Towards an Automated Reasoning Tool for Complexity Analysis of Automated Reasoners

Louis Rustenholz[1,3], Manuel V. Hermenegildo[1,3], Pedro López-García[2,3], Alessio Mansutti[3], Félix Ridoux[4,5], and Niki Vazou[3]

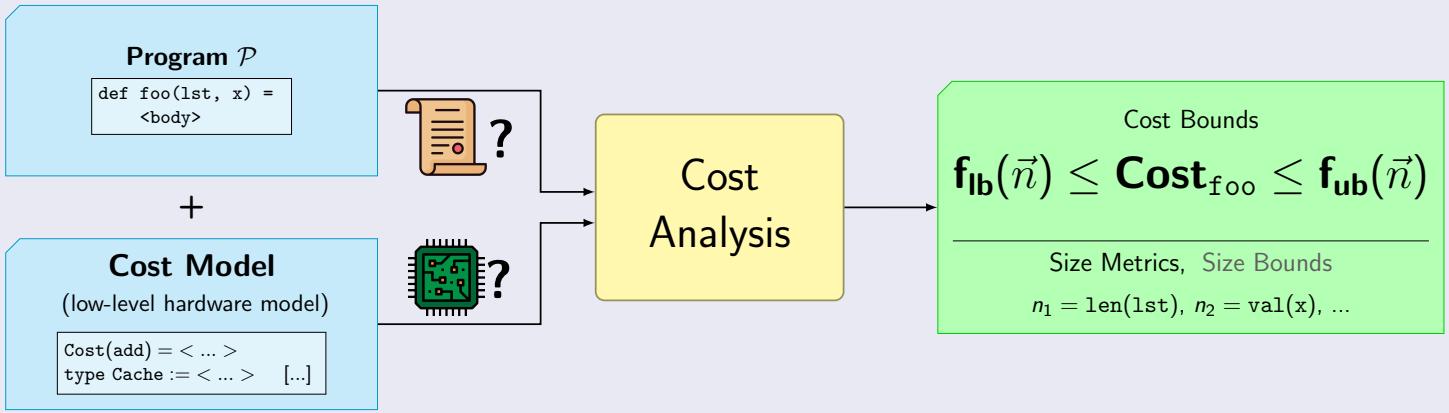
[1]Universidad Politécnica de Madrid (UPM), Spain
[2]Spanish Council for Scientific Research (CSIC), Spain
[3]IMDEA Software Institute, Spain
[4]Computer Science Laboratory of Sorbonne University (LIP6), France
[5]STMicroelectronics, Crolles, France

WST, July 25th, 2026
FLoC26, Lisboa, Portugal

Introduction

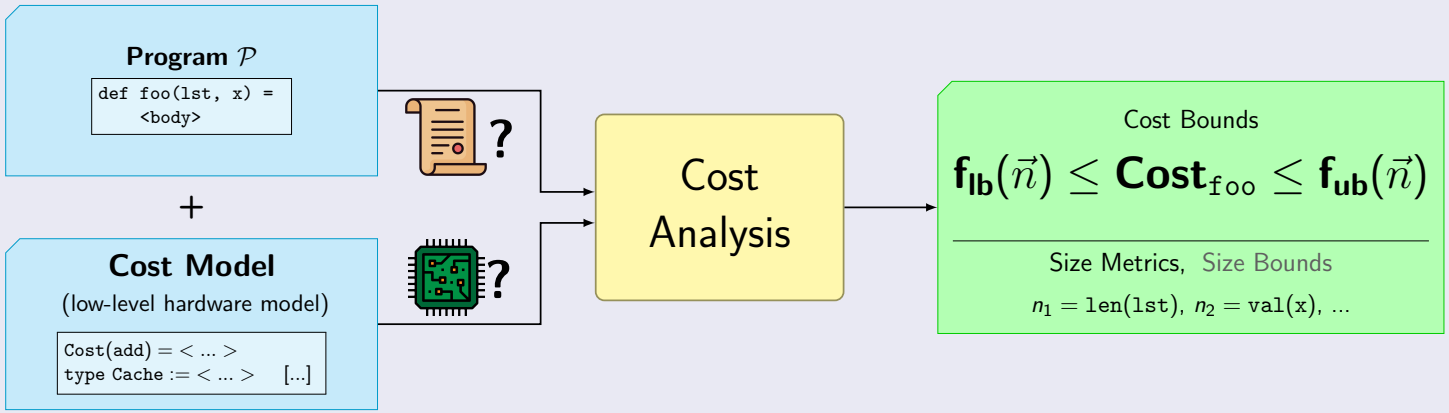
Automated Recurrence-Based Cost Analysis – Usual Setup

