Multivariant Assertion-based Guidance in Abstract Interpretation

Isabel Garcia-Contreras Jose F. Morales Manuel V. Hermenegildo

Frankfurt am Main, LOPSTR 2018, September 5, 2018

IMDEA Software Institute, and

Technical University of Madrid

Abstract interpretation

Simulates the execution of programs using abstract domains.

It guarantees:

- Analysis **termination**, provided that the domain meets some conditions.
- Results are **safe approximations** of the concrete semantics.

Abstract interpretation

Simulates the execution of programs using abstract domains.

It guarantees:

- Analysis **termination**, provided that the domain meets some conditions.
- Results are **safe approximations** of the concrete semantics.

Accurate and efficient approximations have been achieved with:

- clever **abstract domains**,
- **widening and narrowing** techniques,
- sophisticated **fixpoint algorithms**.

Providing User-guidance to the Analysis

Two problems that motivate allowing the user to guide the analyzer:

1. Automatic **approximations** may lead to **imprecise results**:
   - Desired **optimizations** cannot be applied.
   - **Assertions** cannot be **verified** ("false alarms").

   $\rightarrow$ Allow the user to provide the analyzer known properties by making **optional annotations** to regain **precision**.

Providing User-guidance to the Analysis

Two problems that motivate allowing the user to guide the analyzer:

1. Automatic **approximations** may lead to **imprecise results**:
   - Desired **optimizations** cannot be applied.
   - **Assertions** cannot be **verified** ("false alarms").

   $\rightarrow$ Allow the user to provide the analyzer known properties by making **optional annotations** to regain **precision**.

2. Analysis may require excessive **resources** (time or space):

   $\rightarrow$ Allow the user to provide the analyzer with suggestions to **speed up** fixpoint computation.

Previous work

We focus on the techniques that provide the programmer to optionally annotate program parts to **guide invariants inference**:

Astreè [ESOP '05] uses at program point:

- *asserts* with properties that have to be verified.
- *known facts* used to refine abstract state.

CiaoPP [ESOP '96] uses assertions that can be qualified with a status:

- *check*: meaning that it needs to be verified.
- *trust*: representing knowledge that the user guarantees to be true (beliefs).

There is no precise description of how **annotations interact** with fixpoint computation.

Previous work

We focus on the techniques that provide the programmer to optionally annotate program parts to **guide invariants inference**:

Astreè [ESOP '05] uses at program point:

- *asserts* with properties that have to be verified.
- *known facts* used to refine abstract state.

CiaoPP [ESOP '96] uses assertions that can be qualified with a status:

- *check*: meaning that it needs to be verified.
- *trust*: representing knowledge that the user guarantees to be true (beliefs).

There is no precise description of how **annotations interact** with fixpoint computation.

Our goals:

- **clarify the influence of annotations** on the analysis result.
- propose different **strategies to apply** such annotations during analysis.
- provide precise conditions for detecting when annotations may lead to *erroneous* analysis results.

The Ciao Assertion Language

Assertions express **abstractions of the behavior** of programs.

`pred` assertions (subset)

`:- [Status] pred` *Head* `[:` *Pre*`] [=>` *Post*`].`

- *Head*: predicate that the assertion applies to.
- *Pre*: properties about how the predicate is used.
- *Post*: properties that hold if Pre holds and the predicate succeeds.
- `Status` qualifies the meaning of assertions:
  - `check` (default): program specifications.
  - `trust`: assertions whose validity is guaranteed by the programmer.

The Ciao Assertion Language

Assertions express **abstractions of the behavior** of programs.

pred assertions (subset)

`:- [Status] pred` *Head* `[:` *Pre*`] [=>` *Post*`].`

- *Head*: predicate that the assertion applies to.
- *Pre*: properties about how the predicate is used.
- *Post*: properties that hold if Pre holds and the predicate succeeds.
- Status qualifies the meaning of assertions:
  - check (default): program specifications.
  - trust: assertions whose validity is guaranteed by the programmer.

```
:- trust pred fact(N, R)          => (int(N), R > 0).
:- trust pred fact(N, R) : N > 1 => even(R).
```

Using Trust Assertions

Trust assertions may be used to:

- **Regain precision** during analysis.

Using Trust Assertions

Trust assertions may be used to:

- **Regain precision** during analysis.

```
% (y > 0) % Analyzing with an intervals domain (non relational)
x = y + 2;
% (x > 2, y > 0)
z = x - y;
```

Using Trust Assertions

Trust assertions may be used to:

- **Regain precision** during analysis.

```
% (y > 0) % Analyzing with an intervals domain (non relational)
x = y + 2;
% (x > 2, y > 0)
z = x - y;
% (int(z), x > 2, y > 0)
```

Using Trust Assertions

Trust assertions may be used to:

- **Regain precision** during analysis.

```
% (y > 0) % Analyzing with an intervals domain (non relational)
x = y + 2;
% (x > 2, y > 0)
z = x - y;
% (int(z), x > 2, y > 0)
```

But we know `x = y + 2`.

Using Trust Assertions

Trust assertions may be used to:

- **Regain precision** during analysis.

```
1 % (y > 0) % Analyzing with an intervals domain (non relational)
2 x = y + 2;
3 % (x > 2, y > 0)
4 z = x - y;
5 % (int(z), x > 2, y > 0)
```

But we know `x = y + 2`.

```
1 % (y > 0)
2 x = y + 2;
3 % (x > 2, y > 0)
4 z = x - y;
5 % (int(z), x > 2, y > 0)
6 trust(z == 2); % Because of line 2
7 % (z = 2, x > 2, y > 0)
```

Using Trust Assertions

Trust assertions may be used to:

- **Regain precision** during analysis.
- **Speed up computation** of analysis.

