0) + n + 300 & \text{if } n > 0 \text{ and } c \geq 100, \\ f(n-1, c+1) + n & \text{if } n > 0 \text{ and } c < 100, \\ c & \text{if } n = 0. \end{cases}$$

For a complete lattice, order $\mathbb{N}^2 \to \mathbb{R}$ **pointwise**, extend to $\overline{\mathbb{R}} := \mathbb{R} \cup \{\pm\infty\}$

Proposition (by Knaster-Tarski)

Let $\Phi : (\mathbb{N}^2 \to \overline{\mathbb{R}}) \to (\mathbb{N}^2 \to \overline{\mathbb{R}})$ be **monotone**, and $f_{sol} := \mathrm{lfp}\, \Phi$.
Then, for $\hat{f} : \mathbb{N}^2 \to \mathbb{R}$, we have $\hat{f} \in \mathrm{Postfp}(\Phi) \implies f_{sol} \leq \hat{f}$, i.e.

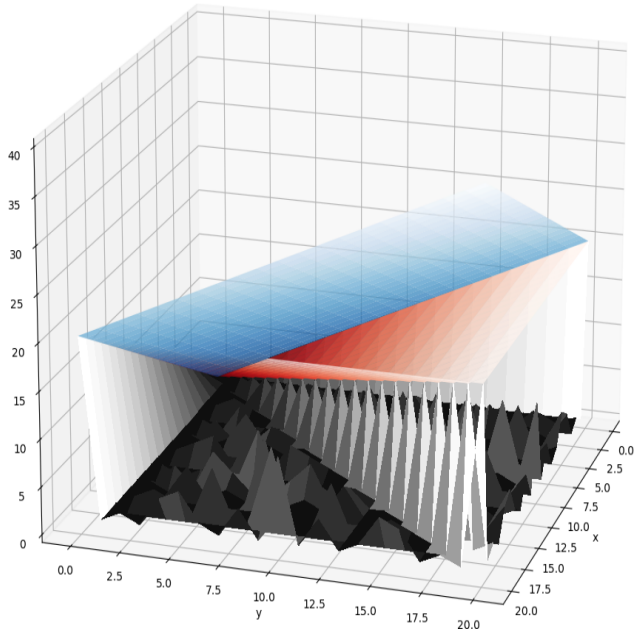
$$\Phi \hat{f} \leq \hat{f} \implies f_{sol} \leq \hat{f}.$$

Proof Technique – Equations as Operators viewpoint

Theorem

Let $\Phi : (\mathcal{D} \to \overline{\mathbb{R}}) \to (\mathcal{D} \to \overline{\mathbb{R}})$ *be a* **monotone equation**.

- *If* $f \in \mathrm{Postfp}(\Phi)$, *i.e.* $\Phi f \leq f$, *then* $\mathrm{lfp}\,\Phi \leq f$.
- *If* $f \in \mathrm{Prefp}(\Phi)$, *i.e.* $f \leq \Phi f$, *then* $f \leq \mathrm{gfp}\,\Phi$.



- $\hat{f}$
- $\Phi\hat{f}$
- f_{sol} (unknown)

Under termination hypotheses, $\mathrm{lfp}\,\Phi = \mathrm{gfp}\,\Phi \in (\mathcal{D} \to \mathbb{R})$, so (inductive) lower/upper bounds directly correspond to pre/postfixpoints.

Monotonicity

What can we handle? – Grammars of Equations

Monotone equations are common (even for $\dot{\leq}$)

Theorem

Let Φ *be a* **monotone** *equation.*
Then, $\Phi f \leq f \implies \operatorname{lfp} \Phi \leq f$.

Definitions

$(\mathcal{D} \to \overline{\mathbb{R}}, \leq)$. $(f \leq g) := \forall \vec{n},\ f(\vec{n}) \leq g(\vec{n})$
Φ monotone $:= \forall f, g,\ f \leq g \Rightarrow \Phi f \leq \Phi g$

Monotone equations are common (even for $\dot{\leq}$)

Theorem

Let Φ *be a* **monotone** *equation.*
Then, $\Phi f \leq f \implies \operatorname{lfp} \Phi \leq f$.

Definitions

$(\mathcal{D} \to \overline{\mathbb{R}}, \leq)$. $(f \leq g) := \forall \vec{n},\ f(\vec{n}) \leq g(\vec{n})$
Φ monotone $:= \forall f, g,\ f \leq g \Rightarrow \Phi f \leq \Phi g$

Example (template)

$$\Phi : f \mapsto \vec{n} \mapsto \begin{cases} \cdots \\ \sum_{j=1}^{k_i} \left(a_{i,j}(\vec{n}) \cdot f(\phi_{i,j}(\vec{n}))\right) + b_i(\vec{n}) & \text{if } \varphi_i(\vec{n}) \\ \cdots \end{cases}$$

where $a_{i,j} \geq 0$, and $\phi_{i,j}, b_i, \varphi_i$ arbitrary.

Example (grammar)

$$\frac{P : \mathcal{D} \to \mathbb{R}}{(f \mapsto \vec{n} \mapsto P(\vec{n})) : \texttt{Eqs}}\ (\text{Base-Case}) \qquad \frac{\phi : \mathcal{D} \to \mathcal{D}}{(f \mapsto f \circ \phi) : \texttt{Eqs}}\ (\text{Rec-Call})$$

$$\frac{\Phi_1 : \texttt{Eqs},\ \Phi_2 : \texttt{Eqs}}{(f \mapsto \Phi_1(f) + \Phi_2(f)) : \texttt{Eqs}}\ (\text{Sum}) \qquad \frac{\Phi : \texttt{Eqs},\ a : \mathcal{D} \to \mathbb{R}_+}{(f \mapsto \vec{n} \mapsto a(\vec{n}) \cdot \Phi(f)(\vec{n})) : \texttt{Eqs}}\ (\text{Scale})$$

