Verasco: a Formally Verified C Static Analyzer

Jacques-Henri Jourdan, Vincent Laporte, Sandrine Blazy, Xavier Leroy, David Pichardie

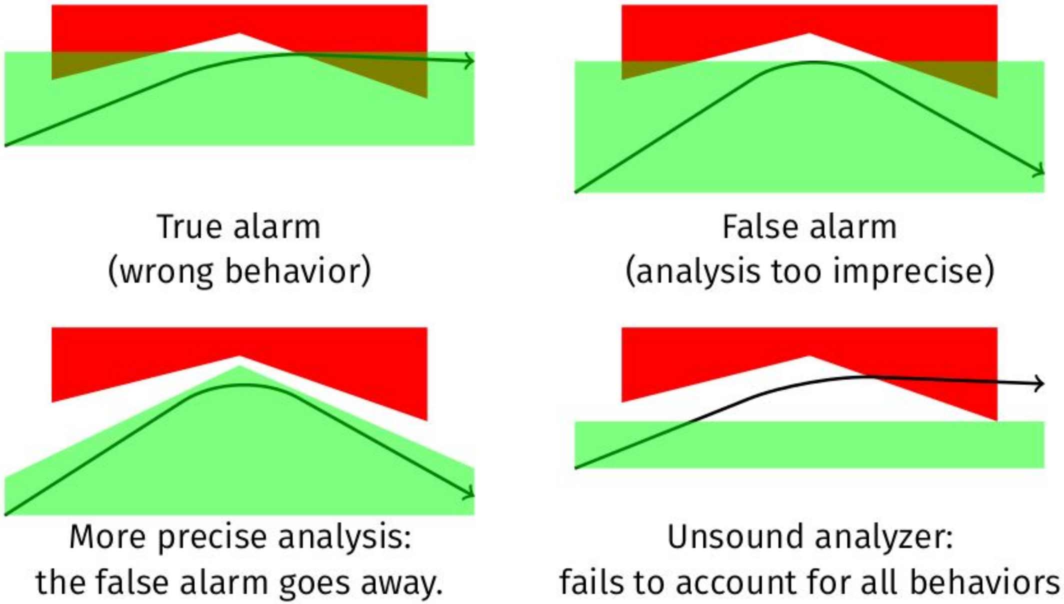
2022/4/19组会分享

Static analyzers

1. **Statically** (The inputs to the program are not known, must terminate, must run in reasonable time and space) **infer** (no help from the programmer) properties of a program that hold for all its executions.
2. They **automatically** prove the absence of certain kinds of bugs (“No invalid memory access”, “No division by 0”)

Abstract interpretation

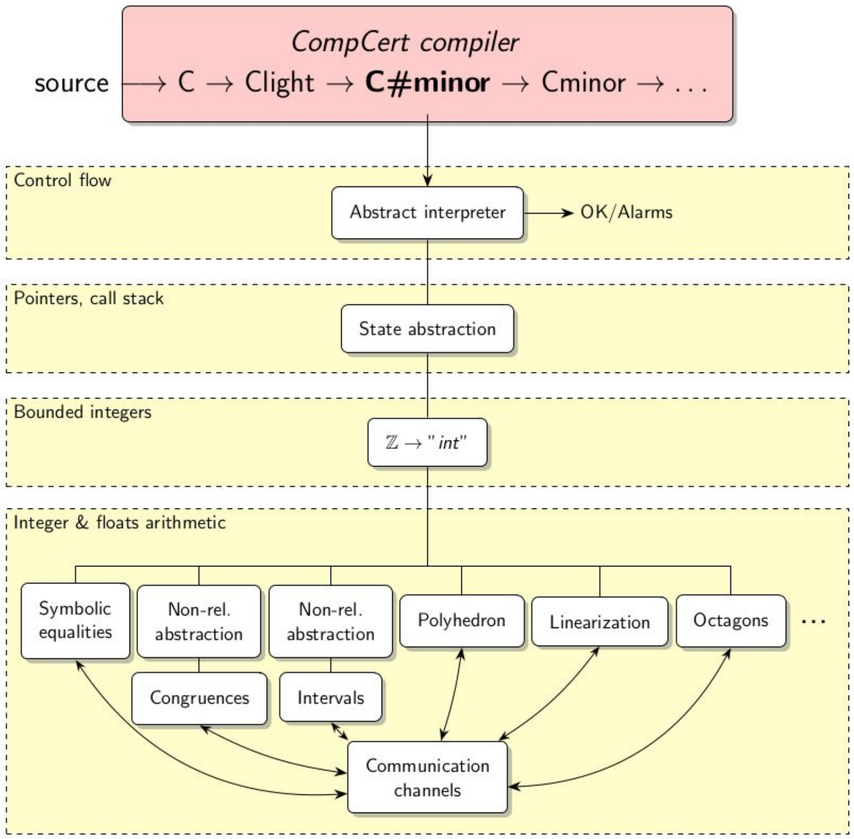
1. Run the program using an **abstract semantics**
2. Always **terminating** computation
3. Soundly **approximating** the concrete semantics