Using Trust Assertions

Trust assertions may be used to:

- **Regain precision** during analysis.
- **Speed up computation** of analysis.

```
:- trust pred html_escape(S0, S) => (string(S0), string(S)).
html_escape( "‘‘"||S0 , "&ldquo;"||S ) :- !, html_escape(S0, S).
html_escape( "’’"||S0 , "&rdquo;"||S ) :- !, html_escape(S0, S).
html_escape( "’" ||S0 , "&quot;" ||S ) :- !, html_escape(S0, S).
html_escape( """"||S0 , "&apos;" ||S ) :- !, html_escape(S0, S).
html_escape(    [X|S0],        [X | S]) :- !, character_code(X),
                                             html_escape(S0, S).
html_escape(        "",              "").
```

Using Trust Assertions

Trust assertions may be used to:

- **Regain precision** during analysis.
- **Speed up computation** of analysis.
- Define **abstract usage or specifications** of libraries or dynamic predicates.

Using Trust Assertions

Trust assertions may be used to:

- **Regain precision** during analysis.
- **Speed up computation** of analysis.
- Define **abstract usage or specifications** of libraries or dynamic predicates.

```
:- module(sockets, []).

:- export(receive/2).
:- pred receive(S, M) : (socket(S), var(M)) => list(M, utf8).
:- impl_defined(receive/2).
% receive is written in C
```

Using Trust Assertions

Trust assertions may be used to:

- **Regain precision** during analysis.
- **Speed up computation** of analysis.
- Define **abstract usage or specifications** of libraries or dynamic predicates.
- (Re)define the **language semantics** for abstract domains (transfer funct.).

Using Trust Assertions

Trust assertions may be used to:

- **Regain precision** during analysis.
- **Speed up computation** of analysis.
- Define **abstract usage or specifications** of libraries or dynamic predicates.
- (Re)define the **language semantics** for abstract domains (transfer funct.).

```
:- trust pred '*'(A, B, C) : (int(A), int(B)) => int(C).
:- trust pred '*'(A, B, C) : (flt(A), int(B)) => flt(C).
:- trust pred '*'(A, B, C) : (int(A), flt(B)) => flt(C).
:- trust pred '*'(A, B, C) : (flt(A), flt(B)) => flt(C).
```

Target language - Horn Clauses

We perform abstract interpretation of **Horn Clauses**. The concrete semantics is **goal-dependent** and based on the notion of generalized AND trees:

- An AND tree represents the **execution of a query**.
- A **node** is a **call** to a predicate with:
  - Constraint state for that call.
  - Constraint state if the call **succeeds**.

We assume that programs have already been translated to Horn Clauses.

Analysis output

For all the predicates we obtain a set of tuples $\langle Goal, \lambda^c, \lambda^s \rangle$, where:

- *Goal* is an atom (identifier of the predicate).
- λ^c is a (possible) call pattern to *Goal*.
- λ^s is the answer pattern to *Goal* and λ^c if succeeds.

Analysis output

For all the predicates we obtain a set of tuples $\langle Goal, \lambda^c, \lambda^s \rangle$, where:

- *Goal* is an atom (identifier of the predicate).
- λ^c is a (possible) call pattern to *Goal*.
- λ^s is the answer pattern to *Goal* and λ^c if succeeds.

Example

```
fact(0,1).
fact(N,R) :- N > 0,
             N1 is N - 1,
             fact(N1,R1),
             R is N * R1.
```

Analysis output

For all the predicates we obtain a set of tuples $\langle Goal, \lambda^c, \lambda^s \rangle$, where:

- *Goal* is an atom (identifier of the predicate).
- λ^c is a (possible) call pattern to *Goal*.
- λ^s is the answer pattern to *Goal* and λ^c if succeeds.

Example

```
fact(0,1).
fact(N,R) :- N > 0,
             N1 is N - 1,
             fact(N1,R1),
             R is N * R1.
```

Analysis result:

$\{\langle \mathtt{fact(N,R)}, \top, R/+ \rangle$

For any call to `fact` *that succeeds R is positive.*

$\langle \mathtt{fact(N,F)}, N/-, \perp \rangle$

If `fact` *is called with N a negative number, it fails.* $\}$

Analysis output

For all the predicates we obtain a set of tuples $\langle Goal, \lambda^c, \lambda^s \rangle$, where:

- *Goal* is an atom (identifier of the predicate).
- λ^c is a (possible) call pattern to *Goal*.
- λ^s is the answer pattern to *Goal* and λ^c if succeeds.

Example

```
fact(0,1).
fact(N,R) :- N > 0,
             N1 is N - 1,
             fact(N1,R1),
             R is N * R1.
```

Analysis result:

$\{\langle \mathtt{fact(N,R)}, \top, R/+ \rangle$

For any call to `fact` *that succeeds R is positive.*

$\langle \mathtt{fact(N,F)}, N/-, \bot \rangle$

If `fact` *is called with N a negative number, it fails.* $\}$

Analysis is **multivariant** and **context sensitive**.

Analysis output

For all the predicates we obtain a set of tuples $\langle Goal, \lambda^c, \lambda^s \rangle$, where:

- *Goal* is an atom (identifier of the predicate).
- λ^c is a (possible) call pattern to *Goal*.
- λ^s is the answer pattern to *Goal* and λ^c if succeeds.

Example

```
fact(0,1).
fact(N,R) :- N > 0,
             N1 is N - 1,
             fact(N1,R1),
             R is N * R1.
```

Analysis result:

$\{\langle \mathtt{fact(N,R)}, \top, R/+ \rangle$

For any call to `fact` *that succeeds R is positive.*

$\langle \mathtt{fact(N,F)}, N/-, \bot \rangle$

If `fact` *is called with N a negative number, it fails.* $\}$

Analysis is **multivariant** and **context sensitive**.