$$\frac{\Phi_1 : \texttt{Eqs},\ \Phi_2 : \texttt{Eqs},\ \varphi : \mathcal{D} \to \mathbb{B}}{\left(f \mapsto \text{ite}(\varphi, \Phi_1(f), \Phi_2(f))\right) : \texttt{Eqs}}\ (\text{Choice}) \qquad \frac{\Phi_1 : \texttt{Eqs},\ \Phi_2 : \texttt{Eqs}}{\Phi_2 \circ \Phi_1 : \texttt{Eqs}}\ (\text{Comp})$$

$$\frac{\Phi_1 : \texttt{Eqs},\ \Phi_2 : \texttt{Eqs}}{\left(f \mapsto \max\left(\Phi_1(f), \Phi_2(f)\right)\right) : \texttt{Eqs}}\ (\text{Max}) \qquad \frac{\Phi_1 : \texttt{Eqs},\ \Phi_2 : \texttt{Eqs}}{\left(f \mapsto \min\left(\Phi_1(f), \Phi_2(f)\right)\right) : \texttt{Eqs}}\ (\text{Min})$$

Example (unbounded optimisation)

Non-deterministic quicksort

$$\Phi : f \mapsto n \mapsto \begin{cases} 0 & \text{if } n \leq 2 \\ n + \max_{1 \leq k \leq n-1} f(k) + f(n-k) & \text{if } n > 2 \end{cases}$$

Monotone equations are common (even for $\dot{\leq}$)

Theorem

Let Φ *be a* **monotone** *equation.*
Then, $\Phi f \leq f \implies \operatorname{lfp} \Phi \leq f$.

Definitions

$(\mathcal{D} \to \overline{\mathbb{R}}, \leq)$. $(f \leq g) := \forall \vec{n},\ f(\vec{n}) \leq g(\vec{n})$
Φ monotone $:= \forall f, g,\ f \leq g \Rightarrow \Phi f \leq \Phi g$

Example (template)

$$\Phi : f \mapsto \vec{n} \mapsto \begin{cases} \cdots \\ \sum_{j=1}^{k_i} \left(a_{i,j}(\vec{n}) \cdot f(\phi_{i,j}(\vec{n}))\right) + b_i(\vec{n}) & \text{if } \varphi_i(\vec{n}) \\ \cdots \end{cases}$$

where $a_{i,j} \geq 0$, and $\phi_{i,j}, b_i, \varphi_i$ arbitrary.

Example (grammar)

$$\frac{P : \mathcal{D} \to \mathbb{R}}{(f \mapsto \vec{n} \mapsto P(\vec{n})) : \texttt{Eqs}}\ (\text{Base-Case}) \qquad \frac{\phi : \mathcal{D} \to \mathcal{D}}{(f \mapsto f \circ \phi) : \texttt{Eqs}}\ (\text{Rec-Call})$$

$$\frac{\Phi_1 : \texttt{Eqs},\ \Phi_2 : \texttt{Eqs}}{(f \mapsto \Phi_1(f) + \Phi_2(f)) : \texttt{Eqs}}\ (\text{Sum}) \qquad \frac{\Phi : \texttt{Eqs},\ a : \mathcal{D} \to \mathbb{R}_+}{(f \mapsto \vec{n} \mapsto a(\vec{n}) \cdot \Phi(f)(\vec{n})) : \texttt{Eqs}}\ (\text{Scale})$$

$$\frac{\Phi_1 : \texttt{Eqs},\ \Phi_2 : \texttt{Eqs},\ \varphi : \mathcal{D} \to \mathbb{B}}{\left(f \mapsto \text{ite}(\varphi, \Phi_1(f), \Phi_2(f))\right) : \texttt{Eqs}}\ (\text{Choice}) \qquad \frac{\Phi_1 : \texttt{Eqs},\ \Phi_2 : \texttt{Eqs}}{\Phi_2 \circ \Phi_1 : \texttt{Eqs}}\ (\text{Comp})$$

$$\frac{\Phi_1 : \texttt{Eqs},\ \Phi_2 : \texttt{Eqs}}{\left(f \mapsto \max\left(\Phi_1(f), \Phi_2(f)\right)\right) : \texttt{Eqs}}\ (\text{Max}) \qquad \frac{\Phi_1 : \texttt{Eqs},\ \Phi_2 : \texttt{Eqs}}{\left(f \mapsto \min\left(\Phi_1(f), \Phi_2(f)\right)\right) : \texttt{Eqs}}\ (\text{Min})$$

Example (unbounded optimisation)

Non-deterministic quicksort

$$\Phi : f \mapsto n \mapsto \begin{cases} 0 & \text{if } n \leq 2 \\ n + \max_{1 \leq k \leq n-1} f(k) + f(n-k) & \text{if } n > 2 \end{cases}$$

Counter-example

$$\Phi : f \mapsto n \mapsto \begin{cases} -\mathbf{1} \cdot f(n-1) & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$$

(needs antimonotonicity)

Monotone Equations $\neq$ Monotone Solutions/Resources

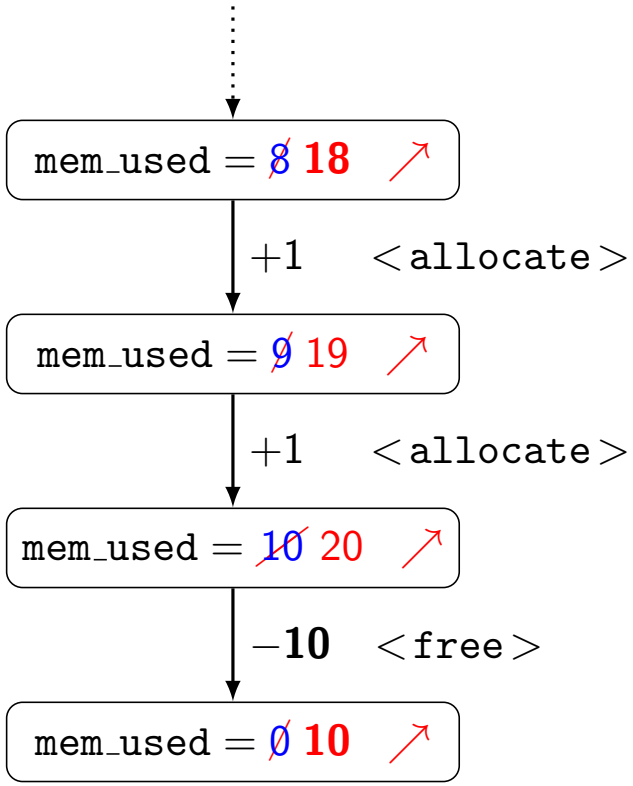
- We can handle "non-monotone resources" (e.g. memory)
- We can handle "non-monotone candidates" (e.g. in `while` loops)
- Monotone Φ reflect "accumulativeness" of cost along control flow (both $+$ and $-$)