Cost Analysis: Bounds on Resource Consumption

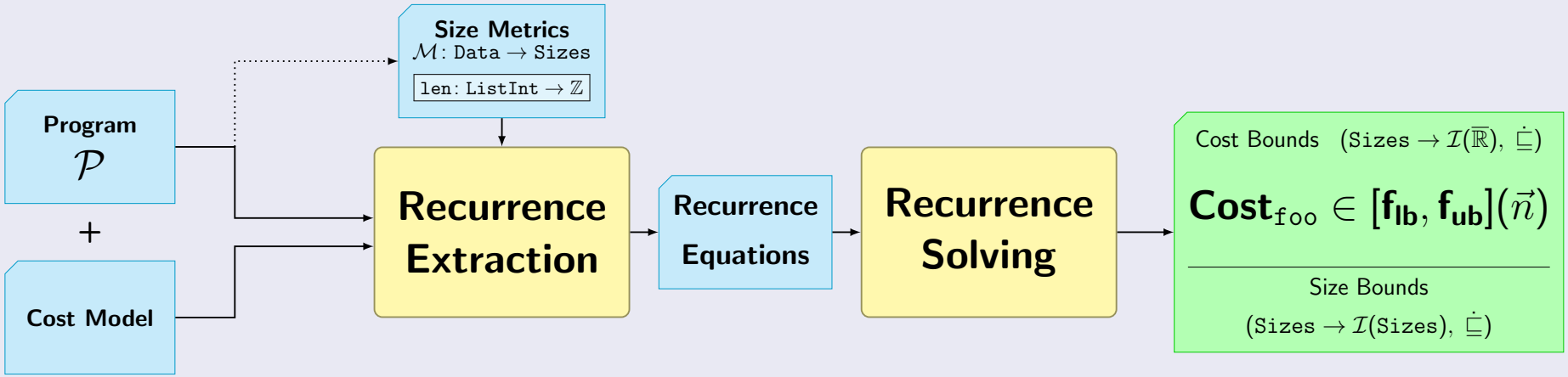


Automated Recurrence-Based Cost Analysis – Usual Setup

Cost Analysis: Bounds on Resource Consumption



Recurrence-Based Cost Analysis



Project Genesis: Manual Complexity Analysis for Arithmetic Theories

Study of $\mathcal{P}a(2^{\mathbb{N}}(\cdot)) := \langle \mathbb{Z}, 1, +, (a \cdot x)_{a\in\mathbb{Z}}, (q \mid x)_{q\in\mathbb{N}_+}, \underline{2^{\mathbb{N}}(x)}, < \rangle$.[1]

Theorem

Quantifier elimination for $\mathcal{P}a(2^{\mathbb{N}}(\cdot))$ *is in* 3ExpTime.

Proof: An algorithm, and its complexity analysis:
metrics, recurrence extraction, bounds on solutions.

Parameters of formulae. As often done for $\mathcal{Pa}$, the complexity analysis of our procedure requires the introduction of several parameters for a formula. We define functions $lin(\cdot)$, $hom(\cdot)$, $heft(\cdot)$, $mod(\cdot)$, $\mathcal{B}(\cdot)$, and $alt(\cdot)$, to track various features of a formula Φ from $\mathcal{Pa}(\lambda x.2^{|x|})$:

- $lin(\Phi)$ is the set containing the terms 0 and 2 as well as all the terms t that appear in inequalities $t < 0$ of Φ (recall that $t_1 < t_2$ is a shorthand for $t_1 - t_2 < 0$);
- $hom(\Phi)$ is the set of *homogeneous terms* obtained from the terms in $lin(\Phi)$ by eliminating their constant term c (alternatively, updating c to 0);
- $heft(\Phi)$ is the maximum number of variables in a term of Φ;
- $mod(\Phi)$ is the least common multiple of all $q \in \mathbb{N}_+$ appearing in constraints $q \mid t$ of Φ (if the formula Φ has no divisibility constraints, then we postulate $mod(\Phi) = 1$);
- $\mathcal{B}(\Phi)$ denotes the number of occurrences of negations $\neg$ and conjunctions $\wedge$ in Φ (note that the syntax permits binary conjunction only);
- $alt(\Phi)$ is the *quantifier alternation rank* (number of quantifier blocks) of a formula Φ in prenex normal form.

[1] *The Complexity of Presburger Arithmetic with Power or Powers*. M. Benedikt, D. Chistikov, A. Mansutti, ICALP23.

Project Genesis: Manual Complexity Analysis for Arithmetic Theories

Study of $\mathcal{P}a(2^{\mathbb{N}}(\cdot)) := \langle \mathbb{Z}, 1, +, (a \cdot x)_{a\in\mathbb{Z}}, (q \mid x)_{q\in\mathbb{N}_+}, \underline{2^{\mathbb{N}}(x)}, < \rangle$.[1]

Theorem

Quantifier elimination for $\mathcal{P}a(2^{\mathbb{N}}(\cdot))$ is in 3ExpTime.

Proof: An algorithm, and its complexity analysis:
metrics, recurrence extraction, bounds on solutions.

B.2 Towards a proof of Theorem 2: PresQE and Octagons

We start by deriving bounds on iterated calls to PresQE.

▶ **Proposition 37.** *Consider $\Psi(\boldsymbol{x}, \boldsymbol{z})$ in $\mathcal{Sem}$ in which variables $\boldsymbol{x} = (x_1, \ldots, x_n)$ always occur linearly, and $n \geq 1$. Let $\mathcal{F} \in \{\mathcal{QF}, \mathcal{Sem}\}$. Run Algorithm 1 from line 3 with $Q = \{(\boldsymbol{x}, \Psi)\}$. After the* **while** *loop in lines 5–10 terminates, $D = \{\psi_1, \ldots, \psi_k\}$ satisfies the parameter table*

	$\#hom$	$\#lin$	$heft$	$\|hom(\cdot)\|$	$\|lin(\cdot)\|$	mod	$\mathcal{B}$
Ψ	$h \geq 2$	q	v	$a \geq 2$	$c \geq 2$	$m \geq 2$	b
ψ_i	h	q	V	$a^{2^{n+1}-1}$	$a^{2^{2n+1}}(c+m)$	$a^{2^{n+2}-n}m$	$b + n \cdot (2 \cdot V + 1)$
$\bigvee_{j=1}^{k} \psi_j$	h^{n+1}		$''$	$''$	$''$	$a^{h^{2n+2}}m$	$k(''+1)$

where $V := \min(2^n v, n + \#\boldsymbol{z})$ and $k \leq q^n \cdot (a^{2^{n+2}} \cdot m)^{n(2V+1)}$. Moreover, the algorithm runs in time $q^{O(n)} m^{O(n \cdot V)} \cdot a^{2^{O(n)} v}$.

[1] *The Complexity of Presburger Arithmetic with Power or Powers.* M. Benedikt, D. Chistikov, A. Mansutti, ICALP23.