Accessing the analysis results:

- look up: $\underline{\lambda^s = a[H, \lambda^c]}$ iff $\langle H, \lambda^c, \lambda^s \rangle \in A$,
- update: $\underline{a[H, \lambda^c] \leftarrow \lambda^{s'}}$ removes $\langle H, \lambda^c, \_ \rangle$ from A and inserts $\langle H, \lambda^c, \lambda^{s'} \rangle$.

Intuition of the fixpoint algorithm

Simplified version of **PLAI** [NACLP '89] (removed optimizations and path-sensitivity).

Analyze$(\mathcal{Q}_\alpha, P)$

input global: $\mathcal{Q}_\alpha$ (initial abstract queries), **P** (program)
output global: **A**, analysis result

$a[L_i, \lambda_i] \leftarrow \bot$ **for all** $L_i : \lambda_i \in \mathcal{Q}_\alpha$, *changes* $\leftarrow$ true

Intuition of the fixpoint algorithm

Simplified version of **PLAI** [NACLP '89] (removed optimizations and path-sensitivity).

Analyze($\mathcal{Q}_\alpha, P$)

input global: $\mathcal{Q}_\alpha$ (initial abstract queries), **P** (program)
output global: **A**, analysis result

$a[L_i, \lambda_i] \leftarrow \bot$ **for all** $L_i : \lambda_i \in \mathcal{Q}_\alpha$, *changes* $\leftarrow$ true
while *changes* **do**
    changes $\leftarrow$ false
    W = get_tuples_to_update()

Intuition of the fixpoint algorithm

Simplified version of **PLAI** [NACLP '89] (removed optimizations and path-sensitivity).

Analyze$(\mathcal{Q}_\alpha, P)$

input global: $\mathcal{Q}_\alpha$ (initial abstract queries), **P** (program)
output global: **A**, analysis result

$a[L_i, \lambda_i] \leftarrow \bot$ **for all** $L_i : \lambda_i \in \mathcal{Q}_\alpha$, *changes* $\leftarrow$ true
while *changes* **do**
  changes $\leftarrow$ false
  $W =$ get_tuples_to_update()
  for each $(G, \lambda^c, \text{cl}) \in W$ **do**
    $\lambda^t \leftarrow$ abs_call$(G, \lambda^c, \text{cl.head})$
    $\lambda^t \leftarrow$ solve_body$(\text{cl.body}, \lambda^t)$
    $\lambda^{s_0} \leftarrow$ abs_proceed$(G, \text{cl.head}, \lambda^t)$

Intuition of the fixpoint algorithm

Simplified version of **PLAI** [NACLP '89] (removed optimizations and path-sensitivity).

Analyze($\mathcal{Q}_\alpha, P$)

input global: $\mathcal{Q}_\alpha$ (initial abstract queries), **P** (program)
output global: **A**, analysis result

$a[L_i, \lambda_i] \leftarrow \bot$ **for all** $L_i : \lambda_i \in \mathcal{Q}_\alpha$, *changes* $\leftarrow$ true
while *changes* **do**
    changes $\leftarrow$ false
    $W = \text{get\_tuples\_to\_update}()$
    for each $(G, \lambda^c, \text{cl}) \in W$ **do**
        $\lambda^t \leftarrow \texttt{abs\_call}(G, \lambda^c, \text{cl.head})$
        $\lambda^t \leftarrow \texttt{solve\_body}(\text{cl.body}, \lambda^t)$
        $\lambda^{s_0} \leftarrow \texttt{abs\_proceed}(G, \text{cl.head}, \lambda^t)$
        $\lambda^{s'} \leftarrow \texttt{abs\_generalize}(\lambda^{s_0}, \{a[G, \lambda^c]\})^\bullet$
        if $\lambda^{s'} \neq a[G, \lambda^c]$ **then**
            $a[G, \lambda^c] \leftarrow \lambda^{s'}$, *changes* $\leftarrow$ true

$\bullet$ *includes* $\sqcup$ *(lub) and widening*

Intuition of the fixpoint algorithm

Simplified version of **PLAI** [NACLP '89] (removed optimizations and path-sensitivity).

Analyze$(\mathcal{Q}_\alpha, P)$

input global: $\mathcal{Q}_\alpha$ (initial abstract queries), **P** (program)
output global: **A**, analysis result

$a[L_i, \lambda_i] \leftarrow \bot$ **for all** $L_i : \lambda_i \in \mathcal{Q}_\alpha$, *changes* $\leftarrow$ true
while *changes* **do**
    changes $\leftarrow$ false
    W = get_tuples_to_update()
    for each $(G, \lambda^c, \mathrm{cl}) \in W$ **do**
        $\lambda^t \leftarrow$ abs_call$(G, \lambda^c, \mathrm{cl.head})$
        $\lambda^t \leftarrow$ solve_body(cl.body, λ^t)
        $\lambda^{s_0} \leftarrow$ abs_proceed$(G, \mathrm{cl.head}, \lambda^t)$
        $\lambda^{s'} \leftarrow$ abs_generalize$(\lambda^{s_0}, \{a[G, \lambda^c]\})^\bullet$
        if $\lambda^{s'} \neq a[G, \lambda^c]$ **then**
            $a[G, \lambda^c] \leftarrow \lambda^{s'}$, *changes* $\leftarrow$ true

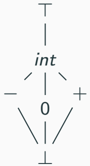
$\bullet$ *includes* $\sqcup$ *(lub) and widening*

function solve_body(B, λ^t)
    for each $L \in B$ **do**
        $\lambda^c \leftarrow$ abs_project(L, λ^t)
        Calls = get_calls_to_pred(L)
        $\lambda^{c'} \leftarrow$ abs_generalize$(\lambda^c, \mathit{Calls})^\bullet$
        $\lambda^s \leftarrow$ solve$(L, \lambda^{c'})$
        $\lambda^t \leftarrow$ abs_extend$(L, \lambda^s, \lambda^t)$
    return λ^t

