Disjunctive relational abstract interpretation for interprocedural program analysis

Nicolas Halbwachs, joint work with Rémy Boutonnet

Verimag/CNRS, and Grenoble-Alpes University
Grenoble, France
Standard program analysis

Goal: compute (an upper approximation of) the reachable states of a program \((S, S_0, \rightarrow)\)

\[
A = S_0 \cup \{s' \mid \exists s \in A, s \rightarrow s'\}
\]

Trace partitioning:
Assume \(S\) is partitioned: \(S = S_1 \oplus S_2 \oplus \ldots \oplus S_k\), and let \(A_i = A \cap S_i, i = 1..k:\)

\[
A_i = (S_0 \cap S_i) \cup \left( \bigcup_{j=1..k} \{s' \in S_i \mid \exists s \in A_j, s \rightarrow s'\} \right)
\]

\(\rightarrow\) a system of recursive semantic equations
Numerical program analysis

A numerical state: \( n \) variables \( \Rightarrow \) a vector \( X \in \mathcal{N}^n \) (\( \mathcal{N} = \mathbb{Z} \) or \( \mathbb{Q} \))
Numerical program analysis

A numerical state: $n$ variables $\Rightarrow$ a vector $X \in \mathcal{N}^n$ ($\mathcal{N} = \mathbb{Z}$ or $\mathbb{Q}$)

Example of semantic equations

$$A_0 = \mathcal{N}^2$$
$$A_1 = A_0[x \leftarrow 0][y \leftarrow 0]$$
$$A_2 = A_1 \cup A_6$$
$$A_3 = A_2 \cap (x \leq 100)$$
$$A_4 = A_3[x \leftarrow x+2]$$
$$A_5 = A_3[x \leftarrow x+1][y \leftarrow y+1]$$
$$A_6 = A_4 \cup A_5$$
$$A_7 = A_2 \cap (x \geq 101)$$
Numerical abstract domains
Numerical abstract domains

Polyhedra

\[
\begin{align*}
  x + 2y &\geq 8 \\
  x &\leq y + 8 \\
  x &\leq 13 \\
  y &\leq 2x + 4 \\
  x + 2y &\leq 28
\end{align*}
\]
Numerical abstract domains

Polyhedra

$x + 2y \geq 8$

$x \leq y + 8$

$x \leq 13$

$y \leq 2x + 4$

$x + 2y \leq 28$
Numerical abstract domains

Intervals

\[ x \in [0, 13] \]
\[ y \in [0, 12] \]
Numerical abstract domains

Intervals
not relational

\[ x \in [0, 13] \]
\[ y \in [0, 12] \]
Numerical abstract domains

Octagons

“weakly relational”

\[ 0 \leq x \leq 13 \]
\[ 0 \leq y \leq 12 \]
\[ 4 \leq x + y \leq 20.5 \]
\[ -8 \leq x - y \leq 8 \]
An example with polyhedra

\[ x := 0 ; y := 0; \]

\[ \text{while} \]
\[ \begin{align*}
  &x \leq 100 \text{ do} \\
  &\quad \text{if} \ ?? \ \text{then} \ x := x+2 \\
  &\quad \text{else} \ x := x+1; \ y := y+1; \\
  &\quad \text{endif}
\end{align*} \]

\[ \text{endwhile} \]
An example with polyhedra

\[
x := 0 \ ; \ y := 0;
\]
\[
x=y=0
\]

while
\[
x \leq 100 \ do
\]
\[
\text{if ?? then } x := x+2
\]
\[
\text{else } x := x+1; \ y := y+1;
\]
\[
\text{endif}
\]
\[
\text{ endwhile}
\]
An example with polyhedra

\[ x := 0 \ ; \ y := 0; \]
\[ x = y = 0 \]
while \( x = y = 0 \)
\[ x \leq 100 \] do
\[ \text{if } ?? \text{ then } x := x + 2 \]
\[ \text{else } x := x + 1; \ y := y + 1; \]
\[ \text{endif} \]
\[ \text{endwhile} \]
\[ \emptyset \]
An example with polyhedra