Project Genesis: Manual Complexity Analysis for Arithmetic Theories

▶ **Proposition 42.** *Consider $\Psi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ in $\mathcal{S}\boldsymbol{em}$ in which variables $\boldsymbol{x} = (x_1, \ldots, x_L)$ always occur linearly, variables $\boldsymbol{y} = (y_1, \ldots, y_E)$ always occur in powers, and $L, E \geq 1$. Let $\mathcal{F} = \mathcal{S}\boldsymbol{em}$, and run Algorithm 1 from line 3 with $Q = \{(\boldsymbol{xy}, \Psi)\}$. At the end of the* **while** *loop of line 5, $D = \{\theta_1, \ldots, \theta_k\}$ satisfies the following table:*

	$\#hom$	$heft$	$\|lin(\cdot)\|$	mod	$\mathcal{B}$
Ψ	$h \geq 2$	$v \geq 2$	$c \geq 4$	$m \geq 4$	b
$\bigvee_{j=1}^{k} \theta_j$	$h^{L+1} \cdot 2^{27V^2L} \cdot \log(c \cdot m)^{V+2} + E^2$	V	$c^{2^{4(L+V)}} \cdot m^{2^{4V}}$	$c^{h^{2L+2}} m$	$k \cdot b'$

where $V := \min(2^L v, L + E + \#\boldsymbol{z})$, $\mathcal{B}(\theta_j) < b' := b + h \cdot 2^{35 \cdot V^2 L} \cdot \log(c \cdot m)^{V+2} \cdot E^3$ and $k \leq 2 \uparrow (h \cdot 2^{60 \cdot (L+2) \cdot V^2} (\log(c \cdot m))^{30 \cdot V^2} \cdot E^3)$. The number of quantifiers added to Π' is at most $h^{L+1} \cdot 2^{2^4 \cdot V^2 L} \cdot \log(c \cdot m)^{V+1}$. The procedure runs in $len(\Psi) \uparrow (h + \log(c \cdot m)) \uparrow \mathrm{poly}(L, V)$. ⌝

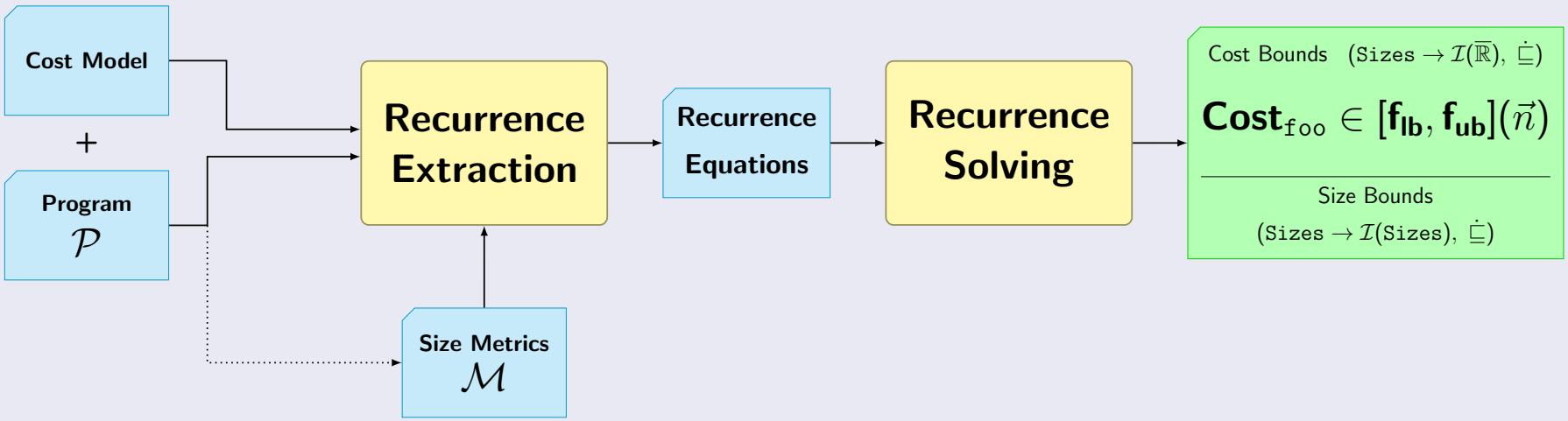
- Here, 57 pages paper,
  $\geq$ 20 pps purely of bound proofs and recurrence extraction.
- Tedious manual proof process, that must be iterated.
- Human ingenuity is concentrated on only some subparts of the proofs.

➔ Automatability potential (CAS, postfixpoints, extraction, ...)

Tool-Assisted Complexity Analysis – Interactive/Automated Hybrid

For advanced *algorithms* (e.g. automated reasoning, logic processing),

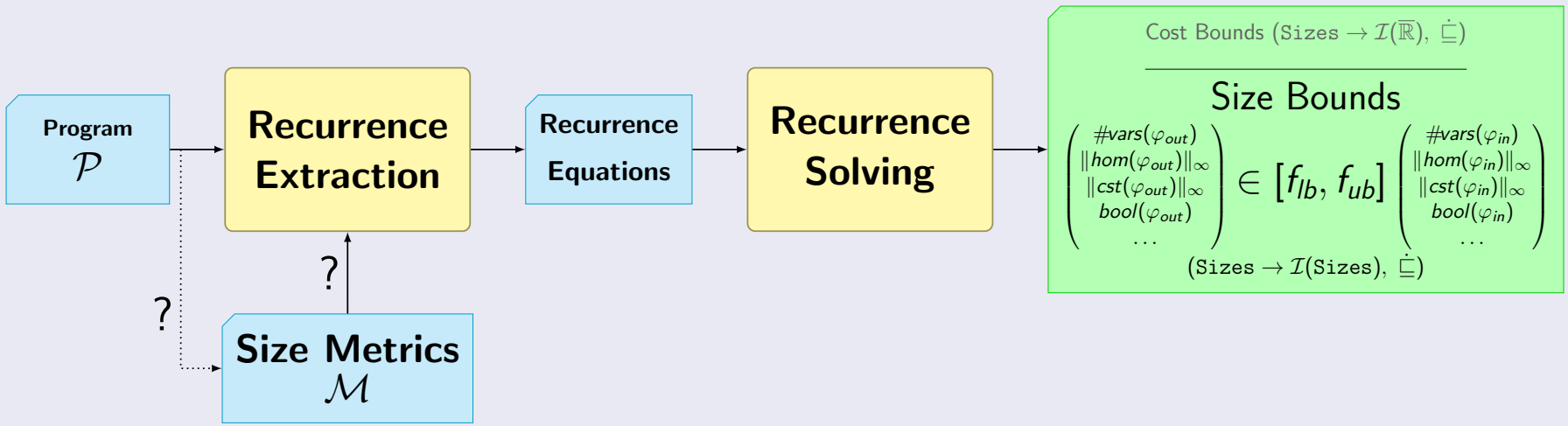
complex metrics, subtle arguments, (simplified cost model), ...