Equation monotonicity:

$$f \leq g \implies \Phi f \leq \Phi g$$

"if cost of a recursive call had been higher, final cost would have been higher"

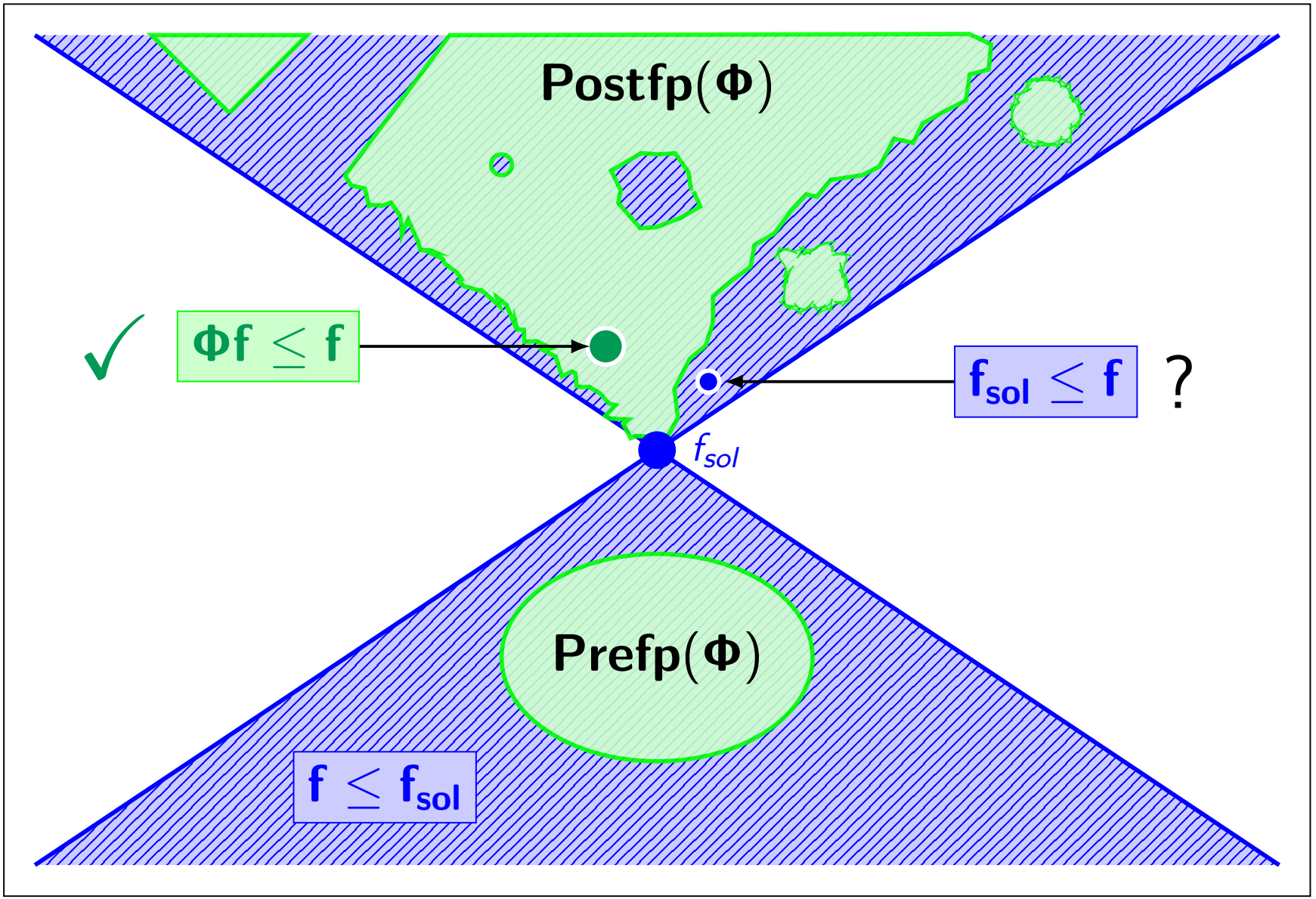
Consequence: $\Phi f \leq f \implies f_{sol} \leq f$