\[
x := 0 \ ; \ y := 0;
\]
\[
x = y = 0
\]
\[
\text{while } x = y = 0
\]
\[
x \leq 100 \text{ do}
\]
\[
\text{if } ?? \text{ then } x := x + 2
\]
\[
x = 2, \ y = 0
\]
\[
\text{else } x := x + 1; \ y := y + 1;
\]
\[
x = 1, \ y = 1
\]
\[
\text{endif}
\]
\[
\text{endwhile}
\]
\[
\emptyset
\]
An example with polyhedra

\[ x := 0 ; y := 0 ; \]
\[ x=y=0 \]
while \( x=y=0 \)
\[ x \leq 100 \text{ do} \]
\[ \text{if } ?? \text{ then } x:= x+2 \]
\[ x=2, y=0 \]
\[ \text{else } x:= x+1 ; y:=y+1 ; \]
\[ x=1, y=1 \]
endif
\[ 1 \leq x \leq 2, x+y=2 \]
endwhile
\[ \emptyset \]
An example with polyhedra

\[ \begin{align*}
&x := 0 \; ; \; y := 0; \\
&x = y = 0 \\
&\text{while} \\
&\quad x \leq 100 \text{ do} \\
&\quad \text{if ?? then } x := x+2 \\
&\quad \text{else } x := x+1; \; y := y+1; \\
&\quad \text{endif} \\
&1 \leq x \leq 2, \; x+y=2 \\
&\text{endwhile}
\end{align*} \]
An example with polyhedra

\[
x := 0 ; y := 0;
\]
\[
x=y=0
\]
while \(0 \leq y \leq x, x+y \leq 1\)
\[
x \leq 100 \text{ do}
\]
if ?? then \(x := x+2\)
\[
\text{else } x := x+1; y := y+1;
\]
endif
\[
1 \leq x \leq 2, x+y = 2
\]
endwhile
An example with polyhedra

\[ x := 0 ; y := 0; \]
\[ x = y = 0 \]

while \( 0 \leq y \leq x, x + y \leq 1 \) \( \quad 0 \leq y \leq x \)
\[ x \leq 100 \] do
\[ \text{if ?? then } x := x + 2 \]
\[ \text{else } x := x + 1; y := y + 1; \]
\[ \text{endif} \]
\[ 1 \leq x \leq 2, x + y = 2 \]
endwhile

Widening
An example with polyhedra

\[ \begin{align*}
x &:= 0 ; y := 0; \\
x &= y = 0
\end{align*} \]

while

\[ \begin{align*}
x &\leq 100 \text{ do } 0 \leq y \leq x \leq 100 \\
\text{if ?? then } x &:= x + 2 \\
\text{else } x &:= x + 1; y := y + 1;
\end{align*} \]

endif

endwhile
An example with polyhedra

\[
x := 0 ; y := 0;
\]
\[
x = y = 0
\]

while
\[
x \leq 100 \text{ do } 0 \leq y \leq x \leq 100
\]

if ?? then \( x := x + 2 \)
\[
0 \leq y \leq x - 2 \leq 100
\]

else \( x := x + 1 ; y := y + 1 ; \)
\[
1 \leq y \leq x \leq 101
\]
endif

endwhile
An example with polyhedra

\[
x := 0 ; y := 0;
x = y = 0
\]

while

\[
x \leq 100
\]
do

\[
0 \leq y \leq x \leq 100
\]

if ?? then

\[
x := x + 2
0 \leq y \leq x - 2 \leq 100
\]

else

\[
x := x + 1; y := y + 1;
1 \leq y \leq x \leq 101
\]
endif

\[
0 \leq y \leq x \leq 102, 2 \leq x + y \leq 202
\]
endwhile
An example with polyhedra

\[
x := 0 \ ; \ y := 0;
\]