Analyzing factorial

$\mathcal{Q}_\alpha = \{\mathtt{fact(N,R)} : \mathtt{int(N)}\}$

```
fact(0,1).
fact(N,R) :- N > 0,
             N1 is N - 1,
             fact(N1, R1),
             R is N * R1.
```



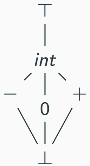
Action	$\lambda^c(\mathtt{fact}(N,R))$	λ^t	$\lambda^s(\mathtt{fact}(N,R))$
init	$N/int, R/\top$	-	-

Analysis:
$\langle \mathtt{fact}(N,R), (N/int, R/\top), \bot \rangle$

Analyzing factorial

$\mathcal{Q}_\alpha = \{\mathtt{fact(N,R)} : \mathtt{int(N)}\}$

```
1 fact(0,1).                  % <-- analyzing
2 fact(N,R) :- N > 0,
3              N1 is N - 1,
4              fact(N1, R1),
5              R is N * R1.
```



Action	$\lambda^c(\mathtt{fact}(N,R))$	λ^t	$\lambda^s(\mathtt{fact}(N,R))$
init	$N/int, R/\top$	-	-
it 1 (l. 1)	$N/int, R/\top$	-	$N/0, R/+$

Analysis:

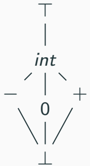
~~$\langle \mathtt{fact}(N,R), (N/int, R/\top), \bot \rangle$~~

$\langle \mathtt{fact}(N,R), (N/int, R/\top), (N/0, R/+) \rangle$

Analyzing factorial

$\mathcal{Q}_\alpha = \{\texttt{fact(N,R)} : \texttt{int(N)}\}$

```
1 fact(0,1).
2 fact(N,R) :- N > 0,     % <-- analyzing
3              N1 is N - 1,
4              fact(N1, R1),
5              R is N * R1.
```



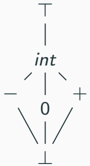
Action	$\lambda^c(\texttt{fact}(N,R))$	λ^t	$\lambda^s(\texttt{fact}(N,R))$
init	$N/int, R/\top$	-	-
it 1 (l. 1)	$N/int, R/\top$	-	$N/0, R/+$
(l. 2)	$N/int, R/\top$	$N/+, N1/0, R1/+$	$N/+, R/+$

Analysis:
$\cancel{\langle \texttt{fact}(N,R), (N/int, R/\top), \perp \rangle}$
$\langle \texttt{fact}(N,R), (N/int, R/\top), (N/0, R/+) \rangle$

Analyzing factorial

$\mathcal{Q}_\alpha = \{\mathtt{fact(N,R)} : \mathtt{int(N)}\}$

```
1 fact(0,1).
2 fact(N,R) :- N > 0,     % <-- analyzing
3              N1 is N - 1,
4              fact(N1, R1),
5              R is N * R1.
```



Action	$\lambda^c(\mathtt{fact}(N,R))$	λ^t	$\lambda^s(\mathtt{fact}(N,R))$
init	$N/int, R/\top$	-	-
it 1 (l. 1)	$N/int, R/\top$	-	$N/0, R/+$
(l. 2)	$N/int, R/\top$	$N/+, N1/0, R1/+$	$N/+, R/+$
store $\sqcup$	$N/int, R/\top$	-	$N/int, R/+$

Analysis:

$\cancel{\langle \mathtt{fact}(N,R), (N/int, R/\top), \bot \rangle}$

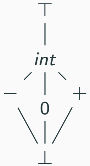
$\cancel{\langle \mathtt{fact}(N,R), (N/int, R/\top), (N/0, R/+) \rangle}$

$\langle \mathtt{fact}(N,R), (N/int, R/\top), (N/int, R/+) \rangle$

Analyzing factorial

$\mathcal{Q}_\alpha = \{\texttt{fact(N,R)} : \texttt{int(N)}\}$

```
1 fact(0,1).
2 fact(N,R) :- N > 0,     % <-- analyzing
3              N1 is N - 1,
4              fact(N1, R1),
5              R is N * R1.
```



Action	$\lambda^c(\texttt{fact}(N,R))$	λ^t	$\lambda^s(\texttt{fact}(N,R))$
init	$N/int, R/\top$	-	-
it 1 (l. 1)	$N/int, R/\top$	-	$N/0, R/+$
(l. 2)	$N/int, R/\top$	$N/+, N1/0, R1/+$	$N/+, R/+$
store $\sqcup$	$N/int, R/\top$	-	$N/int, R/+$
it 2 (l. 2)	$N/int, R/\top$	-	$N/int, R/+$

Analysis:

$\cancel{\langle \texttt{fact}(N,R), (N/int, R/\top), \bot\rangle}$

$\cancel{\langle \texttt{fact}(N,R), (N/int, R/\top), (N/0, R/+)\rangle}$

$\langle \texttt{fact}(N,R), (N/int, R/\top), (N/int, R/+)\rangle$

Meaning of a set of assertions for a predicate

Assertion Conditions

Given a predicate represented by a normalized atom *Head*, and a corresponding set of assertions $\mathcal{A} = \{A_1 \ldots A_n\}$, with $A_i =$ "`:- pred` $Head : Pre_i$ `=>` $Post_i$`.`". The set of assertion conditions for *Head* determined by $\mathcal{A}$ is $\{C_0, C_1, \ldots, C_n\}$, with:

$$C_i = \begin{cases} \texttt{calls}(Head, \bigvee_{j=1}^{n} Pre_j) & i = 0 \\ \texttt{success}(Head, Pre_i, Post_i) & i = 1..n \end{cases}$$

```
:- trust pred fact(N, R)         => (int(N), R > 0).
:- trust pred fact(N, R) : N > 1 => even(R).
```

