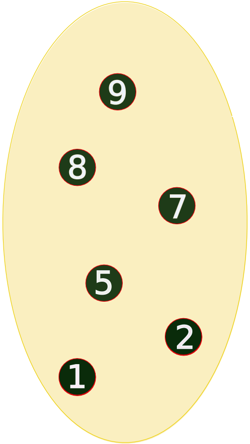
Synthesizing Abstract Transformers

Pankaj Kumar Kalita [1], Sujit Muduli [1], Loris D'Antoni [2], Thomas Reps [2], Subhajit Roy [1]

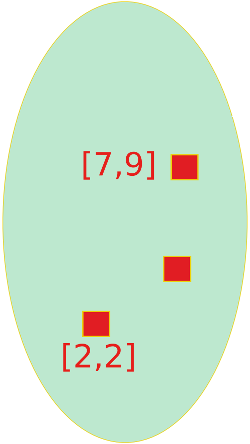
[1]Indian Institute of Technology Kanpur

[2]University of Wisconsin-Madison

Abstract Interpretation

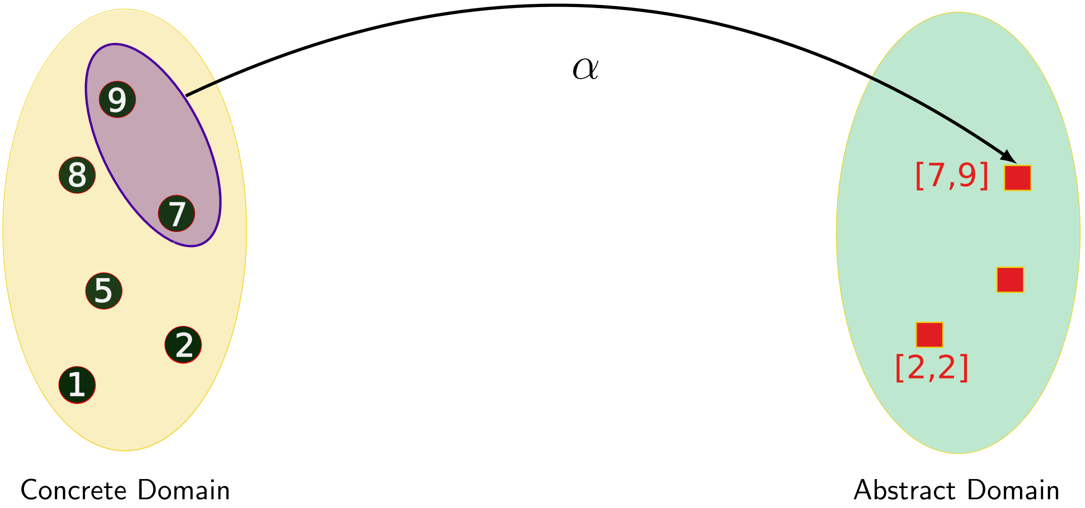


Concrete Domain

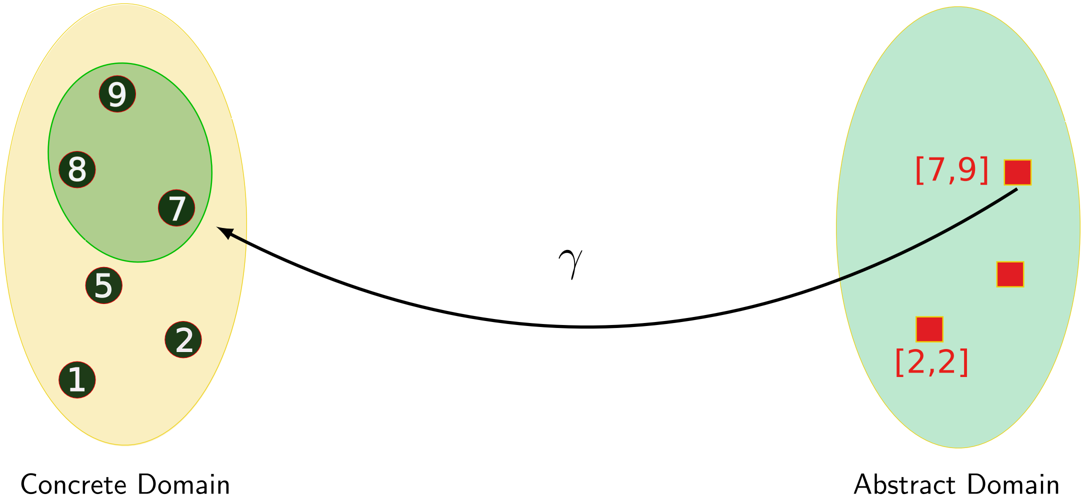


Abstract Domain

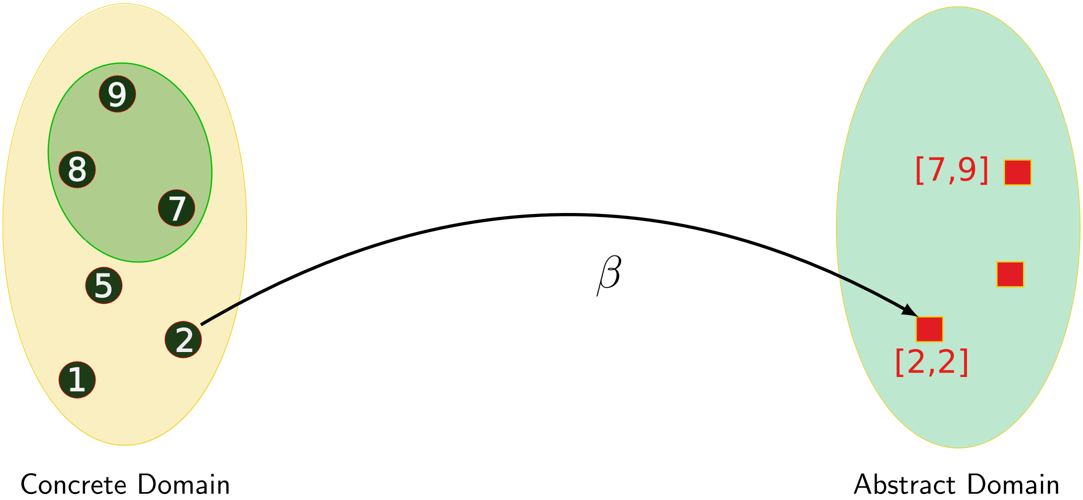
Abstract Interpretation



Abstract Interpretation

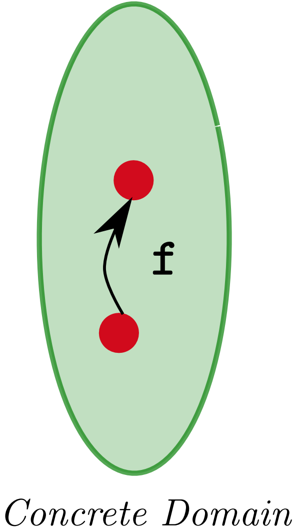


Abstract Interpretation

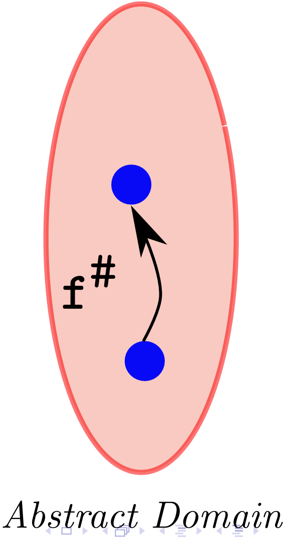


Abstract Transformer

Abstract Transformer



- Tricky even for trivial operation
- Error-prone



Problem statement

Can we automatically *synthesize* an abstract transformer for any operations?