\[
x = y = 0
\]

while \(0 \leq y \leq x \leq 102, \ x + y \leq 202\)

\[
x \leq 100 \ \text{do} \ \ 0 \leq y \leq x \leq 100
\]

if ?? then \(x := x + 2\)

\[
0 \leq y \leq x - 2 \leq 100
\]

else \(x := x + 1; \ y := y + 1;\)

\[
1 \leq y \leq x \leq 101
\]

endif

\[
0 \leq y \leq x \leq 102, \ 2 \leq x + y \leq 202
\]

endwhile

\[
0 \leq y, \ 101 \leq x \leq 102, \ x + y \leq 202
\]
Motivation

- **Relational domains** (polyhedra, octagons, congruences, ...) are too expensive to scale up (on huge monolithic programs)
  - → **interprocedural analysis especially interesting** (analyse medium size procedures once and for all)

- **Bottom-up interprocedural analysis** (build a procedure summary independently of the calling context) little used, because too imprecise with non-relational domains
  - → **take advantage of relational domains to build precise relational procedure summaries**
Interprocedural program analysis

How to deal with procedures and calls?

- **Inlining**

- **Top-down**: at each call, analyse the procedure in the context of its actual parameters.

- **Bottom-up**: analyse the procedure once and for all, synthesize a summary to be used at each call.

- **Hybrid strategies**: mix the two, e.g., compute a summary in the simplest aliasing context (no aliases), compute a new summary when a call with possible aliases is encountered.
From state analysis to relational analysis

**State analysis:** Starting from a set $S_0$ of initial states, compute at each control point $i$, the set

$$\mathcal{A}_i = \{ s \in S_i \mid \exists s_0 \in S_0, s.t., s_0 \rightarrow^* s \}$$

**Relational analysis:** Starting from a set $S_0$ of initial states, compute at each control point $i$, the *relation*

$$\mathcal{R}_i = \{ (s_0, s) \in S_0 \times S_i \mid s_0 \in S_0, s_0 \rightarrow^* s \}$$
From state analysis to relational analysis

**State analysis:** Starting from a set $S_0$ of initial states, compute at each control point $i$, the set

$$A_i = \{ s \in S_i \mid \exists s_0 \in S_0, \text{s.t.,} s_0 \rightarrow^* s \}$$

**Relational analysis:** Starting from a set $S_0$ of initial states, compute at each control point $i$, the *relation*

$$R_i = \{ (s_0, s) \in S_0 \times S_i \mid s_0 \in S_0, s_0 \rightarrow^* s \}$$

**Important fact:** $\exists s. R_i$ is a necessary condition on $s_0$ for point $i$ to be reachable from $s_0$
Relational analysis of a procedure

Procedures:

- All parameters are supposed to be passed by reference (but we ignore pointer manipulation, and we entrust existing analyses to detect aliasing problems).
- Global variables considered as additional parameters.
Relational analysis of a procedure

Procedures:
- All parameters are supposed to be passed by reference (but we ignore pointer manipulation, and we entrust existing analyses to detect aliasing problems).
- Global variables considered as additional parameters.

Relational analysis:
- Duplicate each parameter $x$ with an auxiliary variable $x^0$, to record its initial value.
- Perform a standard analysis of the body, starting from $x = x^0$ for each parameter $x$.
- Project away local variables from the result at return point to obtain a relational summary $\mathcal{R}(X^0, X)$. 
Example: div

```
proc div (a, b, q, r: int) {
    assume a ≥ 0 && b ≥ 1;
    a0=a ; b0=b ; q0=q ; r0=r;
    q=0 ; r=a;
    while(r ≥ b) {
        r = r-b ; q = q+1 ;
    }
}
```
Example: \texttt{div}

\begin{verbatim}
proc div (a, b, q, r: int) {
    assume a \geq 0 && b \geq 1;
    a0=a ; b0=b ; q0=q ; r0=r;
    q=0 ; r=a;
    while(r \geq b) {
        r = r-b ; q = q+1 ;
    }
}
\end{verbatim}