Tool-Assisted Complexity Analysis – Interactive/Automated Hybrid

For advanced *algorithms* (e.g. automated reasoning, logic processing),

complex metrics, subtle arguments, (simplified cost model), ...

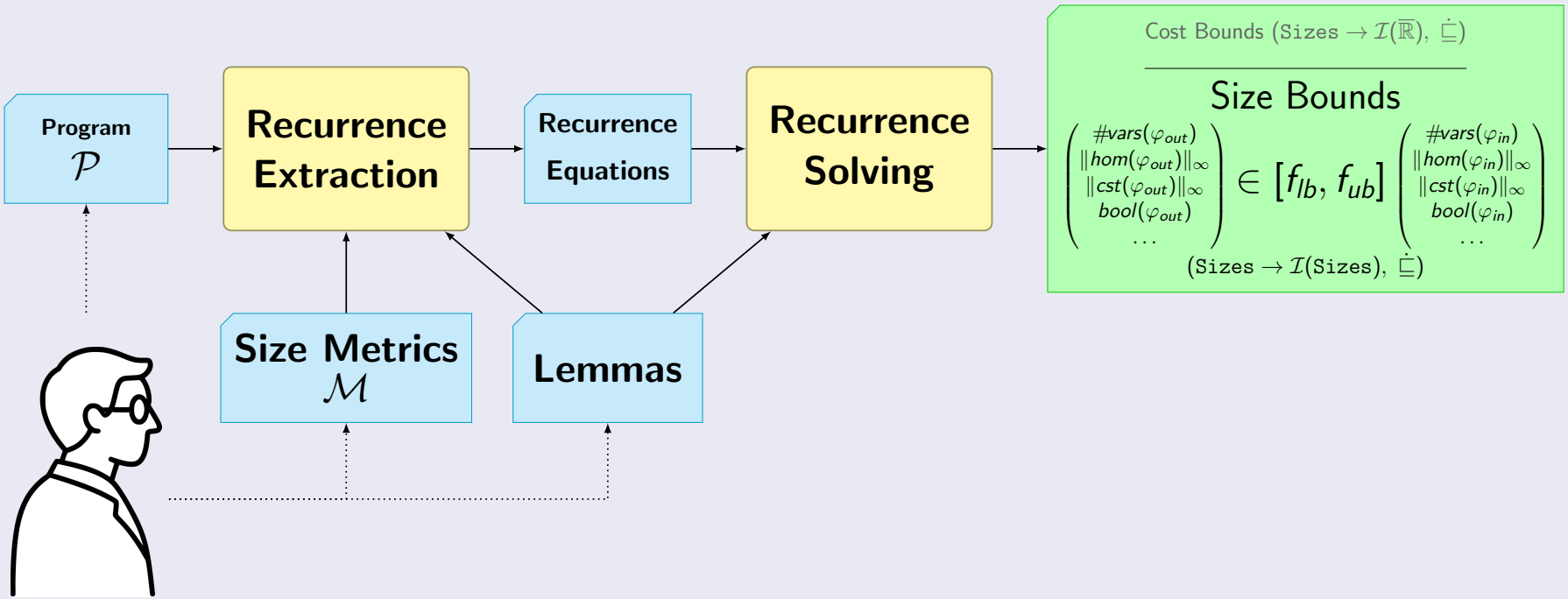


Tool-Assisted Complexity Analysis – Interactive/Automated Hybrid

For advanced *algorithms* (e.g. automated reasoning, logic processing),

complex metrics, subtle arguments, (simplified cost model), ...

include the human expert.

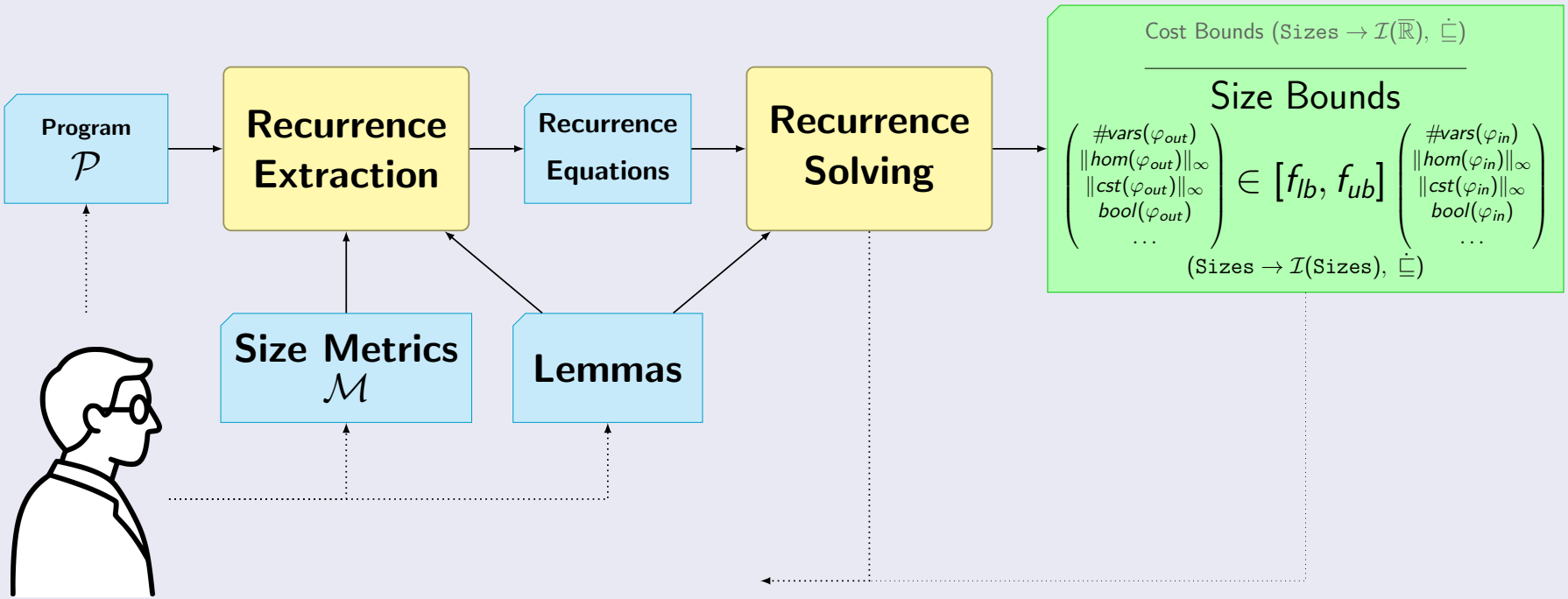


Tool-Assisted Complexity Analysis – Interactive/Automated Hybrid

For advanced *algorithms* (e.g. automated reasoning, logic processing),

complex metrics, subtle arguments, (simplified cost model), ...

include the human expert.