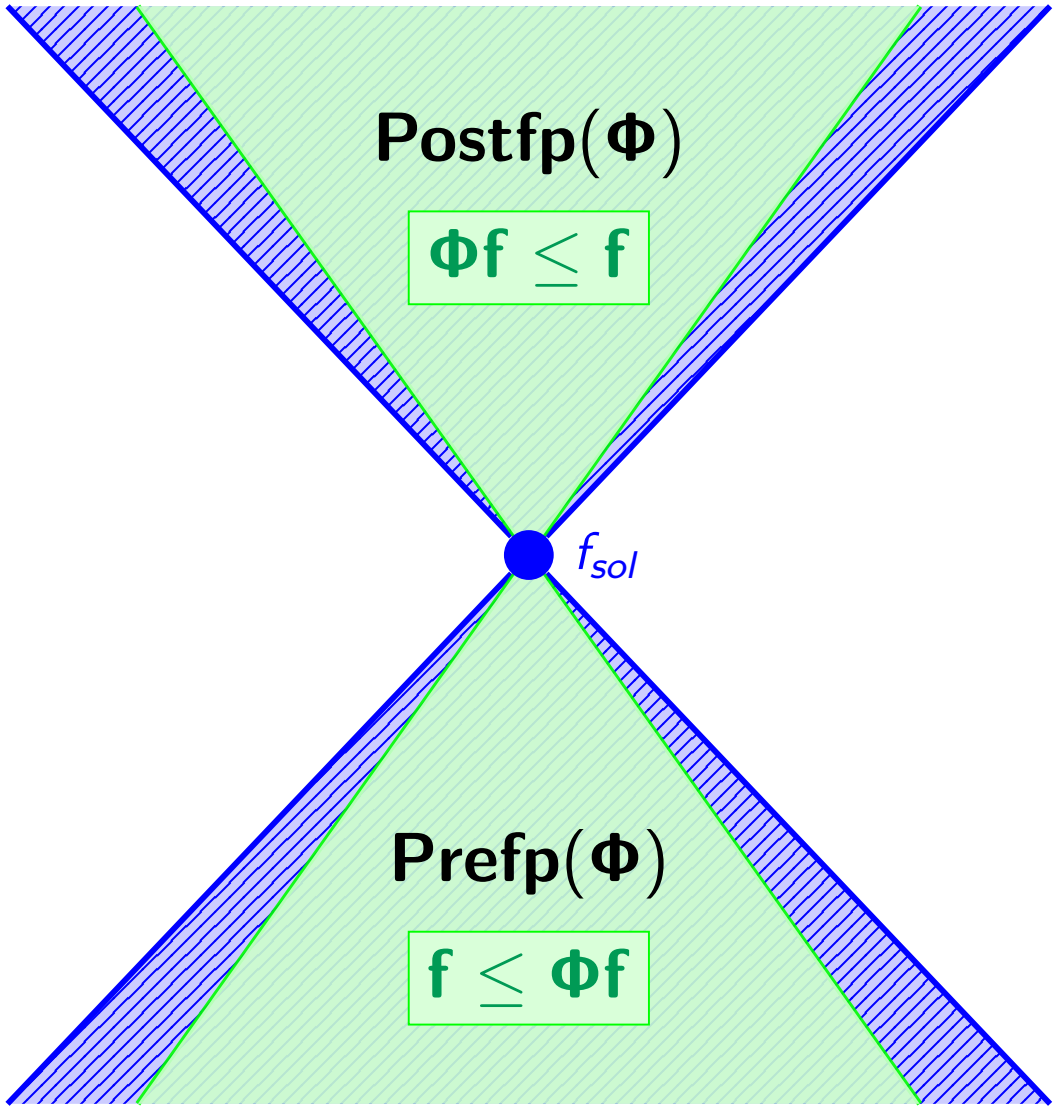
Geometry of Postfp and Prefp

Where are inductive bounds, and how good are they?

Where are we?

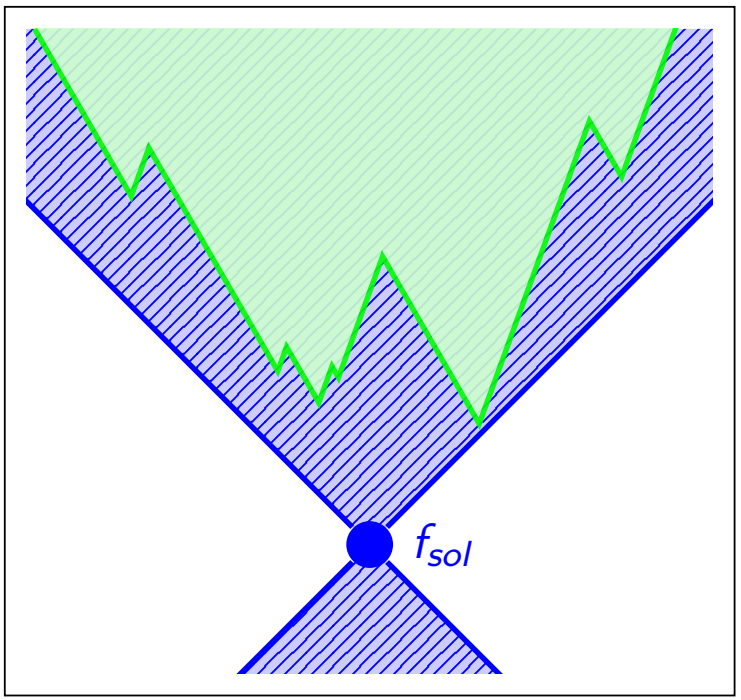


Geometry of $\mathrm{Postfp}(\Phi)$ — (more in the paper)