**Standard Analysis**

\[ r \geq 0 \land q \geq 0 \land b \geq r+1 \land a = a0 \land b = b0 \]
Example: \textit{div}

\begin{verbatim}
proc div (a, b, q, r: int)
    assume a \geq 0 && b \geq 1;
    a0=a ; b0=b ; q0=q ; r0=r;
    q=0 ; r=a;
    while(r \geq b){
        r = r-b ; q = q+1 ;
    }
}
\end{verbatim}

Standard Analysis

\[ r \geq 0 \land q \geq 0 \land b \geq r + 1 \]
\[ \land a = a0 \land b = b0 \]

\frown pre-condition \( a0 \geq 0 \) lost!
Widening limited by precondition

A precondition is obviously an invariant (a procedure may not modify the initial values of its parameters!)
⇒ one can limit (i.e., intersect) the widening by the precondition.

Example (cont.): Intersect the result of the widening with

\[ a_0 \geq 0 \land b_0 \geq 1 \]
Example: `div`

```plaintext
proc div (a, b, q, r: int){
assume a ≥0 && b ≥ 1;
a0=a ; b0=b ; q0=q ; r0=r;
q=0 ; r=a;
while(r ≥ b){
    r = r-b ; q = q+1 ;
}
}
```

Standard Analysis

\[ r \geq 0 \land q \geq 0 \land b \geq r + 1 \land a = a0 \land b = b0 \]

😃 preconditions lost!

With limited widening

\[ r \geq 0 \land q \geq 0 \land b \geq r + 1 \land a \geq q + r \land a = a0 \land b = b0 \]

😢 Better than expected, but still imprecise, as a summary
Disjunctive summaries

- Convex summaries are not expressive enough
- General disjunctions of polyhedra are difficult to manage (ex.: inclusion)
- Idea: disjunctive refinement by partitioning preconditions

precondition $P$ partitioned into $P_1 \oplus P_2 \oplus \ldots \oplus P_m$

- Disjunctive summary

\[
R = R_1 \oplus R_2 \oplus \ldots R_m \quad \text{with} \quad P_i = \exists X. R_i, \ i = 1..m
\]
Example of disjunctive summary

```plaintext
proc div (a, b, q, r: int){
assume a ≥0 && b ≥ 1;
a0=a ; b0=b ; q0=q ; r0=r;
q=0 ; r=a;
while(r ≥ b){
    r = r-b ; q = q+1 ;
}
}
```

Standard analysis:

\[ r \geq 0 \land q \geq 0 \land b \geq r + 1 \]

With limited widening:

\[ r \geq 0 \land q \geq 0 \land b \geq r + 1 \land a \geq q + r \]
Example of disjunctive summary

```
proc div (a, b, q, r: int){
  assume a ≥0 && b ≥1;
  a0=a ; b0=b ; q0=q ; r0=r;
  q=0 ; r=a;
  while (r ≥ b){
    r = r-b ; q = q+1 ;
  }
}
```

Partitioning according to

\[ a_{0} \geq b_{0} , b_{0} \geq a_{0} + 1 \]

(entering the loop or not)

Standard analysis:

\[ r \geq 0 \land q \geq 0 \land b \geq r + 1 \]

With limited widening:

\[ r \geq 0 \land q \geq 0 \land b \geq r + 1 \land a \geq q + r \]
Example of disjunctive summary

```plaintext
proc div (a, b, q, r: int){
assume a ≥ 0 && b ≥ 1;
a0=a ; b0=b ; q0=q ; r0=r;
q=0 ; r=a;
while(r ≥ b){
r = r-b ; q = q+1 ;
}
}
```

Standard analysis:

\[ r \geq 0 \land q \geq 0 \land b \geq r + 1 \]

With limited widening:

\[ r \geq 0 \land q \geq 0 \land b \geq r + 1 \land a \geq q + r \]