Problem statement

Can we automatically *synthesize* an abstract transformer for any operations?

Given the

- concrete semantics Φ_f of a concrete transformer f,
- description of an abstract domain $(A, \sqsubseteq, \sqcup)$, and its relation to the concrete domain (α, γ, β), and
- a domain-specific language L,

synthesize the **best** abstract transformer $\widehat{f^\sharp}$ of f for A in L.

Problem statement

Can we automatically *synthesize* an abstract transformer for any operations?

Given the

- concrete semantics Φ_f of a concrete transformer f,
- description of an abstract domain $(A, \sqsubseteq, \sqcup)$, and its relation to the concrete domain (α, γ, β), and
- a domain-specific language L,

synthesize the **best** abstract transformer $\widehat{f^\sharp}$ of f for A in L.

$$\widehat{f^\sharp} = \lambda a : \sqcup\{\beta(f(c_i)) \mid c_i \in \gamma(a)\}$$

Motivation

Given the following domain-specific language, we desire to get an abstract transformer for Math.abs in the interval domain:

$$\mathit{Transformer} ::= \lambda \texttt{a}.[E, E]$$
$$E ::= \texttt{a.l} \mid \texttt{a.r} \mid 0 \mid -E \mid +\infty \mid -\infty \mid E + E \mid E - E \mid E * E \mid \texttt{min}(E, E) \mid \texttt{max}(E, E)$$

Motivation

Given the following domain-specific language, we de bstract transformer for Math.abs in the interval domain:

a.l: Left limit

a.r: Right limit

$$\textit{Transformer} ::= \lambda \texttt{a}.[E, E]$$

$$E ::= \texttt{a.l} \mid \texttt{a.r} \mid 0 \mid -E \mid +\infty \mid -\infty \mid E + E \mid E - E \mid E * E \mid \texttt{min}(E, E) \mid \texttt{max}(E, E)$$

Motivation

Given the following domain-specific language, we de… …bstract transformer for Math.abs in the interval domain:

a.l: Left limit
a.r: Right limit

$$\textit{Transformer} ::= \lambda \mathtt{a}.[E, E]$$
$$E ::= \mathtt{a.l} \mid \mathtt{a.r} \mid 0 \mid -E \mid +\infty \mid -\infty \mid E + E \mid E - E \mid E * E \mid \mathtt{min}(E, E) \mid \mathtt{max}(E, E)$$

We could emit one of the following abstract transformers:

$$\mathtt{abs}^{\sharp}_{1}(\mathtt{a} : \mathcal{A}_{\mathtt{intv}}) : \mathcal{A}_{\mathtt{intv}} = [\mathtt{max}(\mathtt{max}(0, \mathtt{a.l}), -\mathtt{a.r}),\ \mathtt{max}(-\mathtt{a.l}, \mathtt{a.r})]$$

Motivation

Given the following domain-specific language, we de bstract transformer for Math.abs in the interval domain:

a.l: Left limit

a.r: Right limit

$$\textit{Transformer} ::= \lambda \mathsf{a}.[E, E]$$

$$E ::= \texttt{a.l} \mid \texttt{a.r} \mid 0 \mid -E \mid +\infty \mid -\infty \mid E + E \mid E - E \mid E * E \mid \texttt{min}(E, E) \mid \texttt{max}(E, E)$$

We could emit one of the following abstract transformers:

$$\mathsf{abs}_1^\sharp(\mathsf{a} : \mathcal{A}_{\texttt{intv}}) : \mathcal{A}_{\texttt{intv}} = [\texttt{max}(\texttt{max}(0, \mathsf{a}.\texttt{l}), -\mathsf{a}.\texttt{r}),\ \texttt{max}(-\mathsf{a}.\texttt{l}, \mathsf{a}.\texttt{r})]$$

$$\mathsf{abs}_2^\sharp(\mathsf{a} : \mathcal{A}_{\texttt{intv}}) : \mathcal{A}_{\texttt{intv}} = [\texttt{max}(0, \mathsf{a}.\texttt{l}) - \texttt{min}(0, \mathsf{a}.\texttt{r}),\ \texttt{max}(-\mathsf{a}.\texttt{l}, \mathsf{a}.\texttt{r})]$$

Challenges

1. $\widehat{f^\sharp}$ may not be computable.

Challenges

1. $\widehat{f^\sharp}$ may not be computable.
2. $\widehat{f^\sharp}$ may not be expressible in L.

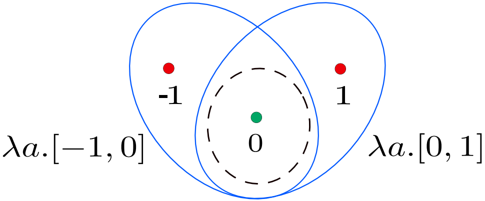
Challenges

1. $\widehat{f^\sharp}$ may not be computable.
2. $\widehat{f^\sharp}$ may not be expressible in L.
3. Precision defines a partial ordering on abstract transformers, so $f^\sharp \in L$ may not be unique.

Challenges

1. $\widehat{f^\sharp}$ may not be computable.
2. $\widehat{f^\sharp}$ may not be expressible in L.
3. Precision defines a partial ordering on abstract transformers, so $f^\sharp \in L$ may not be unique. Example:

`f(x) = 0,`
$L = \{\lambda a.\ [0, k], \lambda a.\ [-k, 0]\ \mid k \in \mathbb{N} \wedge k \geq 1\}$,
$\widehat{\mathcal{S}}^\sharp_L = \{\lambda a.[0,1], \lambda a.[-1,0]\}$



Best L-transformer

***L*-transformer ($f_L^\sharp$)**

An abstract transformer $f^\sharp$ is a L-transformer (denoted as $f_L^\sharp$) if $f^\sharp \in L$.

Best L-transformer

***L-transformer** ($f_L^\sharp$)*

An abstract transformer $f^\sharp$ is a L-transformer (denoted as $f_L^\sharp$) if $f^\sharp \in L$.

