Relational Summaries for Interprocedural Analysis

Remy Boutonnet (UGA)

VERIMAG - PACSS

June 14th, 2018
Program analysis

Program analysis is concerned with finding invariant properties of programs.

Many interesting properties in program analysis are undecidable.
Abstract Interpretation gives safe approximate answers to undecidable questions.

The program is executed by propagating abstract properties.

Extrapolation (widening) is used to deal with loops.

Example:

Set of the possible values of a numerical variable \( \{3, 5, 7, 51\} \) → Interval of integers \([3; 51]\)
Linear Relation Analysis

```plaintext
assume n >= 0;
i := 0;
-- 1: i = 0 and n >= 0

-- 2: i >= 0 and i <= n
while i < n
-- 3: i >= 0 and i <= n-1
   i := i + 1;
-- 4: i >= 1 and i <= n
end;
-- 5: i = n and n >= 0
```

→ Abstract properties are systems of linear equalities and inequalities. Powerful relational analysis but expensive.
Interprocedural analysis: state of the art

**Inlining**: expand the procedure body at a call statement.

**Top-down analysis**: from the caller to the callees, procedures are analyzed in each call context.  

**Bottom-up analysis**: procedure summaries, from the callees to the callers.  

**Hybrid analysis**: top-down with reuse of similar contexts.  
Concrete relational procedure summaries

```plaintext
procedure div (a, b, q, r)
begin
    assert(a >= 0 && b >= 1);
    0:
        q := 0;
        r := a;
    1:
    2: while r >= b
    3:
        r := r - b;
        q := q + 1;
    4:
    end;
end
```

\[ Q_0 = \{(a_0, b_0, q_0, r_0, a, b, q, r) | a_0 \geq 0 \land b_0 \geq 1 \land a = a_0 \land b = b_0 \land q = q_0 \land r = r_0\} \]

\[ Q_1 = Q_0[q := 0][r := a] \]

\[ Q_2 = Q_1 \cup Q_4 \]

\[ Q_5 = Q_2 \cap (r \leq b - 1) \]

\[ Q_3 = Q_2 \cap (r \geq b) \]

\[ Q_4 = Q_3[r := r - b][q := q + 1] \]

The least solution for \( Q_5 \) is the concrete summary of procedure \( \text{div} \):

\[ Q_5 = (a = a_0 \land b = b_0 \land a = bq + r \land q \geq 0 \land b - 1 \geq r \land r \geq 0) \]
procedure div (a, b, q, r)
begin
    assert(a >= 0 && b >= 1);
    0:
        q := 0;
        r := a;
    1:
    2: while r >= b
    3:
        r := r - b;
        q := q + 1;
    4:
    end;
    5:
end

A   =  (a_0 \geq 0 \land b_0 \geq 1)
Q_0 =  (a_0 \geq 0 \land b_0 \geq 1 \land a = a_0 \\
      \land b = b_0 \land q = q_0 \land r = r_0)
Q_1 =  Q_0[q := 0][r := a]
Q_2 =  Q_2 \nabla (Q_1 \sqcup Q_4)
Q_3 =  Q_2 \cap (r \geq b)
Q_4 =  Q_3[r := r - b][q := q + 1]

Summary using standard LRA:

Q_5 =  (r \geq 0 \land q \geq 0 \land b \geq r + 1 \\
      \land a = a_0 \land b = b_0)
procedure abs (x, r)
begin
  if x < 0 then
    r := x * -1;
    -- r = -x and x <= -1
  else
    r := x;
    -- r = x and x >= 0
  end;
  -- r >= x and r >= -x
end

A convex polyhedron → too coarse!

Proposition: A set of polyhedra.
Disjunctive relational summaries

A modular interprocedural analysis to improve the scalability of Linear Relation Analysis.

Applied to LRA, but based on a general framework called disjunctive relational abstract interpretation.

**Principle:** computing disjunctions of abstract input-output relations.
Disjunctive relational summaries

Let $p$ be a procedure with a precondition $\mathcal{A}$.
Let $X_0$ and $X$ be sets of variables representing initial and final values of parameters.

The summary $\sigma_p = \{P_1(X_0, X), \ldots, P_n(X_0, X)\}$ of $p$ is a finite set of polyhedra such that:

$$\{A_k = P_k(X_0, X) \downarrow X_0\}$$

is an abstract partition of $\mathcal{A}$.

$$\forall k_1, k_2 \in \{1, \ldots, n\}, k_1 \neq k_2 \Rightarrow \gamma(A_{k_1}) \cap \gamma(A_{k_2}) = \emptyset$$

$$\bigcup_{k=1}^{n} \gamma(A_k) = \mathcal{A}$$
procedure div (a, b, q, r)
begin
    assert(a >= 0 && b >= 1);
    0:
        q := 0;
        r := a;
    1:
    2: while r >= b
    3:
        r := r - b;
        q := q + 1;
    4:
    end;
end