Assertion conditions from `fact/2`:

$$C_i = \left\{ \begin{array}{llll} \texttt{calls(} & fact(N,R), & (true \vee N > 1)), & \\ \texttt{success(} & fact(N,R), & true & , (\mathrm{int}(N), R > 0)), \\ \texttt{success(} & fact(N,R), & N > 1 & , \mathrm{even}(R)), \end{array} \right\}$$

Intuition of the fixpoint algorithm with guidance

GuidedAnalyze$(\mathcal{Q}_\alpha, P)$

input global: $\mathcal{Q}_\alpha$ (initial abstract queries), **P** (program).
output global: **A**, **E** (analysis-like tuple set – to capture **user errors**)

$a[L_i, \lambda_i] \leftarrow \perp$ **for all** $L_i : \lambda_i \in \mathcal{Q}_\alpha$, *changes* $\leftarrow$ true
$E \leftarrow \emptyset$ ▷ new!

Intuition of the fixpoint algorithm with guidance

GuidedAnalyze$(\mathcal{Q}_\alpha, P)$

input global: $\mathcal{Q}_\alpha$ (initial abstract queries), **P** (program).
output global: **A**, **E** (analysis-like tuple set – to capture **user errors**)

$a[L_i, \lambda_i] \leftarrow \bot$ **for all** $L_i : \lambda_i \in \mathcal{Q}_\alpha$, *changes* $\leftarrow$ true
$E \leftarrow \emptyset$ ▷ new!
while *changes* **do**
    changes $\leftarrow$ false
    W = get_tuples_to_update()

Intuition of the fixpoint algorithm with guidance

GuidedAnalyze($\mathcal{Q}_\alpha, P$)

input global: $\mathcal{Q}_\alpha$ (initial abstract queries), **P** (program).
output global: **A**, **E** (analysis-like tuple set – to capture **user errors**)

$a[L_i, \lambda_i] \leftarrow \bot$ **for all** $L_i : \lambda_i \in \mathcal{Q}_\alpha$, $changes \leftarrow$ true
$E \leftarrow \emptyset$ ▷ new!
while *changes* **do**
    $changes \leftarrow$ false
    $W =$ get_tuples_to_update()
    for each $(G, \lambda^c, \text{cl}) \in W$ **do**
        $\lambda^t \leftarrow \texttt{abs\_call}(G, \lambda^c, \text{cl.head})$
        $\lambda^t \leftarrow \texttt{solve\_body}(\text{cl.body}, \lambda^t)$
        $\lambda^{s0} \leftarrow \texttt{abs\_proceed}(G, \text{cl.head}, \lambda^t)$

Intuition of the fixpoint algorithm with guidance

GuidedAnalyze($\mathcal{Q}_\alpha, P$)

input global: $\mathcal{Q}_\alpha$ (initial abstract queries), **P** (program).
output global: **A**, **E** (analysis-like tuple set – to capture **user errors**)

$a[L_i, \lambda_i] \leftarrow \bot$ **for all** $L_i : \lambda_i \in \mathcal{Q}_\alpha$, *changes* $\leftarrow$ true
$E \leftarrow \emptyset$ ▷ new!
while *changes* **do**
    changes $\leftarrow$ false
    W = get_tuples_to_update()
    for each $(G, \lambda^c, \text{cl}) \in W$ **do**
        $\lambda^t \leftarrow$ `abs_call`$(G, \lambda^c, \text{cl.head})$
        $\lambda^t \leftarrow$ `solve_body`$(\text{cl.body}, \lambda^t)$
        $\lambda^{s0} \leftarrow$ `abs_proceed`$(G, \text{cl.head}, \lambda^t)$
        $\lambda^{s'} \leftarrow$ `apply_succ`$(G, \lambda^c, \lambda^{s0}, a[G, \lambda^c])^\bullet$ ▷ new!
        if $\lambda^{s'} \neq a[G, \lambda^c]$ **then**
            $a[G, \lambda^c] \leftarrow \lambda^{s'}$, *changes* $\leftarrow$ true

• *includes* $\sqcup$ *(lub) and widening*

Intuition of the fixpoint algorithm with guidance

GuidedAnalyze$(\mathcal{Q}_\alpha, P)$

input global: $\mathcal{Q}_\alpha$ (initial abstract queries), **P** (program).
output global: **A**, **E** (analysis-like tuple set – to capture **user errors**)

$a[L_i, \lambda_i] \leftarrow \bot$ **for all** $L_i : \lambda_i \in \mathcal{Q}_\alpha$, *changes* $\leftarrow$ true
$E \leftarrow \emptyset$ ▷ new!
while *changes* **do**
    changes $\leftarrow$ false
    W = get_tuples_to_update()
    for each $(G, \lambda^c, \text{cl}) \in W$ **do**
        $\lambda^t \leftarrow$ `abs_call`$(G, \lambda^c, \text{cl.head})$
        $\lambda^t \leftarrow$ `solve_body`$(\text{cl.body}, \lambda^t)$
        $\lambda^{s_0} \leftarrow$ `abs_proceed`$(G, \text{cl.head}, \lambda^t)$
        $\lambda^{s'} \leftarrow$ `apply_succ`$(G, \lambda^c, \lambda^{s_0}, a[G, \lambda^c])^\bullet$ ▷ new!
        if $\lambda^{s'} \neq a[G, \lambda^c]$ **then**
            $a[G, \lambda^c] \leftarrow \lambda^{s'}$, *changes* $\leftarrow$ true

• *includes* $\sqcup$ *(lub) and widening*

function `solve_body`(B, λ^t)
    for each $L \in B$ **do**
        $\lambda^c \leftarrow$ `abs_project`(L, λ^t)
        $\lambda^{c'} \leftarrow$ `apply_call`$(L, \lambda^c)^\bullet$ ▷ new!
        $\lambda^s \leftarrow$ `solve`$(L, \lambda^{c'})$
        $\lambda^t \leftarrow$ `abs_extend`$(L, \lambda^s, \lambda^t)$
    return λ^t