***best L-transformer** ($\widehat{f_L^\sharp}$)*

An abstract transformer $f^\sharp \in L$ is a **best L-transformer** (denoted as $\widehat{f_L^\sharp}$) if $f_L^\sharp$ is both sound and there does not exist any other transformer in L that is more precise.

Problem statement (revised)

Given the

- concrete semantics Φ_f of a concrete transformer f,
- description of an abstract domain $(A, \sqsubseteq, \sqcup)$, and its relation to the concrete domain (α, γ, β), and
- a domain-specific language L,

synthesize **the best** abstract transformer $\widehat{f^\sharp}$ of f for A in L.

Problem statement (revised)

Given the

- concrete semantics Φ_f of a concrete transformer f,
- description of an abstract domain $(A, \sqsubseteq, \sqcup)$, and its relation to the concrete domain (α, γ, β), and
- a domain-specific language L,

~~synthesize **the best** abstract transformer $\widehat{f^\sharp}$ of f for A in L.~~

synthesize **a best L-transformer** $\widehat{f^\sharp_L}$ of f for A in L.

Problem statement (revised)

Given the

- concrete semantics Φ_f of a concrete transformer f,
- description of an abstract domain $(A, \sqsubseteq, \sqcup)$, and its relation to the concrete domain (α, γ, β), and
- a domain-specific language L,

~~synthesize **the best** abstract transformer $\widehat{f^\sharp}$ of f for A in L.~~

synthesize **a best L-transformer** $\widehat{f^\sharp_L}$ of f for A in L.

We build a tool, अमूर्त (AMURTH), that solves the above problem.

Problem statement (revised)

Given the

- concrete semantics Φ_f of a concrete transformer f,
- description of an abstract domain $(A, \sqsubseteq, \sqcup)$, and its relation to the concrete domain (α, γ, β), and
- a domain-specific language L,

~~synthesize **the best** abstract transformer $\widehat{f^\sharp}$ of f for A in L.~~

synthesize **a best L-transformer** $\widehat{f_L^\sharp}$ of f for A in L.

We build a tool, अमूर्त (AMURTH), that solves the above problem.

Abstract Interpretation Engines for Free!

Who cares?

- Abstract transformers are often non-trivial even for a simple operation.

Who cares?

- Abstract transformers are often non-trivial even for a simple operation.
- Manually written abstract transformers error-prone.

Who cares?

- Abstract transformers are often non-trivial even for a simple operation.
- Manually written abstract transformers error-prone.
- We found multiple bugs in abstract transformers in the existing abstract interpretation engines.

Who cares?

- Abstract transformers are often non-trivial even for a simple operation.
- Manually written abstract transformers error-prone.
- We found multiple bugs in abstract transformers in the existing abstract interpretation engines.

AMURTH can synthesize non-trivial transformers in reasonable time (< 2000 seconds).

Who cares?

- Abstract transformers are often non-trivial even for a simple operation.
- Manually written abstract transformers error-prone.
- We found multiple bugs in abstract transformers in the existing abstract interpretation engines.

AMURTH can synthesize non-trivial transformers in reasonable time (< 2000 seconds).

Now, let's dive into the workings of AMURTH.

Algorithm Overview

- AMURTH uses counterexample-guided inductive synthesis (CEGIS) strategy.

Algorithm Overview

- AMURTH uses counterexample-guided inductive synthesis (CEGIS) strategy.
- Attempts to meet the dual objectives of soundness and precision

Algorithm Overview

- AMURTH uses counterexample-guided inductive synthesis (CEGIS) strategy.
- Attempts to meet the dual objectives of soundness and precision
- Correspondingly, the algorithm is guided by two kinds of examples,

Algorithm Overview

- AMURTH uses counterexample-guided inductive synthesis (CEGIS) strategy.
- Attempts to meet the dual objectives of soundness and precision
- Correspondingly, the algorithm is guided by two kinds of examples,
  - **Positive Example**
    - $\langle a, c' \rangle$ such that, $a \in A$ and $c' \in \gamma(\hat{f}^\sharp(a))$.

Algorithm Overview

- AMURTH uses counterexample-guided inductive synthesis (CEGIS) strategy.
- Attempts to meet the dual objectives of soundness and precision
- Correspondingly, the algorithm is guided by two kinds of examples,
  - **Positive Example**
    - $\langle a, c' \rangle$ such that, $a \in A$ and $c' \in \gamma(\hat{f}^\sharp(a))$.
    - Violation of *soundness verification* generates positive examples (E^+).
      e.g., $\langle [5, 9], 6 \rangle$ (abs in interval domain)

Algorithm Overview

- AMURTH uses counterexample-guided inductive synthesis (CEGIS) strategy.
- Attempts to meet the dual objectives of soundness and precision
- Correspondingly, the algorithm is guided by two kinds of examples,
  - **Positive Example**
    - $\langle a, c' \rangle$ such that, $a \in A$ and $c' \in \gamma(\hat{f}^\sharp(a))$.
    - Violation of *soundness verification* generates positive examples (E^+).
      e.g., $\langle [5, 9], 6 \rangle$ (abs in interval domain)
  - **Negative Example**
    - $\langle a, c' \rangle$ such that, $a \in A$ and $\exists\ \hat{f}^\sharp_L \in \hat{\mathcal{S}}^\sharp_L$ such that, $c' \notin \gamma(\hat{f}^\sharp_L(a))$

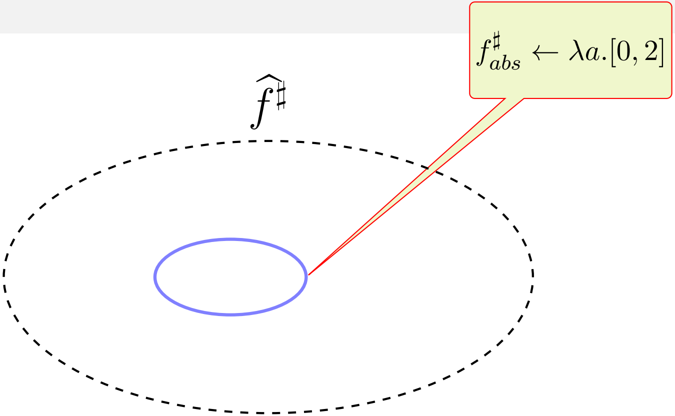
Algorithm Overview

- AMURTH uses counterexample-guided inductive synthesis (CEGIS) strategy.
- Attempts to meet the dual objectives of soundness and precision
- Correspondingly, the algorithm is guided by two kinds of examples,
  - **Positive Example**
    - $\langle a, c' \rangle$ such that, $a \in A$ and $c' \in \gamma(\hat{f}^\sharp(a))$.
    - Violation of *soundness verification* generates positive examples (E^+).
      e.g., $\langle [5, 9], 6 \rangle$ (abs in interval domain)
  - **Negative Example**
    - $\langle a, c' \rangle$ such that, $a \in A$ and $\exists\ \hat{f}^\sharp_L \in \hat{\mathcal{S}}^\sharp_L$ such that, $c' \notin \gamma(\hat{f}^\sharp_L(a))$
    - Violation of *precision verification* generates negative examples (E^-).
      e.g., $\langle [5, 9], 2 \rangle$ (abs in interval domain)