A = (a_0 \geq 0 \land b_0 \geq 1)
Q_0 = (a_0 \geq 0 \land b_0 \geq 1 \land a = a_0 \\
\land b = b_0 \land q = q_0 \land r = r_0)
Q_1 = Q_0[q := 0][r := a]
Q_2 = Q_2 \land (Q_1 \cup Q_4)
Q_3 = Q_2 \land (r \geq b)
Q_4 = Q_3[r := r - b][q := q + 1]

→ We lost the precondition a_0 \geq 0

Q_5 = (r \geq 0 \land q \geq 0 \land b \geq r + 1 \\
\land a = a_0 \land b = b_0)
The precondition $A$ is an obvious invariant: a procedure do not change its initial state.

Because of the use of **widening**, the result may not satisfy the precondition.

It is sound and interesting to use a limited widening $\nabla_A$ such that:

$$Q_i \nabla_A Q_j = (Q_i \nabla Q_j) \cap A$$
procedure div (a, b, q, r)
begin
    assert(a >= 0 && b >= 1);
    0:
        q := 0;
        r := a;
    1:
    2: while r >= b
    3:
        r := r - b;
        q := q + 1;
    4:
        end;
    5:
end

A = (a_0 \geq 0 \land b_0 \geq 1)
Q_0 = (a_0 \geq 0 \land b_0 \geq 1 \land a = a_0 \\
    \land b = b_0 \land q = q_0 \land r = r_0)
Q_1 = Q_0[q := 0][r := a]
Q_2 = (Q_2 \nwedge (Q_1 \lhd Q_4)) \sqcap A
Q_5 = Q_2 \sqcap (r \leq b - 1)
Q_3 = Q_2 \sqcap (r \geq b)
Q_4 = Q_3[r := r - b][q := q + 1]

The summary recovers more than just the precondition.

Q_5 = (r \geq 0 \land q \geq 0 \land b \geq r + 1 \\
    \land a \geq q + r \land a = a_0 \land b = b_0)
Refinement according to local reachability

Let \( \{Q_1, ..., Q_k\} \) be the analysis result using LRA with precondition \( A \).

\[ A_i = Q_i \downarrow X_0 \] is a necessary precondition for program point \( i \) to be \textit{reachable}. Let \( c \) be a linear inequality such that:

\[
A_i \trianglerighteq c
\]

\[
A \cap c \neq \bot \land A \cap \overline{c} \neq \bot
\]

\( A \cap \overline{c} \) is a sufficient precondition for program point \( i \) to be \textit{unreachable}. \((A \cap c, A \cap \overline{c})\) are good candidates to refine precondition \( A \).
procedure div (a, b, q, r)
begin
    assert(a >= 0 && b >= 1);
0:
    q := 0;
    r := a;
1:
2:  while r >= b
3:     r := r - b;
    q := q + 1;
4:  end;
5:  end

Analysis results for the branches of the loop condition:

Q₃ = (r ≥ b ∧ q ≥ 0 ∧ b ≥ 1 ∧ a ≥ q + r ∧ a = a₀ ∧ b = b₀)
Q₅ = (r ≥ 0 ∧ q ≥ 0 ∧ b ≥ r + 1 ∧ a ≥ q + r ∧ a = a₀ ∧ b = b₀)
Refinement according to local reachability

procedure div (a, b, q, r)
begin
  assert(a >= 0 && b >= 1);
  0: q := 0;
      r := a;
  1:
  2: while r >= b
  3:     r := r - b;
      q := q + 1;
  4:
end;
end

A = (a_0 \geq 0 \land b_0 \geq 1)

Projections on initial values:

A_3 = Q_3 \downarrow X_0 = (a_0 \geq b_0 \land b_0 \geq 1)
A_5 = Q_5 \downarrow X_0 = (a_0 \geq 0 \land b_0 \geq 1)

\[ c = (a_0 \geq b_0) \land A_3 \sqsubseteq c \]

A_3 \neq A we can refine A into:

A'_3 = A \cap \overline{c} = (a_0 < b_0 \land a_0 \geq 0)
A''_3 = A \cap c = (a_0 \geq b_0 \land b_0 \geq 1)

A_5 = A
Refinement according to local reachability

```plaintext
procedure div (a, b, q, r)
begin
    assert(a >= 0 && b >= 1);

0:
    q := 0;
    r := a;

1:
    while r >= b

2:
    r := r - b;
    q := q + 1;

end
end
```

\[ A'_3 = (a_0 < b_0 \land a_0 \geq 0) \]