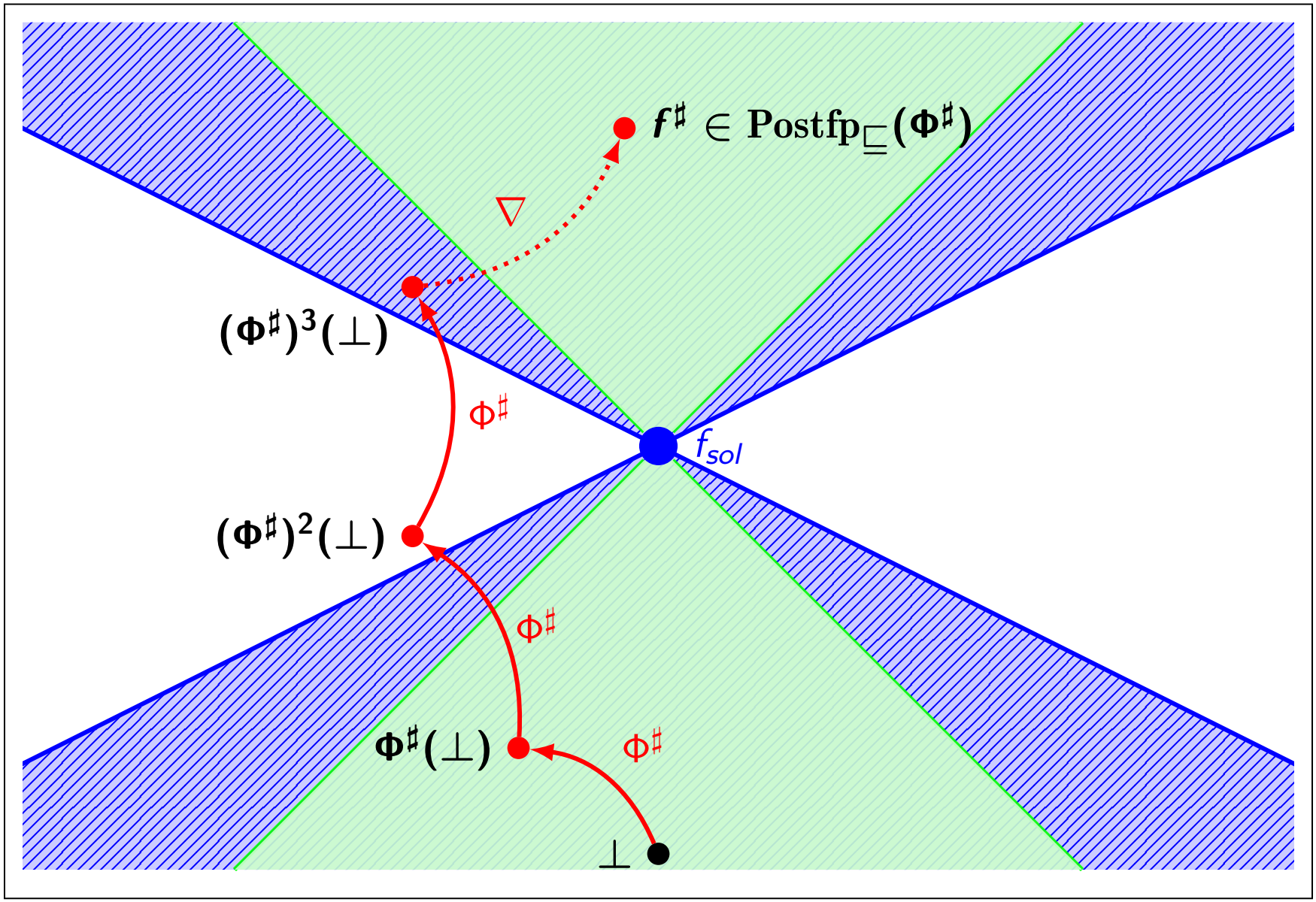


- Descriptions of $\mathrm{Postfp}(\Phi)$, $\mathrm{Prefp}(\Phi)$ for families of Φ. Internal/External
- **Cones** or beyond, "Icebergs"
- Rkfun, discrete derivatives