Algorithm Overview

- AMURTH uses counterexample-guided inductive synthesis (CEGIS) strategy.
- Attempts to meet the dual objectives of soundness and precision
- Correspondingly, the algorithm is guided by two kinds of examples,
  - **Positive Example**
    - $\langle a, c' \rangle$ such that, $a \in A$ and $c' \in \gamma(\hat{f}^\sharp(a))$.
    - Violation of *soundness verification* generates positive examples (E^+).
      e.g., $\langle [5, 9], 6 \rangle$ (abs in interval domain)
  - **Negative Example**
    - $\langle a, c' \rangle$ such that, $a \in A$ and $\exists\ \hat{f}^\sharp_L \in \hat{\mathcal{S}}^\sharp_L$ such that, $c' \notin \gamma(\hat{f}^\sharp_L(a))$
    - Violation of *precision verification* generates negative examples (E^-).
      e.g., $\langle [5, 9], 2 \rangle$ (abs in interval domain)
- Counterexamples generated by the **soundness** and **precision** verifiers drive two CEGIS loops.

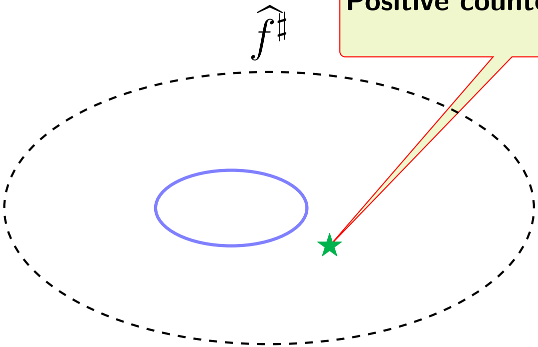
AMURTH in action!



AMURTH in action!

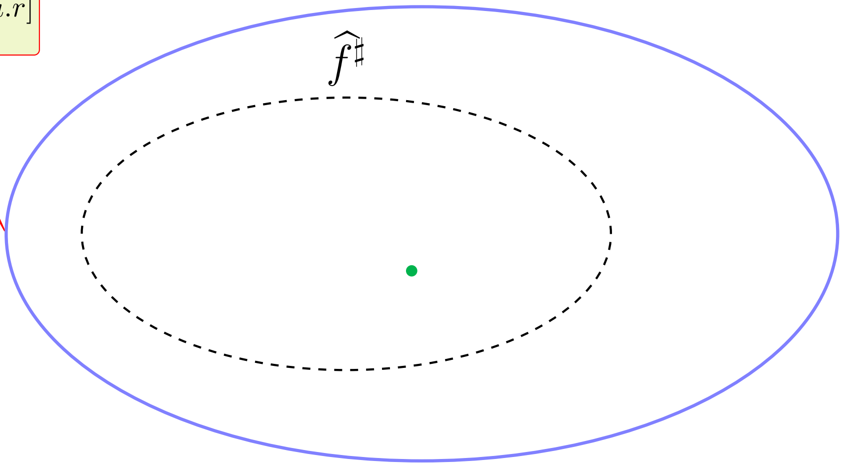
$$f^{\sharp}_{abs} \leftarrow \lambda a.[0, 2]$$

Positive counterexample: $\langle [0, 5], 3 \rangle$



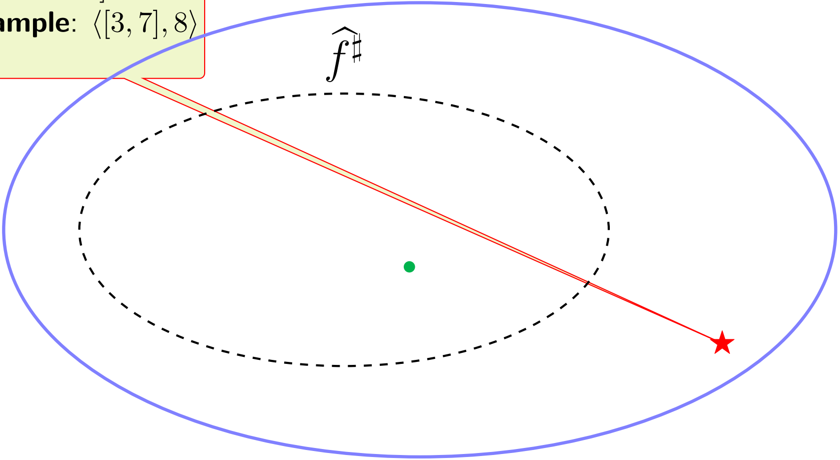
Amurth in action!

$$f^\sharp_{abs} \leftarrow \lambda a.[0, a.l + a.r]$$

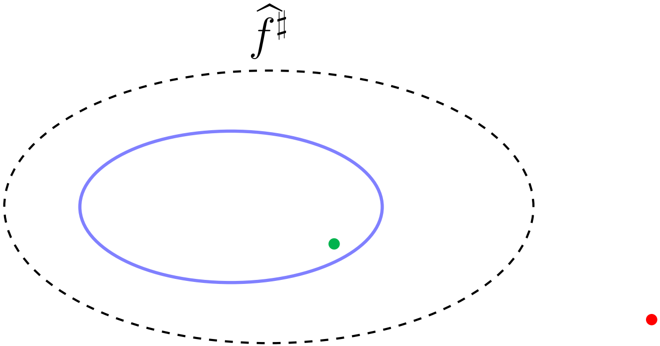


$$f^{\sharp}_{abs} \leftarrow \lambda a.[0, a.l + a.r]$$

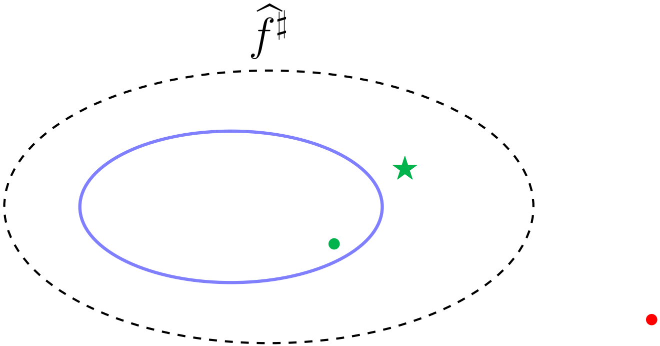
Negative counterexample: $\langle [3,7], 8 \rangle$



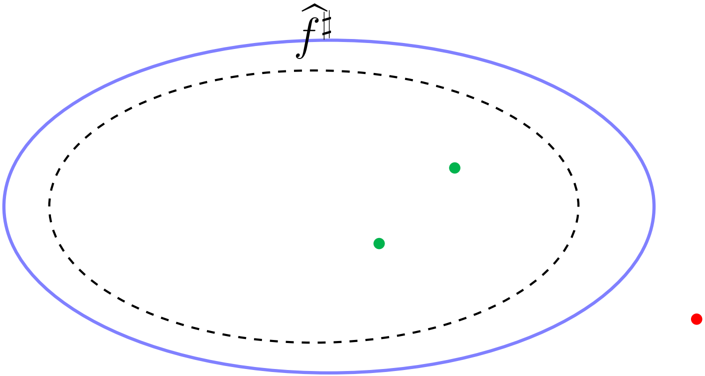
AMURTH in action!