Tool Pipeline

ITP
- Algorithms
- Metrics
  $m\colon \mathtt{Data}_T \to \mathbb{Z}$
- Lemmas and proofs

IR

Analysable program:
- Algebraic data types
- Catamorphic metrics
- Assumed lemmas

Size Abstraction

Abstract Compilation

Generalised Recurrence Equations

Numerical Logic Program (CHCs),
Equation on Function *Intervals*,
Other *Operators* $\Phi^\sharp$
...

Recurrence Solving

CAS, postfixpoints, SMT, ...

Parametric Bounds

ITP Frontend

Example: 2-adic saturation

Saturate $S \mapsto S' := \left\{ r \cdot 2^k \,\middle|\, r \cdot 2^n \in S,\ r \in \mathbb{Z},\ 0 \leq k \leq n \right\}$.

- Two metrics: set cardinality $|S|$, max absolute value $a(S)$.

Example: 2-adic saturation

```haskell
module Saturate where

import PA.Prelude as Int
import PA.Set     as Set

{-# ANN setSize "metric" #-}
setSize :: Set Int -> Int
setSize s = Set.size s

{-# ANN absMax "metric" #-}
absMax :: Set Int -> Int
absMax s = Set.fold (\n m -> Int.max m (Int.abs n)) 0 s

div2 :: Int -> Int
div2 n = n `div` 2

decrease :: Set Int -> Set Int
decrease s = Set.map div2 (Set.filter (\n -> Int.even n && n /= 0) s)

{-# ANN saturateTwos "withAssume ubSaturateTwos"  #-}
saturateTwos :: Set Int -> Set Int
saturateTwos s
  | Set.null s = s
  | otherwise  = s `Set.union` saturateTwos (decrease s)

ubSaturateTwos :: Set Int -> Bool
ubSaturateTwos s = setSize (saturateTwos s) <= 2 * absMax s + 1
```

Saturate $S \mapsto S' := \left\{ r \cdot 2^k \;\middle|\; r \cdot 2^n \in S,\ r \in \mathbb{Z},\ 0 \leq k \leq n \right\}$.

- Two metrics: set cardinality $|S|$, max absolute value $a(S)$.

Example: 2-adic saturation

```
module Saturate where

import PA.Prelude as Int
import PA.Set     as Set

{-# ANN setSize "metric" #-}
setSize :: Set Int -> Int
setSize s = Set.size s

{-# ANN absMax "metric" #-}
absMax :: Set Int -> Int
absMax s = Set.fold (\n m -> Int.max m (Int.abs n)) 0 s

div2 :: Int -> Int
div2 n = n `div` 2

decrease :: Set Int -> Set Int
decrease s = Set.map div2 (Set.filter (\n -> Int.even n && n /= 0) s)

{-# ANN saturateTwos "withAssume ubSaturateTwos"  #-}
saturateTwos :: Set Int -> Set Int
saturateTwos s
  | Set.null s = s
  | otherwise  = s `Set.union` saturateTwos (decrease s)

ubSaturateTwos :: Set Int -> Bool
ubSaturateTwos s = setSize (saturateTwos s) <= 2 * absMax s + 1
```

Saturate $S \mapsto S' := \left\{ r \cdot 2^k \;\middle|\; r \cdot 2^n \in S,\ r \in \mathbb{Z},\ 0 \leq k \leq n \right\}$.

- Two metrics: set cardinality $|S|$, max absolute value $a(S)$.
- Automated: $|S'| \leq |S| \cdot \left(1 + \log_2(a(S))\right)$.

Example: 2-adic saturation

```
module Saturate where

import PA.Prelude as Int
import PA.Set     as Set

{-# ANN setSize "metric" #-}
setSize :: Set Int -> Int
setSize s = Set.size s

{-# ANN absMax "metric" #-}
absMax :: Set Int -> Int
absMax s = Set.fold (\n m -> Int.max m (Int.abs n)) 0 s

div2 :: Int -> Int
div2 n = n `div` 2

decrease :: Set Int -> Set Int
decrease s = Set.map div2 (Set.filter (\n -> Int.even n && n /= 0) s)

{-# ANN saturateTwos "withAssume ubSaturateTwos"  #-}
saturateTwos :: Set Int -> Set Int
saturateTwos s
  | Set.null s = s
  | otherwise  = s `Set.union` saturateTwos (decrease s)

ubSaturateTwos :: Set Int -> Bool
ubSaturateTwos s = setSize (saturateTwos s) <= 2 * absMax s + 1
```

Saturate $S \mapsto S' := \left\{ r \cdot 2^k \;\middle|\; r \cdot 2^n \in S,\ r \in \mathbb{Z},\ 0 \leq k \leq n \right\}$.

- Two metrics: set cardinality $|S|$, max absolute value $a(S)$.
- Automated: $|S'| \leq |S| \cdot \left(1 + \log_2(a(S))\right)$.
- Human lemma (pigeonhole): $|S'| \leq 2 \cdot a(S) + 1$.