Analysis result for \( Q_3, Q_5 \) under \( A'_3 \)

\[ Q_3 = \bot \]
\[ Q_5 = (q = 0 \land r = a \land a < b \land a = a_0 \land b = b_0) \]
Refinement according to local reachability

```plaintext
procedure div (a, b, q, r)
begin
    assert(a >= 0 && b >= 1);
    0:
        q := 0;
        r := a;
    1:
        2: while r >= b
            3:
                r := r - b;
                q := q + 1;
    4:
        end;
    5:
end
```

\[ A_3'' = (a_0 \geq b_0 \land b_0 \geq 1) \]

Analysis result for \( Q_5 \) under \( A_3'' \)

\[ Q_5 = (q + r \geq 1 \land b \geq r + 1 \land r \geq 0 \land a + 1 \geq b + q \land q \geq 0 \land a \geq b \land a = a_0 \land b = b_0) \]
Refinement according to local reachability

procedure div (a, b, q, r)
begin
    assert(a >= 0 && b >= 1);
    0: q := 0;
    r := a;
    1:
    2: while r >= b
    3:     r := r - b;
          q := q + 1;
    4: end;
end

\[ A_3'' = (a_0 \geq b_0 \land b_0 \geq 1) \]

The loop is executed at least once but we don’t have \( q \geq 1 \).

\[ Q_5 = (q + r \geq 1 \land b \geq r + 1 \]
\[ \land a + 1 \geq b + q \]
\[ \land r \geq 0 \land q \geq 0 \]
\[ \land a \geq b \land a = a_0 \land b = b_0) \]
procedure div (a, b, q, r) begin
  assert(a >= 0 && b >= 1);
  0: q := 0;
  r := a;
  1: while r >= b
  2:   r := r - b;
  3:     q := q + 1;
  4: end;
  5: end

A'' = (a_0 \geq b_0 \land b_0 \geq 1)
Q_0 = (a_0 \geq b_0 \land b_0 \geq 1 \land a = a_0 \\
\land b = b_0 \land q = q_0 \land r = r_0)
Q_1 = Q_0[q := 0][r := a]
Q_2 = (Q_2 \nabla (Q_1 \sqcup Q_4)) \sqcap A''
Q_5 = Q_2 \sqcap (r < b)
Q_5 = (Q_1 \sqcap (r < b)) \sqcup (Q_4 \sqcap (r < b))
Q_3 = Q_2 \sqcap (r \geq b)
Q_4 = Q_3[r := r - b][q := q + 1]

We have now for Q_5 under A'':

Q_5 = (r \geq 0 \land q \geq 1 \land q + r \geq 1 \\
\land b \geq r + 1 \land a + 1 \geq b + q + r \\
\land a_0 \geq b_0 \land b_0 \geq 1 \land a = a_0 \land b = b_0)
Refinement according to local reachability

procedure \texttt{div} (a, b, q, r) begin
  assert(a >= 0 && b >= 1);
  0: q := 0; r := a;
  1: while r >= b
  2: r := r - b;
  3: q := q + 1;
  4: end;
  5: end

\[ A'_3 = (a_0 < b_0 \land a_0 \geq 0) \]
\[ A''_3 = (a_0 \geq b_0 \land b_0 \geq 1) \]
\[ Q_5(A'_3) = (q = 0 \land r = a \land a < b \land a = a_0 \land b = b_0) \]
\[ Q_5(A''_3) = (r \geq 0 \land q \geq 1 \land q + r \geq 1 \land b \geq r + 1 \land a + 1 \geq b + q + r \land a_0 \geq b_0 \land b_0 \geq 1 \land a = a_0 \land b = b_0) \]

\[ Q_5(A'_3) \downarrow X_0 = (a_0 < b_0 \land a_0 \geq 0) = A'_3 \]
\[ Q_5(A''_3) \downarrow X_0 = (a_0 \geq b_0 \land b_0 \geq 1) = A''_3 \]

\( \rightarrow \) No more refinement is possible for \( A'_3 \) and \( A''_3 \).
The summary of div is $\sigma_{\text{div}} = \{R_1, R_2\}$ such that:

\[
R_1 = (a_0 \geq b_0 \land b_0 \geq 1 \land r \geq 0 \land q \geq 1 \land q + r \geq 1 \\
\land b \geq r + 1 \land a + 1 \geq b + q + r \land a = a_0 \land b = b_0)
\]

\[
R_2 = (a_0 < b_0 \land a_0 \geq 0 \land q = 0 \land r = a \land a = a_0 \land b = b_0)
\]
A disjunctive summary can be used to obtain the effect of a call.

Let $p$ be a procedure and $R = \{R_1, ..., R_n\}$ be the disjunctive polyhedral summary of $p$ and $P(Y_0, Y)$ be the call context. Let $\pi$ be the parameter renaming corresponding to the call to $p$.

Let $Y_0$ denote the initial values of actual parameters. Let $Y$ denote the final values of actual parameters.