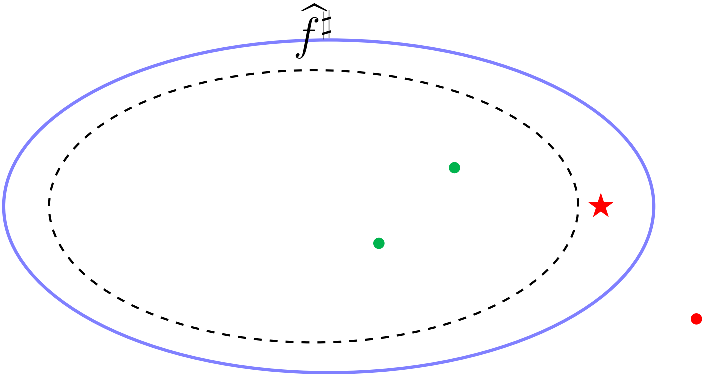
AMURTH in action!



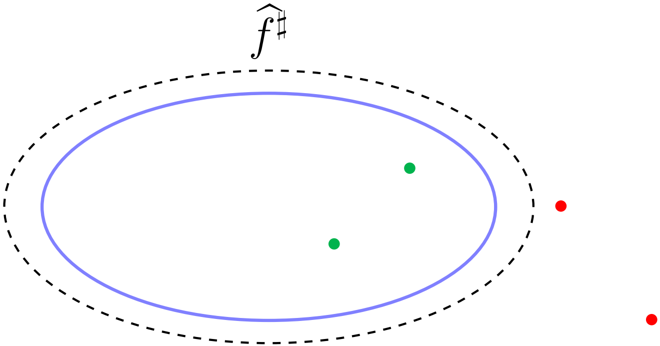
AMURTH in action!



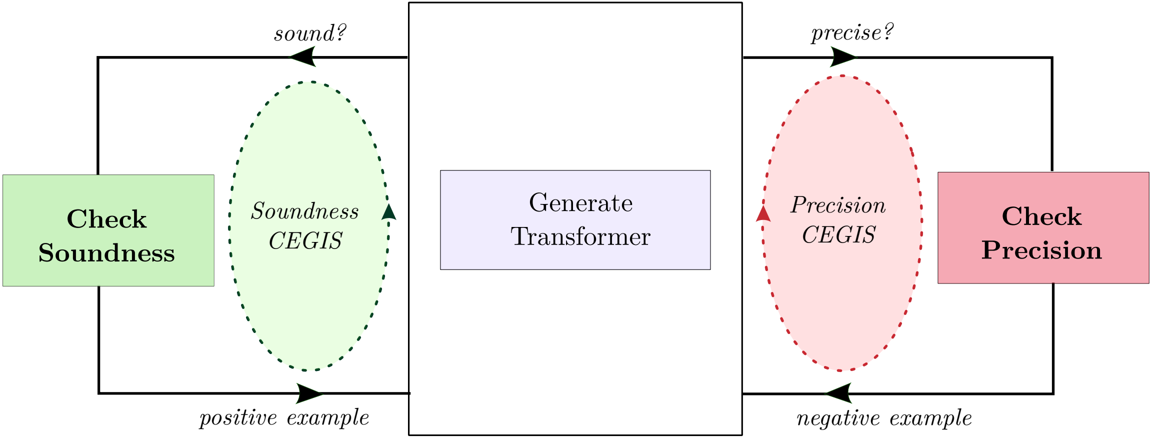
AMURTH in action!



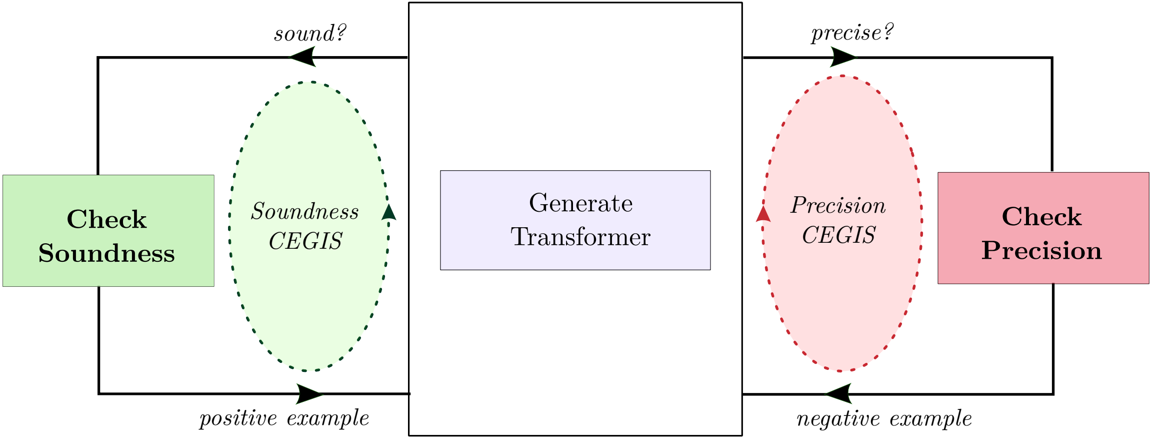
AMURTH in action!



Algorithm


sound?
precise?
Check
Soundness
Soundness
CEGIS
Generate
Transformer
Precision
CEGIS
Check
Precision
positive example
negative example

Algorithm



Additional algorithmic components are needed! (see the paper for details)

Claims

Theorem 1

If Algorithm terminates, it returns a best L-transformer for the concrete function f.

Theorem 2

If the DSL L is finite, Algorithm always terminates.