Partitioning according to

\[ a0 \geq b0 , b0 \geq a0 + 1 \]

(entering the loop or not)

Result:

either \[ b0 - 1 \geq a0 \geq 0 \]
and \[ q = 0 \land r = a \]
or \[ a0 \geq b0 \geq 1 \]
and \[ r \geq 0 \land q \geq 0 \land q + r \geq 1 \land b \geq r + 1 \land a + 1 \geq b + q \]
Partitioning preconditions

Refinement according to local reachability: Let $R_k(P)$ be the result at control point $k$ of a relational analysis from precondition $P$. It relates the initial values $X^0$ and the current values $X$. Then,

$$R^0_k = \exists X. R_k(P)$$

is a necessary condition on $X^0$ for the point $k$ to be reachable.
Partitioning preconditions

Refinement according to local reachability: Let $\mathcal{R}_k(P)$ be the result at control point $k$ of a relational analysis from precondition $P$. It relates the initial values $X^0$ and the current values $X$. Then,

$$R^0_k = \exists X.\mathcal{R}_k(P)$$

is a necessary condition on $X^0$ for the point $k$ to be reachable.

$R^0_k \subseteq P$, thanks to widening limited by precondition, but if $R^0_k \neq P$, any constraint $c$ of $R^0_k$ not satisfied by $P$, may be used to split the precondition $P$ into

$$P' = P \land c \quad \text{and} \quad P'' = P \land \neg c$$

and point $k$ is not reachable from $P''$. 

R. Boutonnet, N. Halbwachs (Verimag) Disjunctive relational abstract interpretation 16 / 30
Partitioning preconditions: example

**Example:** From the precondition $P = (a_0 \geq 0 \land b_0 \geq 1)$, the analysis of $\text{div}$ provides

$$R = (a = a_0 \land b = b_0 \land r \geq b \geq 1 \land q \geq 0 \land a \geq q + r)$$

at the loop entry. Now,

$$R_0 = \exists (a, b, q, r). R = (a_0 \geq b_0 \geq 1)$$

Since $a_0 \geq b_0$ is not satisfied by the precondition $P$, it provides the (obvious) refinement

$$P' = (a_0 \geq b_0 \geq 1) \quad , \quad P'' = (b_0 - 1 \geq a_0 \geq 0)$$

that we used before.
Postponing loop feedback

The problem: In the div example, our partitioning separates the cases when the loop is entered at least once or not. In the former case, our analysis provides

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

We should find \(q \geq 1\) (the loop is executed at least once)
Postponing loop feedback

The problem: In the div example, our partitioning separates the cases when the loop is entered at least once or not. In the former case, our analysis provides

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

We should find \(q \geq 1\) (the loop is executed at least once)

The explanation:
At loop exit, the semantic equation is

\[R_{exit} = (R_{in} \sqcup R_{feedback}) \sqcap \neg con\]

while \(R_{in} \sqcap \neg con = \emptyset\)
Postponing loop feedback

The problem: In the div example, our partitioning separates the cases when the loop is entered at least once or not. In the former case, our analysis provides

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

We should find \(q \geq 1\) (the loop is executed at least once)

The explanation:
At loop exit, the semantic equation is

\[R_{exit} = (R_{in} \sqcup R_{feedback}) \cap \neg \text{cond}\]

while \(R_{in} \cap \neg \text{cond} = \emptyset\)

The solution: Compute instead

\[R_{exit} = (R_{in} \cap \neg \text{cond}) \sqcup (R_{feedback} \cap \neg \text{cond})\]
Postponing loop feedback

Previous result:

\[(r \geq 0 \land q \geq 0 \land q + r \geq 1 \land b \geq r + 1 \land a + 1 \geq b + q)\]

New result:

\[(r \geq 0 \land q \geq 1 \land b \geq r + 1 \land a + 1 \geq b + q + r)\]

Better than expected
Using procedure summaries

- Let \( p \) be a procedure, with a call \( k : q(A) ; k' : \ldots \)
  where \( A = (a_1, a_2, \ldots, a_n) \) are the actual parameters.