Example: 2-adic saturation

```
module Saturate where

import PA.Prelude as Int
import PA.Set     as Set

{-# ANN setSize "metric" #-}
setSize :: Set Int -> Int
setSize s = Set.size s

{-# ANN absMax "metric" #-}
absMax :: Set Int -> Int
absMax s = Set.fold (\n m -> Int.max m (Int.abs n)) 0 s

div2 :: Int -> Int
div2 n = n `div` 2

decrease :: Set Int -> Set Int
decrease s = Set.map div2 (Set.filter (\n -> Int.even n && n /= 0) s)

{-# ANN saturateTwos "withAssume ubSaturateTwos"  #-}
saturateTwos :: Set Int -> Set Int
saturateTwos s
  | Set.null s = s
  | otherwise  = s `Set.union` saturateTwos (decrease s)

ubSaturateTwos :: Set Int -> Bool
ubSaturateTwos s = setSize (saturateTwos s) <= 2 * absMax s + 1
```

Saturate $S \mapsto S' := \left\{ r \cdot 2^k \;\middle|\; r \cdot 2^n \in S,\ r \in \mathbb{Z},\ 0 \leq k \leq n \right\}$.

- Two metrics: set cardinality $|S|$, max absolute value $a(S)$.
- Automated: $|S'| \leq |S| \cdot \left(1 + \log_2(a(S))\right)$.
- Human lemma (pigeonhole): $|S'| \leq 2 \cdot a(S) + 1$.
- *Combined*: $|S'| \leq |S| \cdot (3 + \log_2 \frac{a(S)}{|S|})$.
  When $a(S) \approx |S| \cdot \log_2 |S|$, we get $|S'| \in O(|S| \cdot \log_2 \log_2 |S|)$.

(Non-)Catamorphic Metrics

Definition (Metric)

Any function $m : T \to \mathbb{Z}$ from a datatype T to integers.

For which metrics can recurrence extraction be automated?
We introduce *catamorphic metrics*.

Definition (Catamorphic Metric – informally)

A metric determined by the *sizes* of immediate subterms: for type $\mathtt{T} = \ldots \mid C_i(\ldots, A_{i,j}, \ldots) \mid \ldots$,
$\exists \vec{e},\ \mathbf{m_T}(C_i(\ldots, a_{i,j}, \ldots))) = e_i(\ldots, \vec{m}_{A_{i,j}}(a_{i,j}), \ldots)$.

$$\begin{array}{ccc} F\mathcal{T} & \xrightarrow{\cong} & \mathcal{T} \\ \downarrow{\scriptstyle F\mathcal{M}} & & \downarrow{\scriptstyle \mathcal{M}} \\ F\left(\prod_\tau \mathbb{Z}^{k_\tau}\right) & \overset{\exists}{\dashrightarrow} & \prod_\tau \mathbb{Z}^{k_\tau} \end{array}$$

- Can be composed, support boolean properties, ...

➔ Non-trivial metrics combining *shape* and *content* information.

- Example of non-catamorphic metric: *cardinality*.

➔ *Overapproximate* non-catamorphic metrics, and add *lemmas*.

Lemmas and IR conversion

```
/* Interpreted by "quotient" */
type Set_int =  {
  | Set_int_Nil()
  | Set_int_Cons(int, Set_int)
}

/* Catamorphic overapproximation of *cardinality* metric */
metric setSize(s: Set_int) -> int {
  let res: int = top;
  match (s) {
    | Set_int_Nil() =>  {
        res = 0;
      }
    | Set_int_Cons(x, xs) =>  {
        /* Interval operation */
        res = [0, 1] + setSize(xs);
      }
  }
  return res;
}
```

```
def saturateTwos(s: Set_int) -> Set_int {
  let res: Set_int = top;
  if (null_Set_int(s) >= 1) {
    res = s;
  } else {
    res = union_Set_int(s, saturateTwos(decrease(s)));
  }
  assume (setSize(res) <= 2 * absMax(s) + 1);
  return res;
}
```

```
def mod(a: int, b: int) -> int {
  let res: int = top;
  assume (((0 <= res) && (res <= (b - 1))));
  if (((0 <= a) && (a <= (b - 1)))) {
    assume ((a == res));
  }
  return res;
}

def filter_even_nz(s: Set_int) -> Set_int {
  let res: Set_int = top;
  match (s) {
    | Set_int_Nil() =>  {
        res = Set_int_Nil();
      }
    | Set_int_Cons(x, xs) =>  {
        if mod(x,2) == 0 && !(x == 0) {
            res = Set_int_Cons(x, filter_even_nz(xs));
        } else {
            res = filter_even_nz(xs);
        }

      }
  }
  return res;
}

def decrease(s: Set_int) -> Set_int {
  return map_div2(filter_even_nz(s));
}
```

Abstract Compilation: Recurrence Extraction

Recurrence Extraction as HO Abstract Interpretation

"Recurrences are Abstractions of Programs"

Programs $\longleftrightarrow$ Operators $\longleftrightarrow$ Recurrences

Operator Semantics (and its abstractions)

$$[\![\texttt{Prog}]\!]_\Phi = \Phi_{\texttt{Prog}} \in \mathrm{End}\Big(\prod_{f \in \mathcal{F}} \mathrm{Mem}\mathcal{V}_{f,in} \to \mathcal{P}(\mathrm{Mem}\mathcal{V}_{f,out})\Big)$$

Size abstraction; $(\mathrm{Mem} : \prod_\tau \mathcal{V}_\tau \to [\![\tau]\!]_\mathcal{T}) \rightsquigarrow (\mathrm{Mem}^\sharp : \prod_\tau \mathcal{V}_\tau \to \mathbb{Z}^{k_\tau})$.
Numerical FOL to more usual recurrences: *effect abstraction*; $\mathcal{P} \rightsquigarrow \mathcal{I}$.

Flexibility, optimality, ...