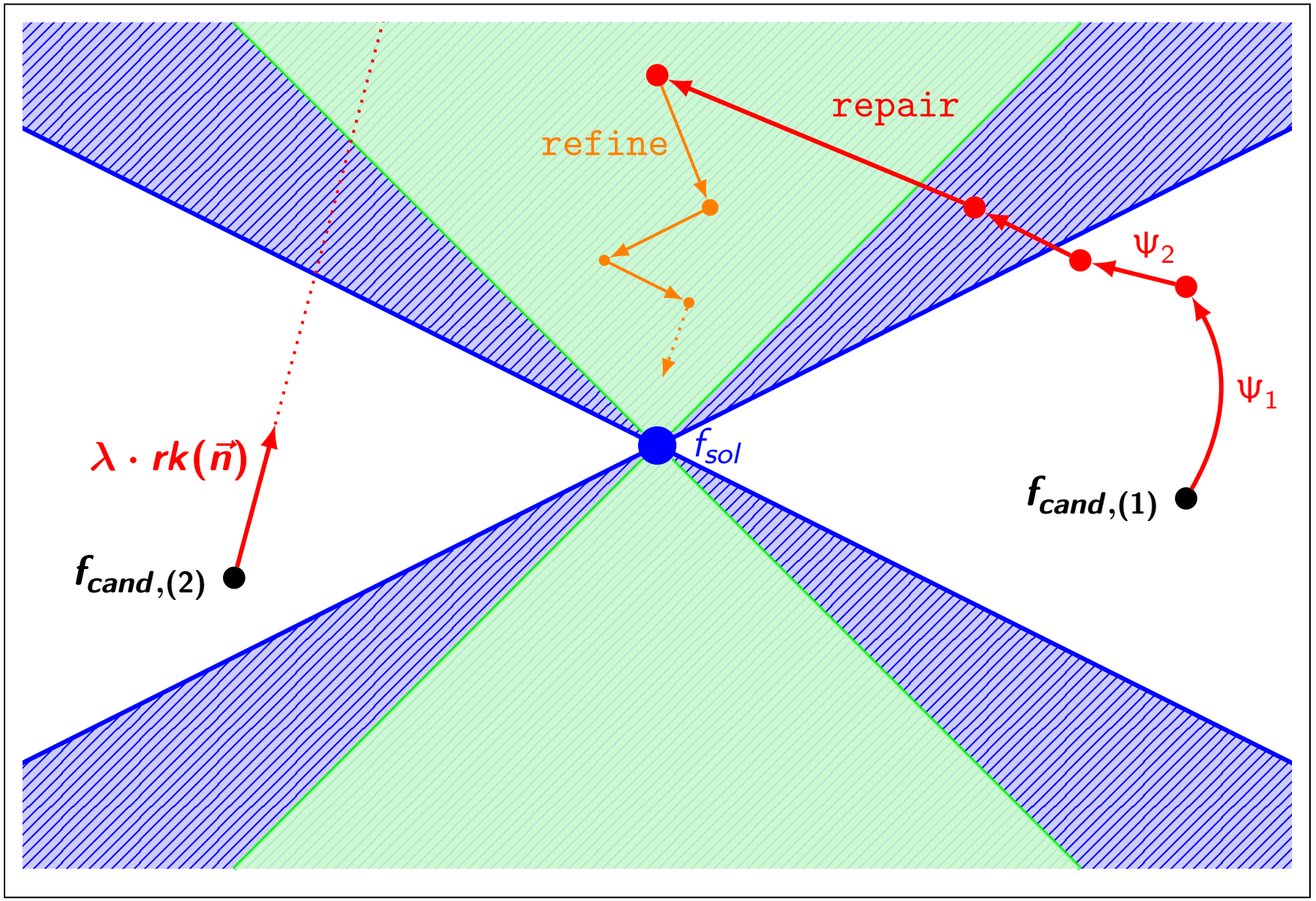
➔ More in the paper



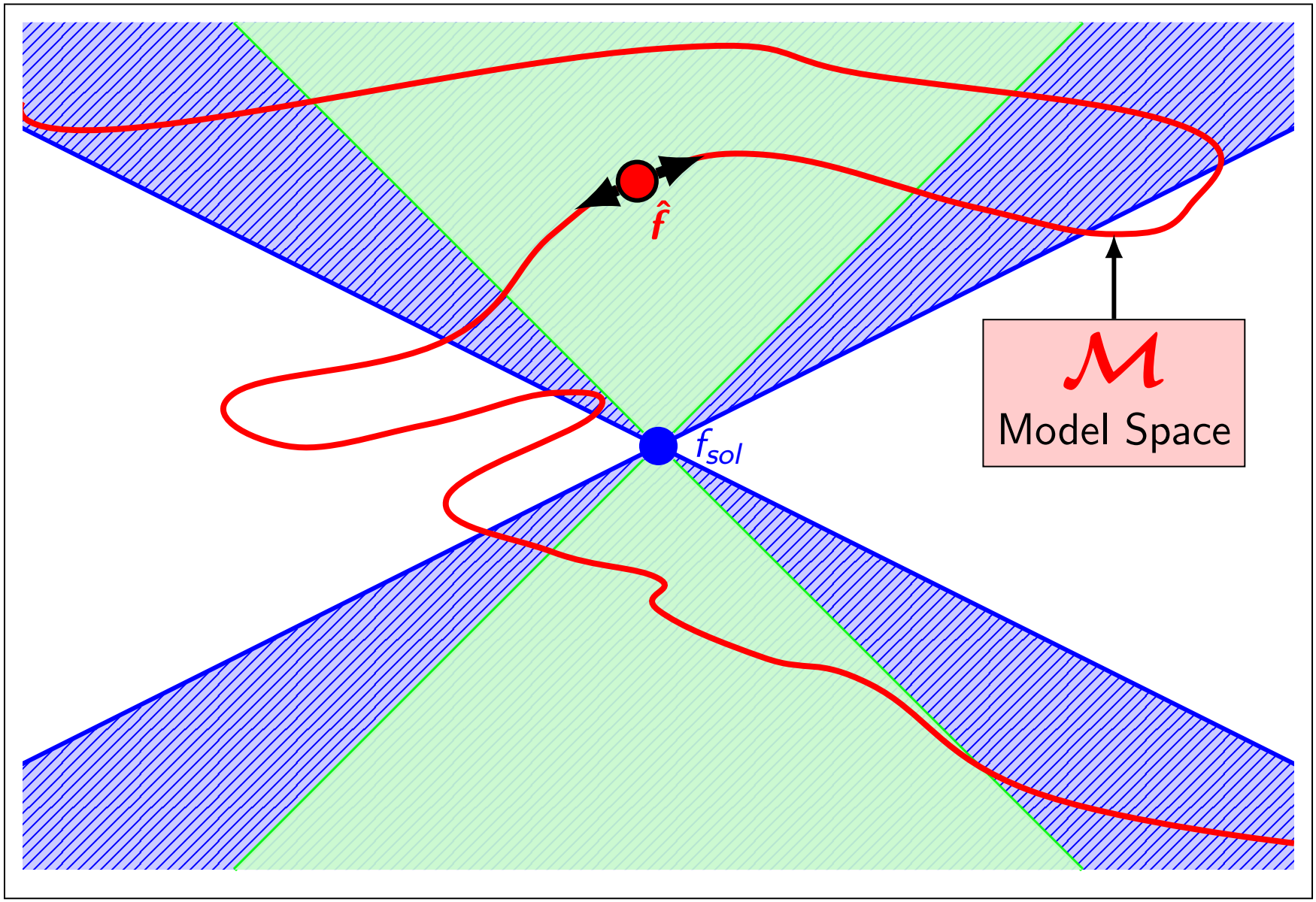
Exploration (1) – Kleene Sequences, Abstract Iteration



Exploration (2) – Repair/Refine Expression Transformation



Exploration (3) – Templates, Search, Optimisation ($\forall$-elim, etc.)



Constrained-optimisation-based Instantiation

A cooperation of dynamic and static techniques

A simple instantiation, for experiments

Finite-dimensional, affine

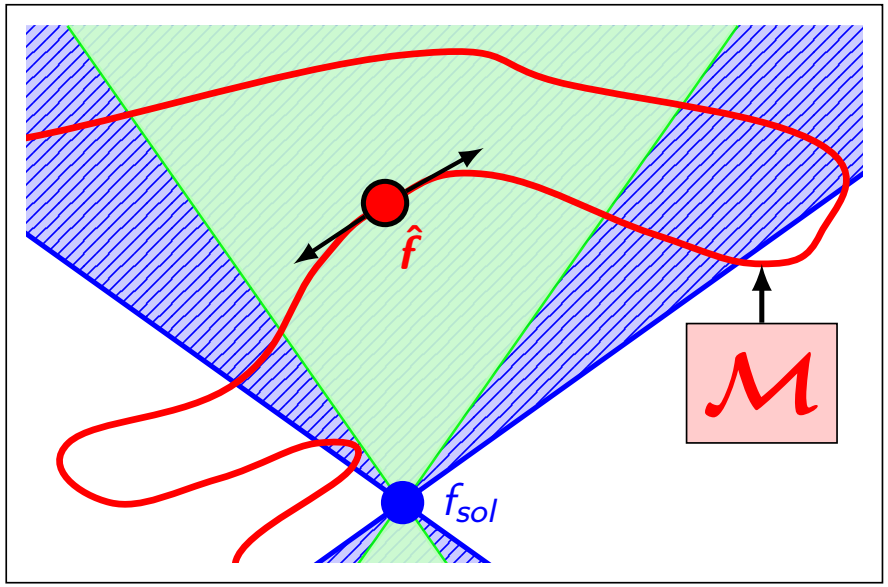
- Model space $\mathcal{M}$ (features)
  $\hat{f} = f_{\vec{\alpha}} := f_{base} + \sum_{i=1}^{p} \alpha_i f_i, \quad f_i \in \mathcal{F}$
- Observation space $\mathcal{I}_{sample} \to \mathbb{R}$
  $\{f_{sol}(\vec{n})\}, \{(\Phi f_i - f_i)(\vec{n})\}, \quad \vec{n} \in \mathcal{I}$