Intuition of the fixpoint algorithm with guidance

GuidedAnalyze($\mathcal{Q}_\alpha, P$)

input global: $\mathcal{Q}_\alpha$ (initial abstract queries), **P** (program).
output global: **A**, **E** (analysis-like tuple set – to capture **user errors**)

$a[L_i, \lambda_i] \leftarrow \bot$ **for all** $L_i : \lambda_i \in \mathcal{Q}_\alpha$, *changes* $\leftarrow$ true
$E \leftarrow \emptyset$ ▷ new!
while *changes* **do**
  changes $\leftarrow$ false
  W = get_tuples_to_update()
  for each $(G, \lambda^c, \mathrm{cl}) \in W$ **do**
    $\lambda^t \leftarrow$ `abs_call`$(G, \lambda^c, \mathrm{cl.head})$
    $\lambda^t \leftarrow$ `solve_body`$(\mathrm{cl.body}, \lambda^t)$
    $\lambda^{s_0} \leftarrow$ `abs_proceed`$(G, \mathrm{cl.head}, \lambda^t)$
    $\lambda^{s'} \leftarrow$ `apply_succ`$(G, \lambda^c, \lambda^{s_0}, a[G, \lambda^c])^\bullet$ ▷ new!
    if $\lambda^{s'} \neq a[G, \lambda^c]$ **then**
      $a[G, \lambda^c] \leftarrow \lambda^{s'}$, *changes* $\leftarrow$ true

function `solve_body`(B, λ^t)
  for each $L \in B$ **do**
    $\lambda^c \leftarrow$ `abs_project`(L, λ^t)
    $\lambda^{c'} \leftarrow$ `apply_call`$(L, \lambda^c)^\bullet$ ▷ new!
    $\lambda^s \leftarrow$ `solve`$(L, \lambda^{c'})$
    $\lambda^t \leftarrow$ `abs_extend`$(L, \lambda^s, \lambda^t)$
  return λ^t

• *includes* $\sqcup$ *(lub) and widening*

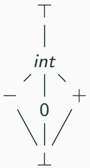
E is used to find **incompatibilities** between assertions and the information inferred (online or offline).

Assertions may be applied to **regain precision** or to **gain performance**.

Analyzing factorial with guidance

$\mathcal{Q}_\alpha = \{\mathtt{fact(N,R)} : \mathtt{int(N)}\}$

```
:- trust pred fact(N,R) : int(N) => (int(N), R > 0).
fact(0,1).
fact(N,R) :- N > 0,
             N1 is N - 1,
             fact(N1, R1),
             R is N * R1.
```



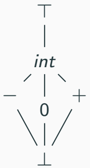
Action	$\lambda^c(\mathtt{fact}(N,R))$	λ^t	$\lambda^s(\mathtt{fact}(N,R))$
init	$N/\mathit{int}, R/\top$	-	-

Analysis:
$\langle \mathtt{fact}(N,R), (N/\mathit{int}, R/\top), \perp \rangle$

Analyzing factorial with guidance

$\mathcal{Q}_\alpha = \{\texttt{fact(N,R)} : \texttt{int(N)}\}$

```
1 :- trust pred fact(N,R) : int(N) => (int(N), R > 0).
2 fact(0,1).                 % <-- analyzing
3 fact(N,R) :- N > 0,
4              N1 is N - 1,
5              fact(N1, R1),
6              R is N * R1.
```



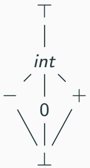
Action	$\lambda^c(\text{fact}(N,R))$	λ^t	$\lambda^s(\text{fact}(N,R))$
init	$N/int, R/\top$	-	-
it 1 (l. 2)	$N/int, R/\top$	-	$N/int, R/+$

Analysis:
$\cancel{\langle \text{fact}(N,R), (N/int, R/\top), \bot \rangle}$
$\langle \text{fact}(N,R), (N/int, R/\top), (N/int, R/+) \rangle$

Analyzing factorial with guidance

$\mathcal{Q}_\alpha = \{\texttt{fact(N,R)} : \texttt{int(N)}\}$

```
1 :- trust pred fact(N,R) : int(N) => (int(N), R > 0).
2 fact(0,1).
3 fact(N,R) :- N > 0,      % <-- analyzing
4              N1 is N - 1,
5              fact(N1, R1),
6              R is N * R1.
```



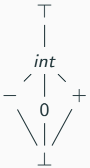
Action	$\lambda^c(\mathrm{fact}(N,R))$	λ^t	$\lambda^s(\mathrm{fact}(N,R))$
init	$N/int, R/\top$	-	-
it 1 (l. 2)	$N/int, R/\top$	-	$N/int, R/+$
(l. 3)	$N/int, R/\top$	$N/+, N1/int, R1/+$	$N/int, R/+$

Analysis:
$\cancel{\langle \mathrm{fact}(N,R), (N/int, R/\top), \bot \rangle}$
$\langle \mathrm{fact}(N,R), (N/int, R/\top), (N/int, R/+) \rangle$

Analyzing factorial with guidance

$\mathcal{Q}_\alpha = \{\texttt{fact(N,R)} : \texttt{int(N)}\}$

```
1 :- trust pred fact(N,R) : int(N) => (int(N), R > 0).
2 fact(0,1).
3 fact(N,R) :- N > 0,      % <-- analyzing
4               N1 is N - 1,
5               fact(N1, R1),
6               R is N * R1.
```



Action	$\lambda^c(\texttt{fact}(N,R))$	λ^t	$\lambda^s(\texttt{fact}(N,R))$
init	$N/int, R/\top$	-	-
it 1 (l. 2)	$N/int, R/\top$	-	$N/int, R/+$
(l. 3)	$N/int, R/\top$	$N/+, N1/int, R1/+$	$N/int, R/+$

Analysis:
$\cancel{\langle \texttt{fact}(N,R), (N/int, R/\top), \bot \rangle}$
$\langle \texttt{fact}(N,R), (N/int, R/\top), (N/int, R/+) \rangle$

One step less!