- ➔ *Abstract compiler* (implemented in Rust)
  abstract interpreter with HO domains; for recurrence extraction
  - Backends in SMT, CHC, several receq formats, etc.
- ➔ Auxiliary analyses, verification, inference, ...
- (Bonus: *Size lifter* for abstract domains)

Recurrence Solving: Search and Verification of (Inductive) Bounds

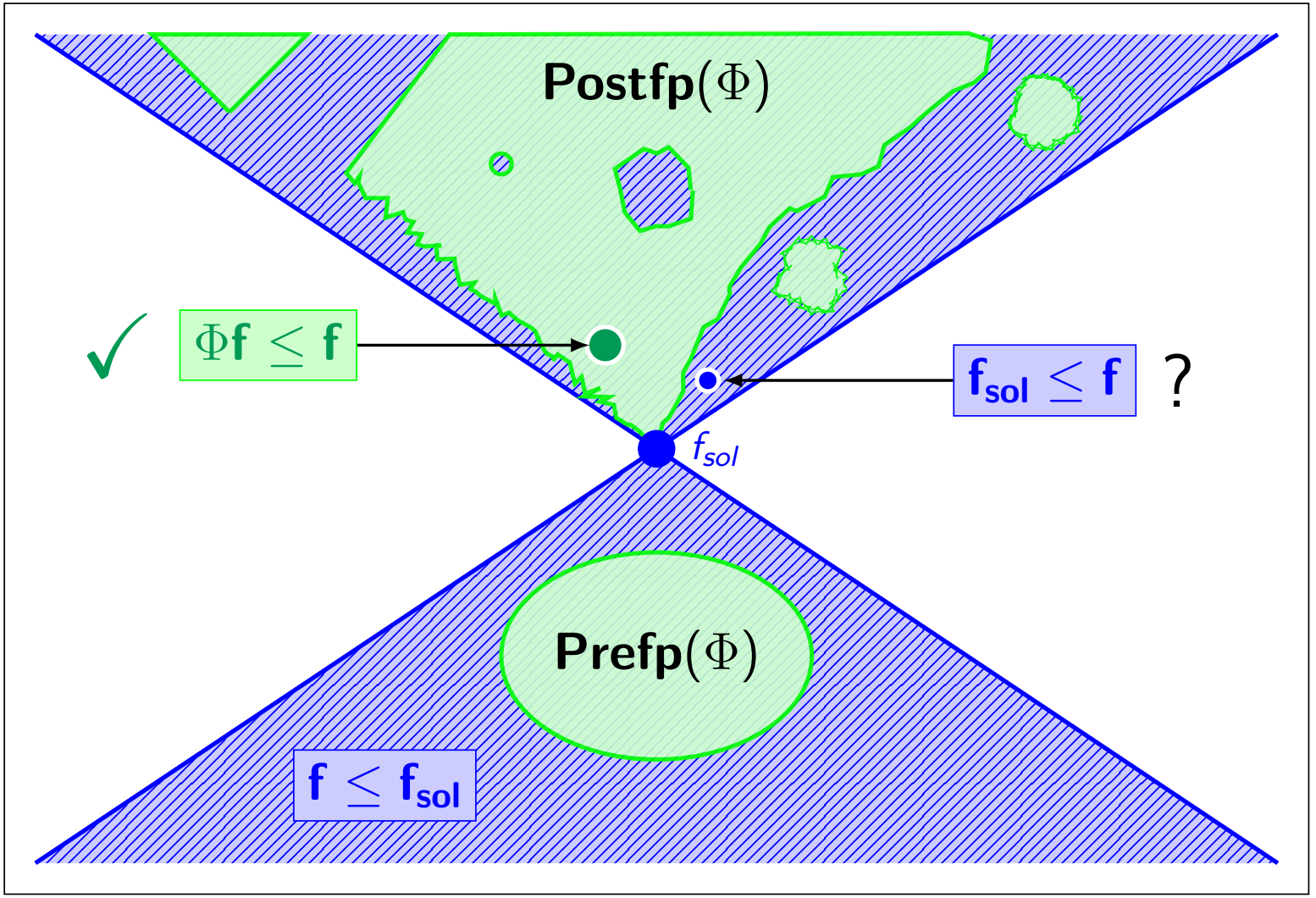
Exact Recurrence Resolution and its Limitations

- "Recurrence Solving" is a powerful tool:
  obtain non-linear, non-polynomial invariants.
- But the class of problems for which exact resolution is possible is limited.

➔ Postfixpoint-based techniques.

Equation solving as pre/postfixpoint search [SAS24, STTT26]

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



Equation solving as pre/postfixpoint search [SAS24, STTT26]

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

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

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

Theorem (Knaster-Tarski corollary)

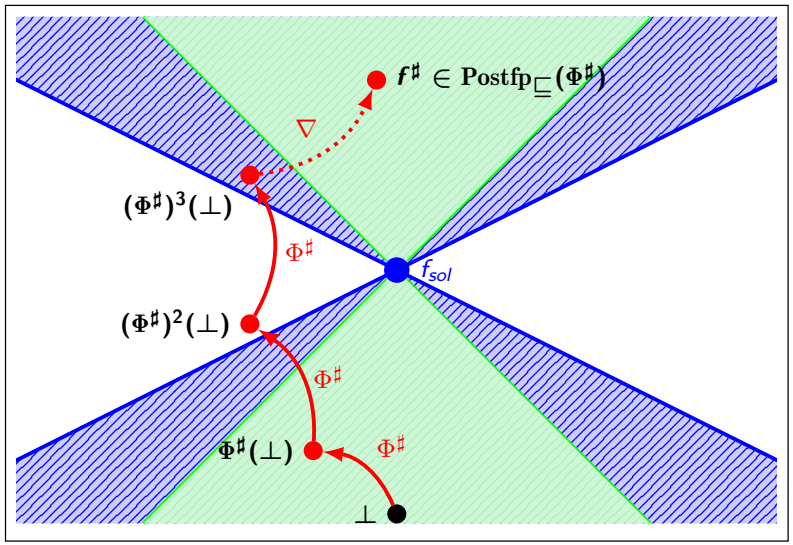
Let Φ be a **monotone equation**.