Evaluation: Domains and Operations

Domain Type	Abstract Domains	Operations
String	Constant String ($\mathcal{CS}$)	charAt$^\sharp$
	String Set (size k) ($\mathcal{SS}_k$)	concat$^\sharp$,
	Char Inclusion ($\mathcal{CI}$)	contains$^\sharp$,
	Prefix-Suffix ($\mathcal{PS}$)	toLower$^\sharp$, toUpper$^\sharp$,
	String Hash ($\mathcal{SH}$)	trim$^\sharp$
Fixed Bitwidth Interval	Unsigned-Int ($\mathcal{A}_{\texttt{uintv}}$)	add$^\sharp$, sub$^\sharp$, mul$^\sharp$,
	Signed-Int ($\mathcal{A}_{\texttt{uintv}}$)	and$^\sharp$, or$^\sharp$, xor$^\sharp$,
	Wrapped ($\mathcal{W}$)	shl$^\sharp$, ashr$^\sharp$, lshr$^\sharp$

Domains proposed in prior work.

Evaluation: Domains and Operations

Domain Type	Abstract Domains	Operations
String	Constant String ($\mathcal{CS}$) String Set (size k) ($\mathcal{SS}_k$) Char Inclusion ($\mathcal{CI}$) Prefix-Suffix ($\mathcal{PS}$) String Hash ($\mathcal{SH}$)	$\texttt{charAt}^\sharp$ $\texttt{concat}^\sharp$, $\texttt{contains}^\sharp$, $\texttt{toLower}^\sharp$, $\texttt{toUpper}^\sharp$, $\texttt{trim}^\sharp$
Fixed Bitwidth Interval	Unsigned-Int ($\mathcal{A}_{\texttt{uintv}}$) Signed-Int ($\mathcal{A}_{\texttt{uintv}}$) Wrapped ($\mathcal{W}$)	$\texttt{add}^\sharp$, $\texttt{sub}^\sharp$, $\texttt{mul}^\sharp$, $\texttt{and}^\sharp$, $\texttt{or}^\sharp$, $\texttt{xor}^\sharp$, $\texttt{shl}^\sharp$, $\texttt{ashr}^\sharp$, $\texttt{lshr}^\sharp$

Domains proposed in prior work.

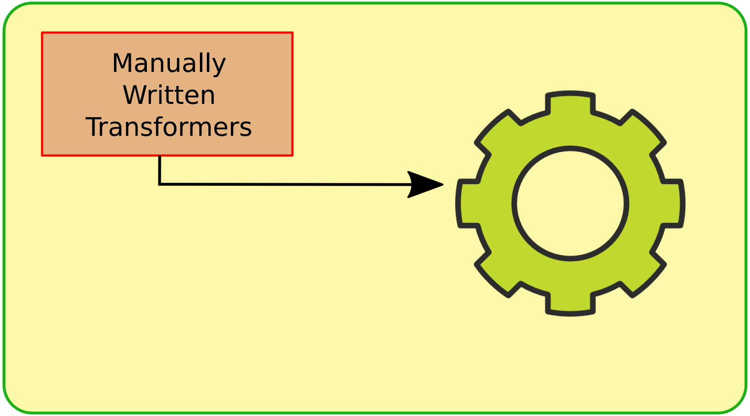
Evaluation: Domains and Operations

Domain Type	Abstract Domains	Operations
String	Constant String ($\mathcal{CS}$)	$\texttt{charAt}^\sharp$
	String Set (size k) ($\mathcal{SS}_k$)	$\texttt{concat}^\sharp$,
	Char Inclusion ($\mathcal{CI}$)	$\texttt{contains}^\sharp$,
	Prefix-Suffix ($\mathcal{PS}$)	$\texttt{toLower}^\sharp$, $\texttt{toUpper}^\sharp$,
	String Hash ($\mathcal{SH}$)	$\texttt{trim}^\sharp$
Fixed Bitwidth Interval	Unsigned-Int ($\mathcal{A}_{\texttt{uintv}}$)	$\texttt{add}^\sharp$, $\texttt{sub}^\sharp$, $\texttt{mul}^\sharp$,
	Signed-Int ($\mathcal{A}_{\texttt{uintv}}$)	$\texttt{and}^\sharp$, $\texttt{or}^\sharp$, $\texttt{xor}^\sharp$,
	Wrapped ($\mathcal{W}$)	$\texttt{shl}^\sharp$, $\texttt{ashr}^\sharp$, $\texttt{lshr}^\sharp$

Domains proposed in prior work.