Fundamental results

Assertions are correctly applied during analysis.

Lemma (Applied success conditions)

The abstract success states inferred are covered by the success assertion conditions (if exist), i.e., there are no inferred states that escape the annotated assertions:

$\forall \langle L, \lambda^c, \lambda^s \rangle \in A$, $\texttt{success}(H, Pre, Post) \in C$ *s.t.* $L = \sigma(H)$

$\lambda^c \sqsupseteq \lambda^-_{TS(\sigma(Pre),P)} \Rightarrow \lambda^s \sqsubseteq \lambda^+_{TS(\sigma(Post),P)}$.

Lemma (Applied call conditions)

The abstract call states inferred are covered by the call assertion conditions:

$\forall \langle L, \lambda^c, \lambda^s \rangle \in A$, $\texttt{calls}(H, Pre) \in C, \exists\ \sigma\ L = \sigma(H) \Rightarrow \lambda^c \sqsubseteq \lambda^+_{TS(\sigma(Pre),P)}$.

Correct trusted assertions

Correctness of assertions with respect to the **concrete semantics**:

Definition (Correct trust call conditions)

A condition calls(H, Pre) is correct if
$\forall L \in P, \forall \theta^c \in calling\_context(L, P, \mathcal{Q}) \exists\ \sigma, L = \sigma(H) \Rightarrow \theta^c \in \gamma(\lambda^+_{TS(Pre,P)})$

Definition (Correct trust success conditions)

A condition success$(H, Pre, Post)$ is correct if
$\forall L \in P, \theta^c \in \gamma(\lambda^-_{TS(Pre,P)}), \exists\ \sigma, L = \sigma(H),\ L : \theta^c$ succeeds in P with $\theta^s \Rightarrow \theta^s \in \gamma(\lambda^+_{TS(Post,P)})$.

Correctness modulo assertions

Theorem (Correctness modulo assertions)

Given a program P with assertion conditions C and $\mathcal{Q}_\alpha$ a set of initial abstract queries. Let $\mathcal{Q}$ be the set of concrete queries:
$\mathcal{Q} = \{L\theta \mid \theta \in \gamma(\lambda) \wedge L : \lambda \in \mathcal{Q}_\alpha\}$.

The computed analysis $A = \{\langle L_1, \lambda_1^c, \lambda_1^s\rangle, \ldots, \langle L_n, \lambda_n^c, \lambda_n^s\rangle\}$ for P with $\mathcal{Q}_\alpha$ is correct for $P, \mathcal{Q}$ if all conditions are correct.

Detecting potential user errors

We compare the tuples of E with the description in assertions:

- `calls` conditions with the λ^c
- `success` conditions with the λ^s if Pre is applicable (to λ^c).

In general, let:

- λ be the value in a tuple in E.
- λ^a be the value in the assertion condition.

Use the tuples from E

Warnings: If $\lambda \sqsupset \lambda^a$ means that applying the assertion (refines the state) eliminates possible abstract states.

Errors: If $\lambda \sqcap \lambda^a = \bot$ it means that the inferred information is **incompatible** with the condition.

Summary

Ciao assertions can be used to *regain analysis precision, speed up analysis computation, describe external code, ... + provide specifications*. They:

- Can talk about **call/success states at procedure** (predicate) level.
- Can express **multi-variance**, i.e., refer to several call/success situations.

Summary

Ciao assertions can be used to *regain analysis precision, speed up analysis computation, describe external code, ... + provide specifications*. They:

- Can talk about **call/success states at procedure** (predicate) level.
- Can express **multi-variance**, i.e., refer to several call/success situations.

We have:

- Provided an **algorithm** to apply correctly such assertions.
- Provided means to detect **incompatible** trust assertions.

- Also, extended the semantics to cover several **run-time** behaviors (see paper).

Thanks!

Run-time behaviors

Status	Use in analyzer	Run-time test (if not discharged at compile-time)
`trust`	yes	no (believe and report)
`check`	yes	yes
`sample-check`	no	optional

Applying conditions during fixpoint

Condition type	λ vs Cond	Use	Debug
`calls`$(Head, Pre)$	$\lambda^c = Pre$	any	–
	$\lambda^c \sqsupset Pre$	λ^c	warning
	$\lambda^c \sqsubset Pre$	Pre	–
	$\lambda^c \sqcap Pre = \bot$	error	error
	$\lambda^c \sqcap Pre \neq \bot$	$\lambda^c \sqcap Pre$ (general)	warning
Applicable	$\lambda^c = Pre$	yes	
Pre of	$\lambda^c \sqsupset Pre$	no (over-approx.)	
`succ`$(Head, Pre, Post)$	$\lambda^c \sqsubset Pre$	yes	
	$\lambda^c \sqcap Pre = \bot$	no	
	$\lambda^c \sqcap Pre \neq \bot$	no	
`succ`	$\lambda^s = Post$	any	–
(if applicable)	$\lambda^s \sqsupset Pre$	*Post*	warning
	$\lambda^s \sqsubset Post$	λ^s	–
	$\lambda^s \sqcap Post = \bot$	error	error
	$\lambda^s \sqcap Post \neq \bot$	$\lambda^s \sqcap Post$ (general)	warning