- Let \( R_k \) be the relation computed at point \( k \).

- Let \( F = (f_1, f_2, \ldots, f_n) \) be the formal parameters of \( q \), and
  \( (S_1, \ldots, S_m) \) be a summary of \( q \).
Using procedure summaries

- Let $p$ be a procedure, with a call $k : q(A) ; k' : \ldots$
- where $A = (a_1, a_2, \ldots, a_n)$ are the actual parameters.
- Let $R_k$ be the relation computed at point $k$.
- Let $F = (f_1, f_2, \ldots, f_n)$ be the formal parameters of $q$, and $(S_1, \ldots, S_m)$ be a summary of $q$.

Then, the relation at return point $k'$ is

$$R_{k'} = \bigcup_{\ell=1}^{m} \exists A'. R_k[A/A'] \cap S_\ell[F/A][F^0/A']$$

where

- $R_k[A/A']$ is the result of renaming in $R_k$ each variable $a_i$ as $a'_i$
- $S_\ell[F/A][F^0/A']$ is the result of renaming in $S_\ell$ each variable $f_i$ as $a_i$ and each variable $f^0_i$ as $a'_i$
Refinement according to called procedure summary
Since the relation at return point of a call is a $\bigcup$, its a good idea to make some of its terms empty.

Preconditions of the call: $P_\ell = (\exists F. \mathcal{I}_\ell), \ell = 1..m$

Making $C_\ell = \exists A'. \mathcal{R}_k[A/A'] \cap P_\ell[F^0/A']$ empty: refine the precondition of the caller

$\exists X. C_\ell$ is a necessary condition (on initial values in the caller) for $C_\ell$ to be not empty. Use it to split the precondition, as before.
Example, a recursive procedure

McCarthy’s 91 function

```plaintext
proc f91 (x,y: int) {
    var z, t: int ;
1:  if (x > 100)
2:       y = x -10 ;
3:  else {
4:       z = x + 11 ;
5:       f91 (z, t) ;
6:       f91 (t, y) ;
7:   }
}
```
Example, a recursive procedure

McCarthy’s 91 function

```plaintext
proc f91 (x,y: int) {
    var z, t: int ;
1: if (x > 100) then
2:     y = x -10 ;
else
3:     {z = x + 11 ;
4:         f91 (z, t) ;
5:         f91 (t, y) ;
6:     }
7: }
}
```

Semantic equations
(without parameters duplication)

\[ \mathcal{R} = \mathcal{R}_2 \sqcup \mathcal{R}_7 \]
\[ \mathcal{R}_2 = (x \geq 101 \land y = x - 10) \]
\[ \mathcal{R}_7 = (x \leq 100) \cap (\exists t. \mathcal{R}(x + 11, t) \cap \mathcal{R}(t, y)) \]

22mm]
Example, a recursive procedure

McCarthy’s 91 function

```
proc f91 (x,y: int) {
    var z, t: int ;
    1: if (x > 100) {
        2: y = x -10 ;
    } else {
        4: z = x + 11 ;
        5: f91 (z, t) ;
        6: f91 (t, y) ;
    }
}
```

Semantic equations
(without parameters duplication)

\[
\begin{align*}
R &= R_2 \sqcup R_7 \\
R_2 &= (x \geq 101 \land y = x - 10) \\
R_7 &= (x \leq 100) \sqcap \\
&\quad (\exists t. R(x + 11, t) \sqcap R(t, y))
\end{align*}
\]

22mm]Standard analysis:

\[
\begin{align*}
R_2 &= (x \geq 101 \land y = x - 10) \\
R_7 &= (x \leq 100 \land y + 9 \geq x \land y \geq 91) \\
R &= (x \leq y + 10 \land y \geq 91)
\end{align*}
\]
Example, a recursive procedure (cont.)