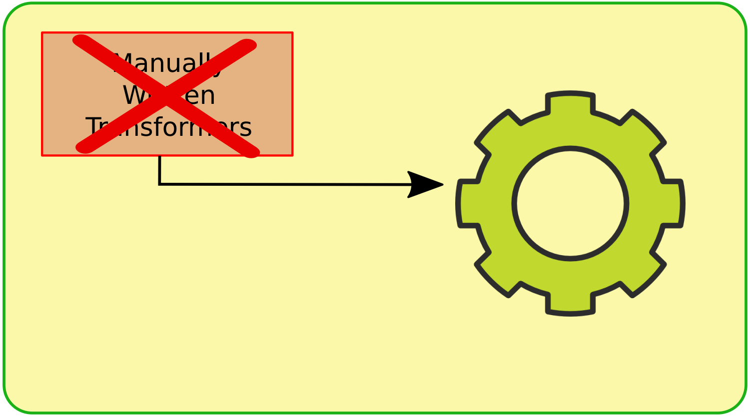
Evaluation: Methodology

(Existing) Abstract Interpretation Engine



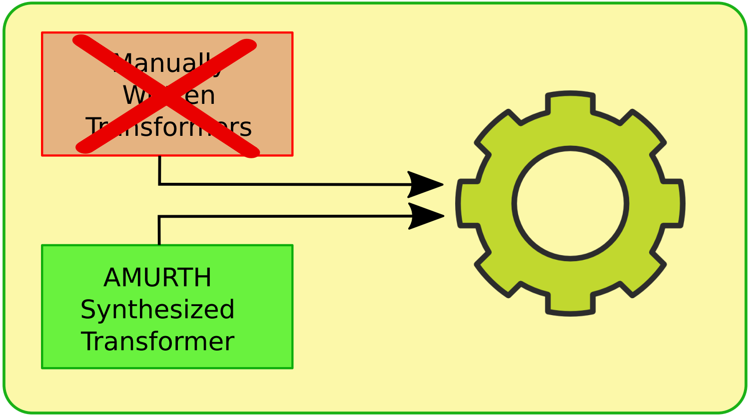
Evaluation: Methodology

(Existing) Abstract Interpretation Engine

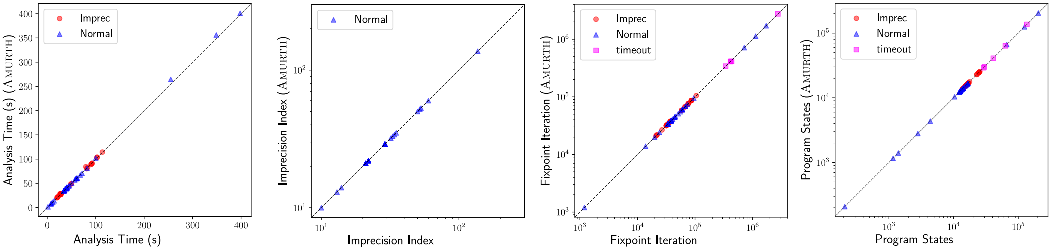


Evaluation: Methodology

(Existing) Abstract Interpretation Engine

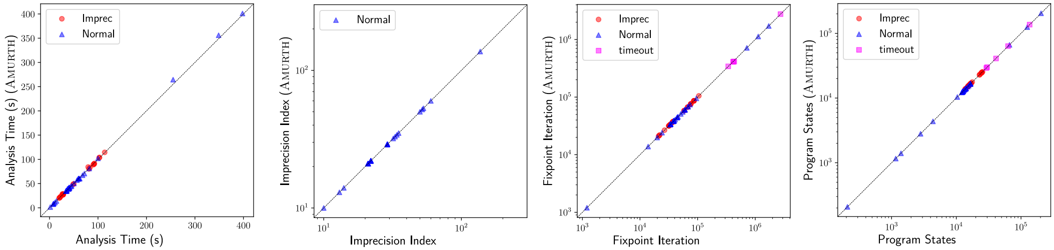


Comparison with manual operations



Similar performance as manually written transformers in terms of analysis time, imprecision index, fixpoint iteration, program states.

Comparison with manual operations



Similar performance as manually written transformers in terms of analysis time, imprecision index, fixpoint iteration, program states.

However, we discovered 4 soundness bugs in the manually written transformers.

Bug #1: contains in $\mathcal{CI}$

- $\mathcal{CI} = \{\bot_{\mathcal{CI}}\} \cup \{[L, U] \mid L, U \subseteq \Sigma, L \subseteq U\}$

Bug #1: `contains` in $\mathcal{CI}$

- $\mathcal{CI} = \{\bot_{\mathcal{CI}}\} \cup \{[L, U] \mid L, U \subseteq \Sigma, L \subseteq U\}$
- L : *must* set
  U : *may* set

Bug #1: contains in $\mathcal{CI}$

- $\mathcal{CI} = \{\bot_{\mathcal{CI}}\} \cup \{[L, U] \mid L, U \subseteq \Sigma, L \subseteq U\}$
- L : *must* set
  U : *may* set
- $\alpha_{\mathcal{CI}}(\{\text{“}fan\text{”}, \text{“}ran\text{”}\}) = [L : \{'a'\}, U : \{'f', 'n', 'r'\}]$

Bug #1: contains in $\mathcal{CI}$

$$\texttt{contains(str1, str2)} = \begin{cases} \top & \text{if } \texttt{str2} \text{ is contiguous substring of } \texttt{str1} \\ \bot & \text{otherwise} \end{cases}$$

Manually written transformer

```
contains#_org(a1 : CI)(a2 : CI) : AbsBool =
  ite(isBot(a1.l, a1.u) ∨ isBot(a2.l, a2.u),
    boolBot,
    ite(isTop(a1.l, a1.u) ∨ isTop(a2.l, a2.u),
      boolTop,
      ite(¬isSubset(a2.l, a1.u),
        boolFalse,
          ite(size(a2.u) ≤ 1 ∧ isSubset(a2.u, a1.l),
            boolTrue,
            boolTop))))
```

Synthesized abstract transformer

```
contains#_syn(a1 : CI)(a2 : CI) : AbsBool =
  ite(isBot(a1.l, a1.u) ∨ isBot(a2.l, a2.u),
    boolBot,


      ite(¬isSubset(a2.l, a1.u),
        boolFalse,
          ite(isEmpty(a2),
            boolTrue,
            boolTop)))
```

Bug #1: contains in $\mathcal{CI}$

$$\texttt{contains(str1, str2)} = \begin{cases} \top & \text{if } \texttt{str2} \text{ is contiguous substring of } \texttt{str1} \\ \bot & \text{otherwise} \end{cases}$$

Manually written transformer

```
contains♯_org(a1 : CI)(a2 : CI) : AbsBool =
  ite(isBot(a1.l, a1.u) ∨ isBot(a2.l, a2.u),
    boolBot,
    ite(isTop(a1.l, a1.u) ∨ isTop(a2.l, a2.u),
      boolTop,
      ite(¬isSubset(a2.l, a1.u),
        boolFalse,
        ite(size(a2.u) ≤ 1 ∧ isSubset(a2.u, a1.l),
          boolTrue,
          boolTop))))
```

Synthesized abstract transformer

```
contains♯_syn(a1 : CI)(a2 : CI) : AbsBool =
  ite(isBot(a1.l, a1.u) ∨ isBot(a2.l, a2.u),
    boolBot,


      ite(¬isSubset(a2.l, a1.u),
        boolFalse,
        ite(isEmpty(a2),
          boolTrue,
          boolTop)))
```

s_1="aa", s_2="aaaaa"

a_1=[{'a'}, {'a'}], a_2=[{'a'}, {'a'}]

Conclusions

- Current techniques at handling such operations are either highly imprecise, unsound, or manual and error-prone.

Conclusions

- Current techniques at handling such operations are either highly imprecise, unsound, or manual and error-prone.
- Our tool, AMURTH, is capable of automatically synthesizing non-trivial abstract transformers

Conclusions

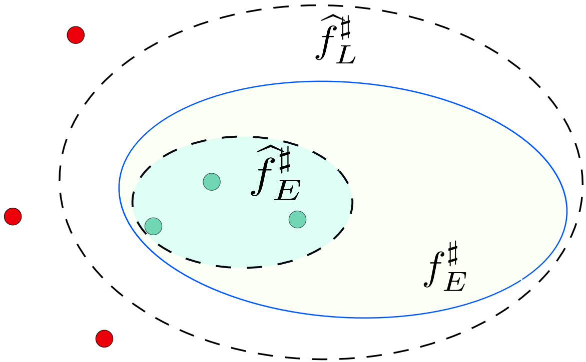
- Current techniques at handling such operations are either highly imprecise, unsound, or manual and error-prone.
- Our tool, AMURTH, is capable of automatically synthesizing non-trivial abstract transformers
- Our experiments on the existing tools shows the value of such an endeavour.

I thank Google for the generous travel grant that allowed me to attend SPLASH 2022.

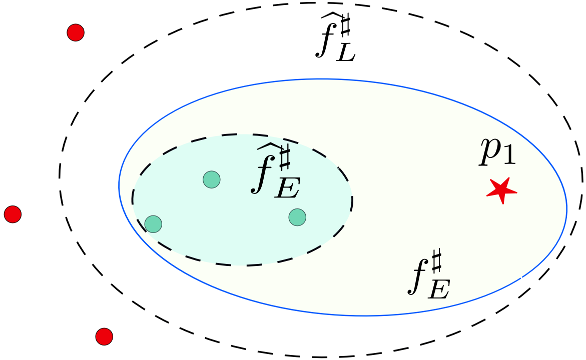
Thank you!