- *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 $f \leq \mathrm{gfp}\,\Phi$.*

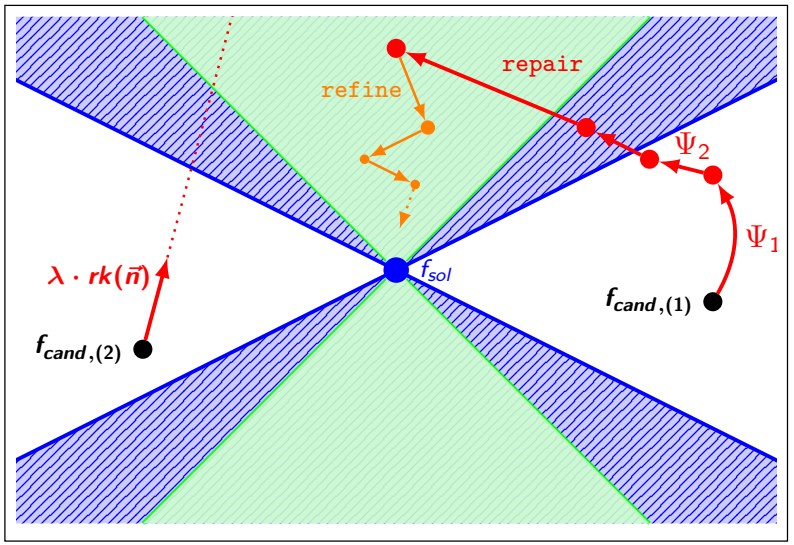
Various orders: pointwise; inclusion order for $\Phi \in \mathrm{End}(\mathbb{Z} \to \mathcal{I}(\mathbb{Z}))$; etc.

"Terminating Equation" $\rightsquigarrow \mathrm{lfp}\,\Phi = \mathrm{gfp}\,\Phi$.

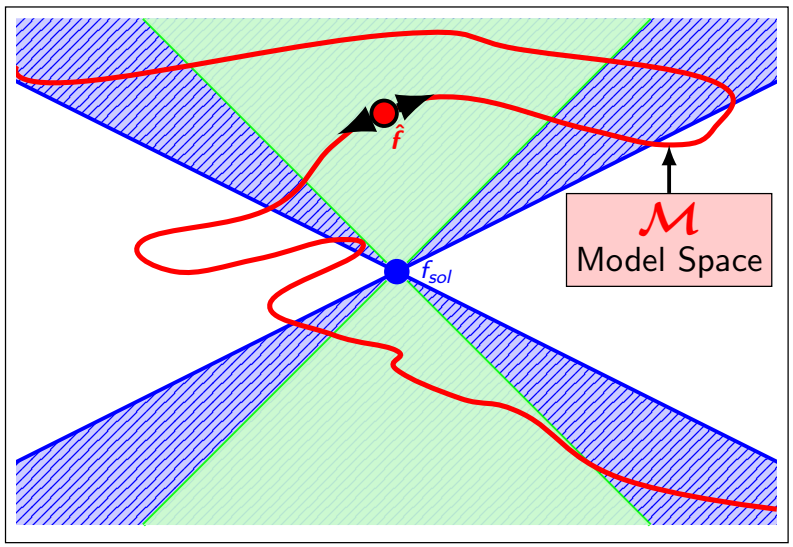
Equation solving as pre/postfixpoint search [SAS24, STTT26]



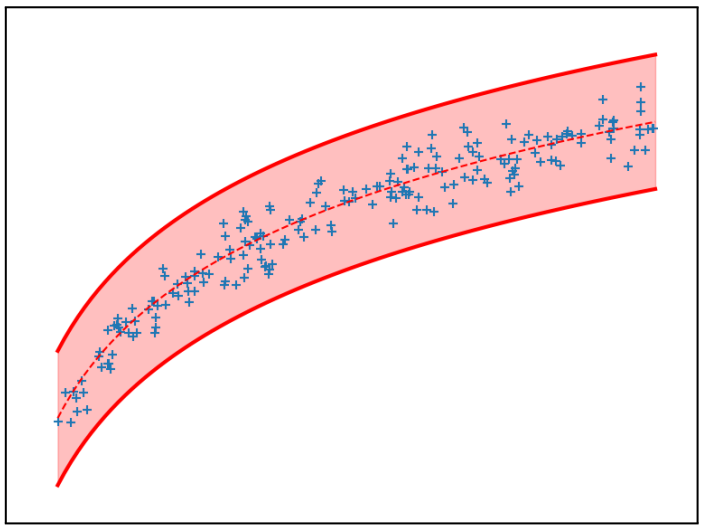
Abstract Interpretation



Geometry-based expression **Repair**



Search on subvarieties: **Templates**, $\forall$-elim



Constrained **Optimisation**,
with *provability constraints*

Function Comparison

A non-trivial primitive for some postfixpoint-based reasoning techniques.

$\forall \vec{n},\ \Phi(\hat{f})(\vec{n}) \overset{?}{\leq} \hat{f}(\vec{n})$

Verification ?

$\exists \vec{\alpha},\ \forall \vec{n},\ \Phi(\hat{f}_{\vec{\alpha}})(\vec{n}) \overset{?}{\leq} \hat{f}_{\vec{\alpha}}$

Inference ??

- Classically: incomplete methods, CAS, rewritings, interval-based localisation of zeros, ...
- For some classes of Φ and $\hat{f}$, complete methods are available (at least under real relaxation).

➔ Recently, promising progress in transcendental SMT (SMT with exponentials, etc.): SWINE, dREAL, METITARSKI, MATHSAT, YICES-TRA, ...

Conclusion, Discussion and Work in Progress

Next Steps

- Complete Implementation
  - ITP/IR integration (+ libraries soundness proofs: "quotient" representation vs "opaque" types).
  - Optimisations and further abstract domains in abstract compiler.
  - Improved treatment of "execution" of non-terminating equations and parametric optimisation, for *inference* (one of the approaches).
  - Plug transcendental SMT for *verification*.
- Experimental Evaluation

➔ **Mechanised 3EXP proof for Presburger Arithmetic**

Next Steps

- Complete Implementation
  - ITP/IR integration (+ libraries soundness proofs: "quotient" representation vs "opaque" types).
  - Optimisations and further abstract domains in abstract compiler.
  - Improved treatment of "execution" of non-terminating equations and parametric optimisation, for *inference* (one of the approaches).
  - Plug transcendental SMT for *verification*.
- Experimental Evaluation
- → **Mechanised 3EXP proof for Presburger Arithmetic**
- Future work: cost model integration, user interaction, generalisation by exploiting abstract compilation flexibility, further applications...

Thank you!

Questions?