Standard analysis:

\[ R_2 = (x \geq 101 \land y = x - 10) \]
\[ R_7 = (x \leq 100 \land y + 9 \geq x \land y \geq 91) \]
\[ R = (x \leq y + 10 \land y \geq 91) \]
Example, a recursive procedure (cont.)

Standard analysis:

\[ R_2 = (x \geq 101 \land y = x - 10) \]
\[ R_7 = (x \leq 100 \land y + 9 \geq x \land y \geq 91) \]
\[ R = (x \leq y + 10 \land y \geq 91) \]

Since \( \exists y. R_2 = (x \geq 101) \), split the initial precondition \( \top \) into \( (x \geq 101) \) and \( (x \leq 100) \).
Example, a recursive procedure (cont.)

Standard analysis:

\[ R_2 = (x \geq 101 \land y = x - 10) \]
\[ R_7 = (x \leq 100 \land y + 9 \geq x \land y \geq 91) \]
\[ R = (x \leq y + 10 \land y \geq 91) \]

Since \( \exists y. R_2 = (x \geq 101) \), split the initial precondition \( \top \) into \( (x \geq 101) \) and \( (x \leq 100) \).

Result:

\[ (x \geq 101 \land y = x - 10) \]
\[ \text{or} \quad (x \leq 100 \land y \geq 91) \]

😊 not much better
Example, a recursive procedure (cont.)

Now, refine according to the summary at the first recursive call, i.e., according to \((x + 11 \geq 101)\).
Example, a recursive procedure (cont.)

Now, refine according to the summary at the first recursive call, i.e., according to \((x + 11 \geq 101)\).

Final result:

\[
(x \geq 101 \land y = x - 10) \\
\text{or } (90 \leq x \leq 100 \land y = 91) \\
\text{or } (x \leq 89 \land y = 91)
\]

😊 most precise summary
Implementation

Method implemented by Rémy Boutonnet in his prototype analyzer MARS (Mars Abstract interpretation Research System)

- Based on CLANG, takes a significant subset of C
- Uses the Apron library of abstract domains
- Applies a standard data-flow analysis to identify pure “value” parameters and pure “result” parameters, which don’t need to be duplicated
- Recursive procedures not yet taken into account
Experiments

**Goal:** Compare the interprocedural analysis to a standard analysis of inlined programs, answer the following questions:

- The construction of summaries involves several analyses, does it induce a significant overhead?
- Does the bottom-up approach result in a significant loss of precision?

**Problems:**

- Find an interesting benchmark (numerical programs with procedures)
- How to compare the precision?
Experiments (cont.)

- Use of the Mälardalen benchmark (worst case execution time), 19 programs sometimes augmented with statement counters
- **Precision**: Compare the results at the end of the main program (after projection on the variables of the main)
  - **Qualitative comparison**: is the result of interprocedural analysis better (⊂), worse (⊃), equal (≡), or incomparable (⊈) w.r.t. the one of inlining analysis?
  - **Quantitative comparison**: compare the number of constraints
First experimental results

- Interprocedural analysis faster on 13 programs over 19 (average speedup 2.9).
  Most procedures called once. The speedup increases rapidly with the number of calls.
- Some precision lost on 3 programs (worst: from 32 to 12 constraints)
- Some precision gained on 5 programs (best: from 1 to 6 constraints)
Future work

Extend to modular analysis of procedures with memories (objects, reactive software)
Future work

Extend to modular analysis of procedures with memories (objects, reactive software)

A reactive module

- $X$ : input variables
- $Y$ : output variables
- $M$ : memory variables

\[
M = M^0; \\
\text{forever } \{ \\
\quad \text{read}(X); \\
\quad (Y,M) = \text{Step}(X,M); \\
\quad \text{write}(Y); \\
\}
\]
Future work

Can we construct a summary of each module (including an invariant on the memory) and use these summaries to analyse the whole system?

R. Boutonnet, N. Halbwachs (Verimag)