Backup slides

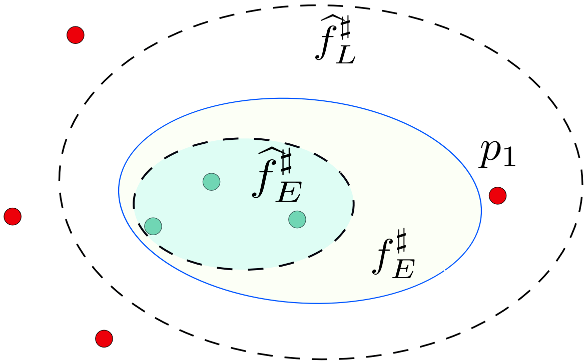
Failed consistency



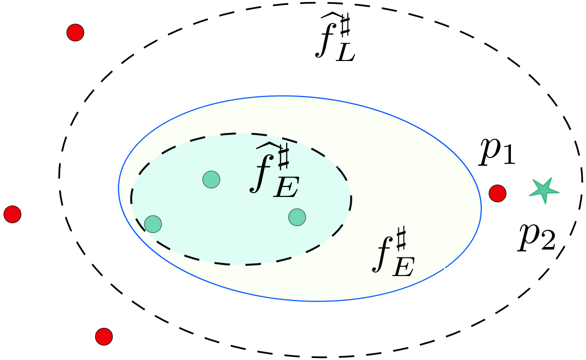
Failed consistency



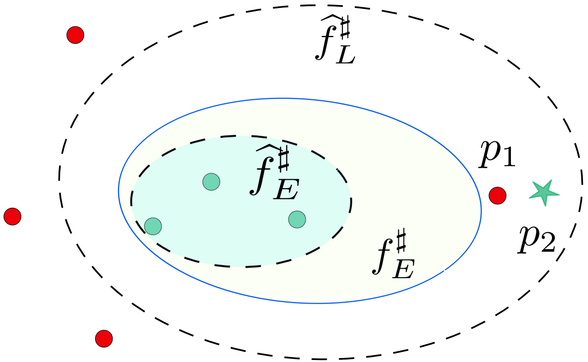
Failed consistency



Failed consistency

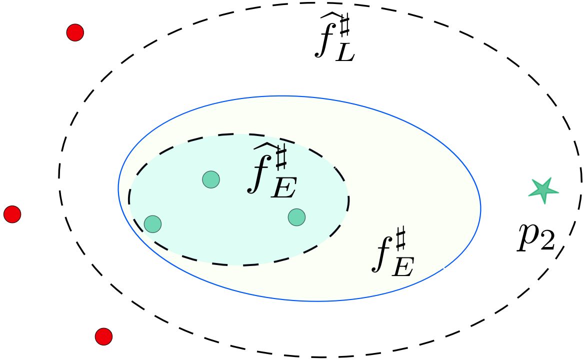


Failed consistency



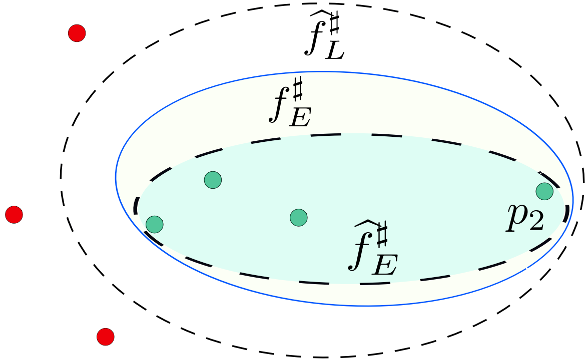
Inconsistent: no $f_E^\sharp \in L$ that satisfies all positive and negative examples.

Failed consistency



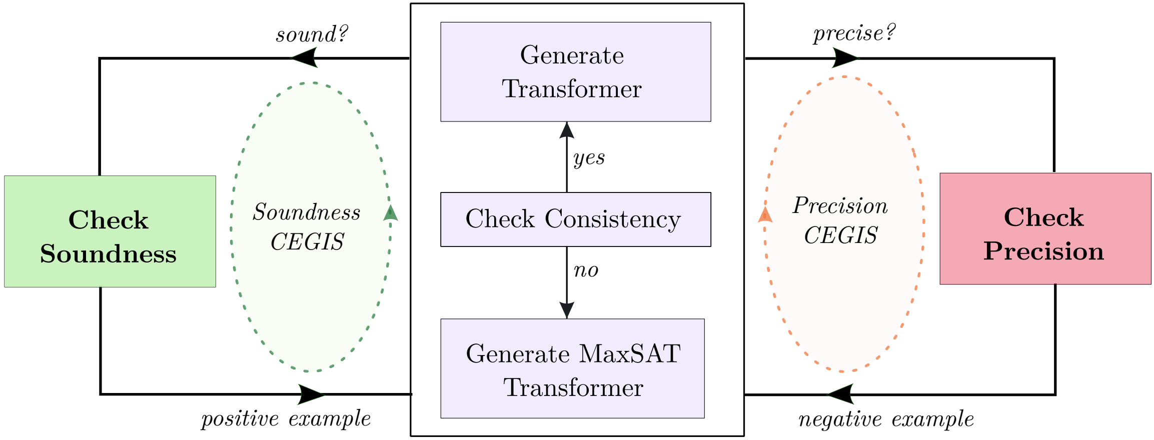
Occam's razor

Failed consistency



Occam's razor

Algorithm


sound?
precise?
Generate Transformer
yes
Check Soundness
Soundness CEGIS
Check Consistency
Precision CEGIS
Check Precision
no
Generate MaxSAT Transformer
positive example
negative example

trim in $\mathcal{CI}$

Original (buggy) transformer in SAFE_{str}

```
trim♯_org(a : CI) : CI =
 ite(isBot(a.l,a.u),
   boolBot,
   ite(isTop(a.l,a.u),
     boolTop,
     ite(size(a.u)≤1∧containsSpace(a.u),
       [∅,∅],
       a )))
```

Synthesized abstract transformer

```
trim♯_syn(a : CI) : CI =
 ite(isBot(a.l,a.u),
   boolBot,
   ite(isTop(a.l,a.u),
     boolTop,
     ite(size(a.u)≤1∧containsSpace(a.u),
       [∅,∅],
       [removeSpace(a.l),a.u] )))
```

strs = {"␣␣abc␣␣", "a␣a"}

$$a = [\{\text{`␣'},\text{`a'}\},\ \{\text{`␣'},\text{`a'},\text{`b'},\text{`c'}\}]$$

$$trim^\sharp(a) = [\{\text{`a'}\},\ \{\text{`␣'},\text{`a'},\text{`b'},\text{`c'}\}]$$