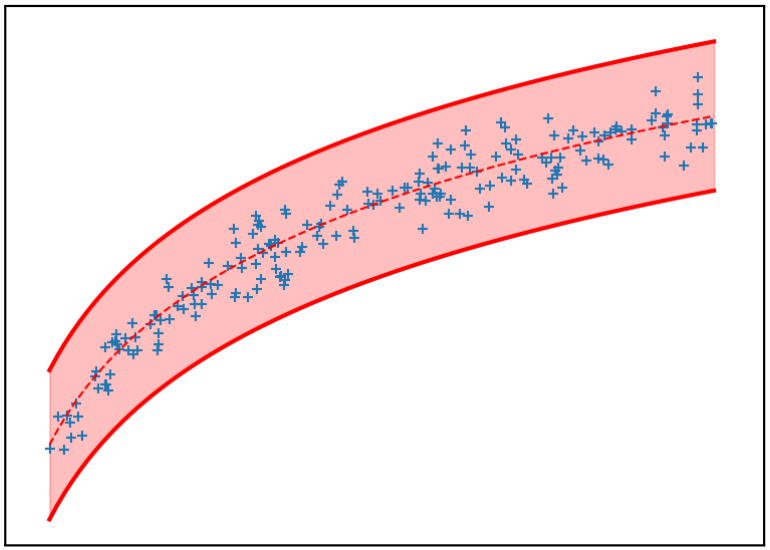
Optimisation under constraints

- Local Safety $f_{sol}(\vec{n}) \leq f_{\vec{\alpha}}(\vec{n})$
- **Local Inductivity** $(\Phi(f_{\vec{\alpha}}) - f_{\vec{\alpha}})(\vec{n}) \leq \mathbf{0}$
  (easy for affine Φ)
- Accuracy $||(\hat{f} - f_{sol})_{|\mathcal{I}}||_2^2$,
  Parsimony $||\vec{\alpha}||_1$

Reduces to QP problem on $\vec{\alpha}$

$$\min_{\vec{\alpha} \in \mathbb{R}^p} ||\vec{y_{sol}} - \mathcal{Y}\vec{\alpha}||_2^2 + \lambda||\vec{\alpha}||_1,$$
$$\vec{y_{sol}} \leq \mathcal{Y}\vec{\alpha} \text{ and } \mathcal{Y}'\vec{\alpha} + \vec{b'} \leq \mathcal{Y}\vec{\alpha}$$





Prototype Solver (`order_recsolv`), Python input/output

$$\Phi(f)(n,c) = \begin{cases} f(n-1,0)+n+300 & \text{if } n>0 \wedge c \geq 100 \\ f(n-1,c+1)+n & \text{if } n>0 \wedge c<100 \\ c & \text{if } n=0 \end{cases}$$

```
def f(n,c):
  if n>0 and c>=100:
    return f(n-1, 0) + n + 300
  if n>0 and c<100:
    return f(n-1, c+1) + n
  return c
```

$$f_{sol}(n,c) \leq \begin{cases} \frac{1}{2}n^2 + \frac{701}{202}n + \frac{30000}{101} & \text{if } n>0 \wedge c \geq 100 \\ \frac{1}{2}n^2 + \frac{701}{202}n + \frac{300}{101}c & \text{if } n>0 \wedge c<100 \\ c & \text{if } n=0 \end{cases}$$