Naive top-down fixpoint algorithm – full

Analyze$(\mathcal{Q}_\alpha, P)$

output global: A

$a[L_i, \lambda_i] \leftarrow \bot$ **for all** $L_i : \lambda_i \in \mathcal{Q}_\alpha$, *changes* $\leftarrow$ true
while *changes* **do**
  changes $\leftarrow$ false
  $W \leftarrow \{(G, \lambda^c, \text{cl}) \mid a[G, \lambda^c]$ is defined
$\wedge \text{cl} \in P \ \wedge \exists\sigma$ s.t. $G = \sigma(\text{cl.head})\}$
  for each $(G, \lambda^c, \text{cl}) \in W$ **do**
    $\lambda^t \leftarrow \texttt{abs\_call}(G, \lambda^c, \text{cl.head})$
    $\lambda^t \leftarrow \texttt{solve\_body}(\text{cl.body}, \lambda^t)$
    $\lambda^{s_0} \leftarrow \texttt{abs\_proceed}(G, \text{cl.head}, \lambda^t)$
    $\lambda^{s'} \leftarrow \texttt{abs\_generalize}(\lambda^{s_0}, \{a[G, \lambda^c]\})$
    if $\lambda^{s'} \neq \lambda^s$ **then**
      $a[G, \lambda^c] \leftarrow \lambda^{s'}$, *changes* $\leftarrow$ true

function $\texttt{solve\_body}(B, \lambda^t)$
  for each $L \in B$ **do**
    $\lambda^c \leftarrow \texttt{abs\_project}(L, \lambda^t)$
    $Calls = \{\lambda \mid a[H, \lambda']$ is defined
$\wedge\ \exists\sigma$ s.t. $\sigma(H) = L \wedge \lambda = \sigma(\lambda')\}$
    $\lambda^{c'} \leftarrow \texttt{abs\_generalize}(\lambda^c, Calls)$
    $\lambda^s \leftarrow \texttt{solve}(L, \lambda^{c'})$
    $\lambda^t \leftarrow \texttt{abs\_extend}(L, \lambda^s, \lambda^t)$
  return λ^t

Naive fixpoint algorithm with guidance – full

GuidedAnalyze($\mathcal{Q}_\alpha, P$)

output global : A, E

$a[L_i, \lambda_i] \leftarrow \bot$ **for all** $L_i : \lambda_i \in \mathcal{Q}_\alpha$, *changes* $\leftarrow$ true
$E \leftarrow \emptyset$
while *changes* **do**
    changes $\leftarrow$ false
    $W \leftarrow \{(G, \lambda^c, \mathrm{cl}) \mid a[G, \lambda^c]$ is defined $\wedge \mathrm{cl} \in P \ \wedge \exists \sigma$ s.t. $G = \sigma(\mathrm{cl.head})\}$
    for each $(G, \lambda^c, \mathrm{cl}) \in W$ **do**
        $\lambda^t \leftarrow$ `abs_call`$(G, \lambda^c, \mathrm{cl.head})$
        $\lambda^t \leftarrow$ `solve_body`$(\mathrm{cl.body}, \lambda^t)$
        $\lambda^{s_0} \leftarrow$ `abs_proceed`$(G, \mathrm{cl.head}, \lambda^t)$
        $\lambda^{s'} \leftarrow$ `apply_succ` $(G, \lambda^c, \lambda^{s_0}, a[G, \lambda^c])$
        if $\lambda^{s'} \neq \lambda^s$ **then**
            $a[G, \lambda^c] \leftarrow \lambda^{s'}$, *changes* $\leftarrow$ true   ▷ Fixpoint not reached

function `solve_body`(B, λ^t)
    for each $L \in B$ **do**
        $\lambda^c \leftarrow$ `abs_project`(L, λ^t)
        $\lambda^{c'} \leftarrow$ `apply_call`(L, λ^c)
        $\lambda^s \leftarrow$ `solve`$(L, \lambda^{c'})$
        $\lambda^t \leftarrow$ `abs_extend`$(L, \lambda^s, \lambda^t)$
    return λ^t

Example - Success to refine result

Taken from Ciao libraries for bit-coded-set operations:

```
% analyzing with eterms (types) domain (no sharing)
bitcode_to_set(0,[])  :- ! .
bitcode_to_set(C,S) :-
    bitcode_to_set(C,0,S) .

:- trust pred bitcode_to_set(A, B, C) => list(C).
bitcode_to_set(0,_,[]) :- ! .
bitcode_to_set(Code,Num,LNum):-
    ( (Code /\ 1)  =\=  0 ->
        LNum = [Num|Tail]
    ;   LNum = Tail), % eterms domain loses precision
    NNum is Num + 1,
    NCode is Code >> 1,
    bitcode_to_set(NCode,NNum,Tail) .
```

This code uses an accumulator for efficiency. Trust can be obtained from the **same** domain using the same program without accumulator.

Example - Success to describe libraries

```
:- trust pred get_code(X) => int(X).
:- trust pred put_code(X) => int(X).

main(X) :-
    print('Starting to Filter'),nl,
    filter.

filter :-
    get_code(X),
    filt(X),
    filter.

filt(X) :-
   0 is X mod 2, !,
   put_code(X).
filt(_).
```

Example - Calls

```
append([], L, L).
append([X|Xs], L, [X|As]) :-
    append(Xs, L, As).
```

Analysis result: $\langle \text{append}(L1, L2, L3), (L1/\top, L2/\top, L3/\top), (L1/\texttt{list}, L2/\top, L3/\top)\rangle$

```
:- trust pred append(L1,L2,L3) : list(L2).
append([], L, L).
append([X|Xs], L, [X|As]) :-
    append(Xs, L, As).
```

Analysis result: $\langle \text{append}(L1, L2, L3), (L1/\top, L2/\texttt{list}, L3/\top), (L1/\texttt{list}, L2/\texttt{list}, L3/\texttt{list})\rangle$