The effect of the call is:

$$\exists Y', P[Y'/Y] \sqcap \pi(R_k)[Y'/Y_0]$$
Summary of McCarthy’s 91 function

```
procedure f91 (x, y)
begin
    z, t : int;
    if x > 100 then
        y := x-10;
        1:
    else
        z := x+11;
        f91(z,t);
        f91(t,y);
        2:
    end;
end
```

\( R = P_1 \sqcup P_2 \)

\( P_1 = (x \geq 101 \land y = x - 10) \)

\( P_2 = (x \leq 100 \land \exists t, (R(x+11,t) \cap R(t,y))) \)

We find with a simple LRA that:

\( P_1 = (x \geq 101 \land y = x - 10) \)

\( P_2 = (x \leq 100 \land y + 9 \geq x \land y \geq 91) \)

\( R = (x \leq y + 10 \land y \geq 91) \)
Summary of McCarthy’s 91 function

**procedure f91 (x, y)**

begin

\[ z, t : \text{int}; \]

if \( x > 100 \) then

\[ y := x - 10; \]

1:

else

\[ z := x + 11; \]

\[ f91(z,t); \]

\[ f91(t,y); \]

2:

end

end

\[ P_1 = (x \geq 101 \land y = x - 10) \]

\[ P_2 = (x \leq 100 \land y + 9 \geq x \land y \geq 91) \]

\[ R = (x \leq y + 10 \land y \geq 91) \]

We have \( P_1 \downarrow \{x\} = (x \geq 101) \).

From precondition \( l_0 = \top \) we obtain new preconditions:

\[ l_1 = (x \geq 101) \]

\[ l_2 = (x \leq 100) \]
Summary of McCarthy’s 91 function

procedure f91 (x, y)
begin
    z, t : int;
    if x > 100 then
        y := x-10;
        1:
    else
        z := x+11;
        f91(z,t);
        f91(t,y);
        2:
    end;
end

\[
\begin{align*}
I_1 &= (x \geq 101) \\
P_1(I_1) &= (x \geq 101 \land y = x - 10) \\
P_2(I_1) &= \bot \\
I_2 &= (x \leq 100) \\
P_1(I_2) &= \bot \\
P_2(I_2) &= (x \leq 100 \land y \geq 91)
\end{align*}
\]

\[
\begin{align*}
R(I_1) &= (x \geq 101 \land y = x - 10) \\
R(I_2) &= (x \leq 100 \land y \geq 91)
\end{align*}
\]

The summary is not much better.
procedure f91 (x, y)
begin
    z, t : int;
    if x > 100 then
        y := x-10;
    1:
    else
        z := x+11;
        f91(z,t);
        f91(t,y);
    2:
end;

\[ I_2 = (x \leq 100) \]
\[ \mathcal{P}_2(I_2) = (x \leq 100 \land y \geq 91) \]

We can refine precondition \( I_2 \) according to the
condition \( x + 11 \geq 101 \) at the first recursive call
\( f91(z, t) \).

\[ (x + 11 \geq 101) \cap (x \leq 100) = (90 \leq x \leq 100) \]

We obtain the new preconditions \( I'_2 \) and \( I''_2 \):

\[ I'_2 = (90 \leq x \leq 100) \]
\[ I''_2 = (x \leq 89) \]
procedure f91 (x, y)
begin
    z, t : int;
    if x > 100 then
        y := x-10;
    else
        z := x+11;
    f91(z,t);
    f91(t,y);
end;
end

The summary of McCarthy’s 91 function is such that:

\begin{align*}
R_1(l'_2) & = (x \leq 89 \land y = 91) \\
R_2(l''_2) & = (90 \leq x \leq 100 \land y = 91) \\
R_3(l_1) & = (x \geq 101 \land y = x - 10)
\end{align*}
We compute summaries of C functions with scalar type parameters (ex: `int`) or pointers-to-scalar types (ex: `int *`).

We focus on the numerical aspects with a very basic treatment of pointers.

We could use an alias analysis to refine summaries according to possible aliasing cases.
Implementation: **Mars**

This approach has been implemented in a new static analyzer called **Mars** (**Mars** Abstract interpretation Research System).

**Mars** is organized as a collection of tools working on the **Mars** intermediate representation.

**Mars** uses a custom frontend based on Clang which translates C programs into the **Mars** IR.
Implementation: MARS

→ Intermediate representation designed for program analysis
→ Modular architecture to ease the development of new static analyses
→ High quality source traceability informations
Summaries of synchronous modules

Synchronous modules like lustre nodes are implemented by step procedures.

Each step procedure can have an **internal memory** which holds the **state** of a module.

These internal memories are accessed and **remanent** from one invocation of the step procedure to another.

→ This entails these procedures to a special treatment.
Methods in object-oriented languages are a special case of procedures with **remanent memory**.

Methods can access and change the values of class members. The values of class members remain from one method invocation to another.