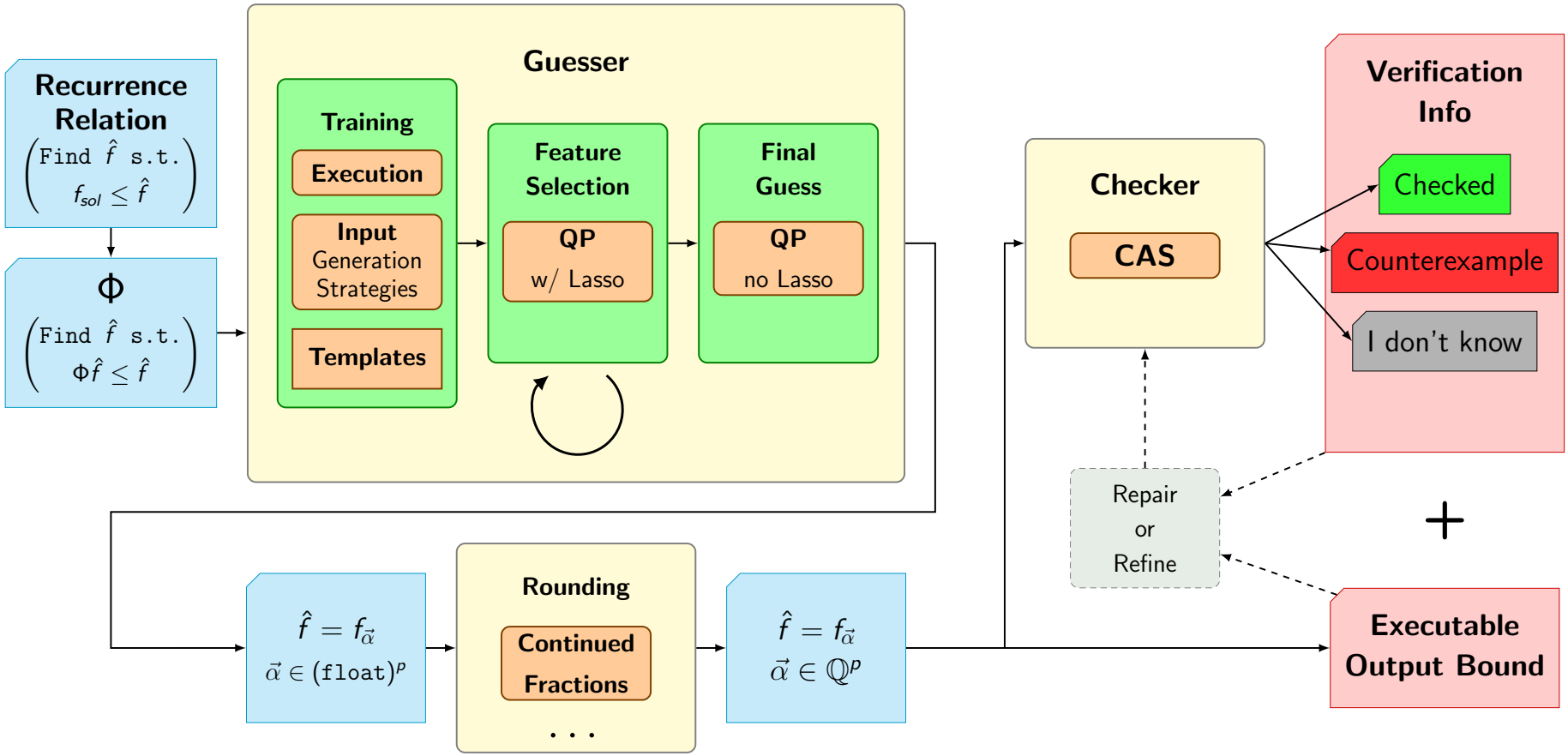
```
def f_ub(n, c):
  if ((n > 0) and (c >= 100)):
    return 1/2*n*n + 701/202*n + 30000/101*1
  elif ((n > 0) and (c < 100)):
    return 1/2*n*n + 701/202*n + 300/101*c
  else:
    return c

# ========================================
# Result: {}.
# Candidate checked (no counter-examples).
# ========================================

def f_lb(n, c):
  if ((n > 0) and (c >= 100)):
    return 1/2*n*n + 701/202*n + 100*1
  elif ((n > 0) and (c < 100)):
    return 1/2*n*n + 701/202*n + 300/101*c \
           + -19900/101*1
  else:
    return c

# ========================================
# Result: {}.
# Candidate checked (no counter-examples).
# ========================================
```

Solver Architecture (`order_recsolv`)



Experimental Evaluation

- Experiments on 21 benchmarks, arranged in 5 categories, coming from programs with a diverse set of features
- Compared with `mlsolve` (previous work), `RaML`, `Cofloco` and `Mathematica`

➔ Promising results, with non-linear bounds, reasonable guess/check times `[9.05,47.5]s`

- More extensive comparison in our previous work with equality only (TPLP24)

  `mlsolve(L)`, `mlsolve(S)`, `Ciao's builtin`, `RaML`, `PUBS`, `Cofloco`, `KoAT`, `Duet`, `Loopus`, `PRS23's solver`, `Sympy`, `PURRS`, `Mathematica`, ...

	Bench	Prototype	mlsolve	RaML	Cofloco	Mathematica
with_noise	1	[∼, ∼]	×	[∼, O]	[Ω, Θ]	None
	2	[∼, ∼]	×	[Θ, None]	[Ω, Θ]	None
	3	$f_{sol} \pm \Theta(1)$	×	[None, O]	[×, O]	None
	4	[∼, ∼]	×	[None, None]	[None, None]	None
nonmonot	5	[$\tilde{\checkmark}$, $\tilde{\checkmark}$]	×	[Θ, None]	[✓, ✓]	None
	6	[$\tilde{\checkmark}$, $\tilde{\checkmark}$]	✓	[None, ∼]	[✓, ✓]	∼ / ×
	7	[$\tilde{\checkmark}$, $\tilde{\checkmark}$]	✓	[None, None]	[✓, ✓]	None
	8	[✓, ✓]	✓	[None, None]	[✓, ✓]	None
divconq	9	[Θ, ∼]	×	[None, O]	[None, None]	None
	10	[≈, ∼]	×	[None, O]	[None, None]	None
	11	[Ω, O]	×	[Ω, O]	[None, None]	None
s-poly	12	[✓, ✓]	✓	[Ω, None]	[None, None]	✓
	13	[✓, ✓]	✓	[Ω, None]	[None, None]	✓
	14	[✓, ✓]	✓	[Ω, None]	None	✓
misc	15	[✓, ✓]	✓	[✓, ✓]	[✓, ✓]	✓
	16	[✓, ✓]	✓	[×, O]	[×, O]	None
	17	[Ω, O]	×	[None, None]	[None, O]	None
	18	[✓, ✓]	✓	[✓, ✓]	[None, None]	None
	19	[Θ, ×]	×	[Ω, Θ]	[✓, ✓]	None
	20	[$\tilde{\checkmark}$, $\tilde{\checkmark}$]	✓	[None, ∼]	[Ω, ✓]	None
	21	[✓, ✓]	✓	[None, ✓]	[Θ, ✓]	None

➔ See Artifact

Conclusion

Conclusions

- Safe approximate numerical equation solving:
  Relevant to static analysis, much to explore
- Order-theoretical viewpoint on equations – Pointwise order $(\mathcal{D} \to \overline{\mathbb{R}}, \dot{\leq})$ from $\leq_{\mathbb{R}}$
- Simple instantiations already enough to improve the state-of-the-art,
  and open the path to better static cost analysis

Open to consideration

- Numerical Static Analysis Beyond Linear Properties
  ➔ Recurrence Equations
- Postfixpoint Search
  ➔ Not just Kleene sequences
- Cooperation between Static and Dynamic techniques
  ➔ Role of Optimisation in Static Analysis/Postfixpoint Search
  ➔ Improve Model Generalisability by improving Verifiability

Future work

- Size abstractions leveraging our new solving techniques
- New (co)domains for new applications (cyberphysical or biochemical systems, etc.)

Thank you for your attention!