Verasco

1. Programmed and **formally verified** using the Coq proof assistant
2. Based on **abstract interpretation**
3. Handles most of C99
    a. **no dynamic memory allocation, no recursion**
4. Proves the absence of undefined behaviors

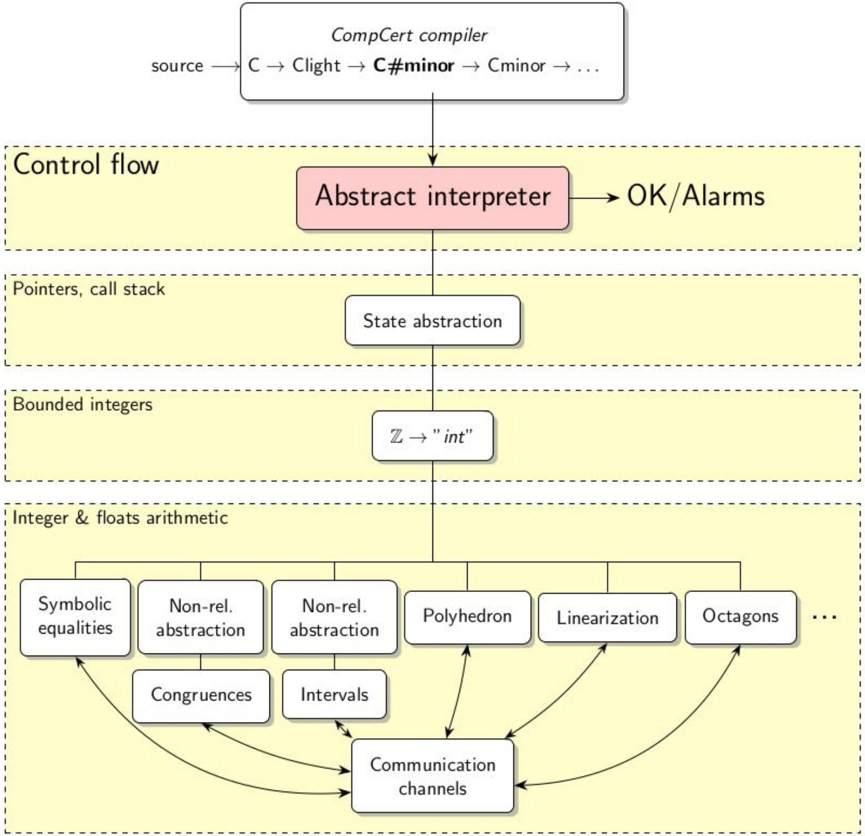
Modular architecture



Reuses the CompCert front-end

1. Until C#minor
2. Uses the same formal semantics as CompCert
3. **Guarantees** provided by the analyzer provably **extend to the assembly code**

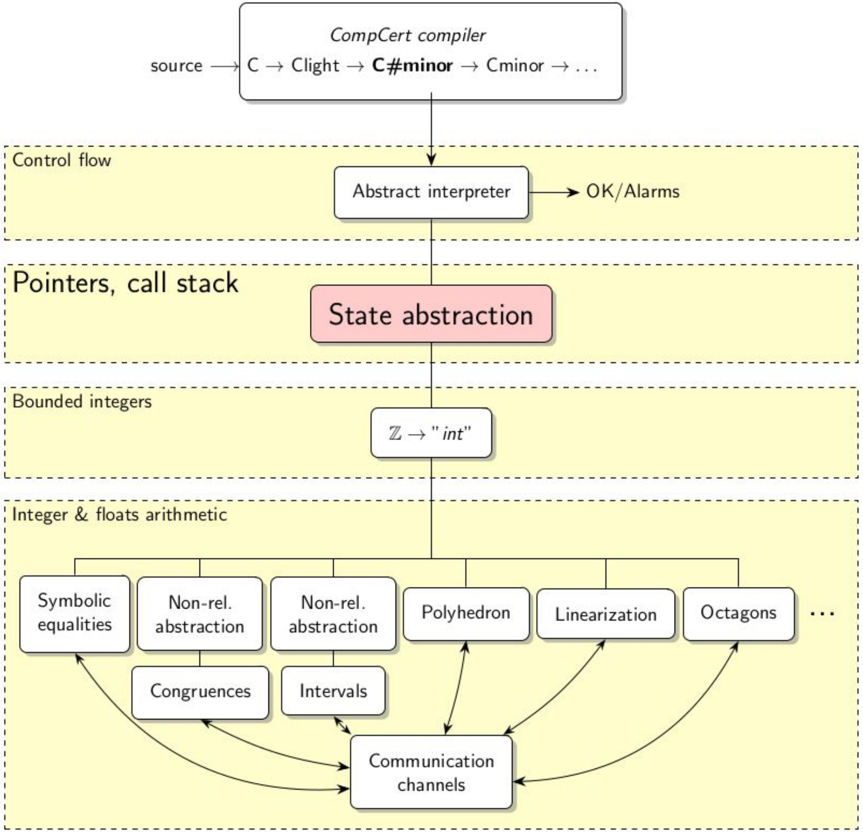
Modular architecture



Abstract interpreter

1. Fixpoints using widening for loops and gotos
2. Proved correct using a Hoare logic for C#minor
3. Parameterized by a state abstract domain

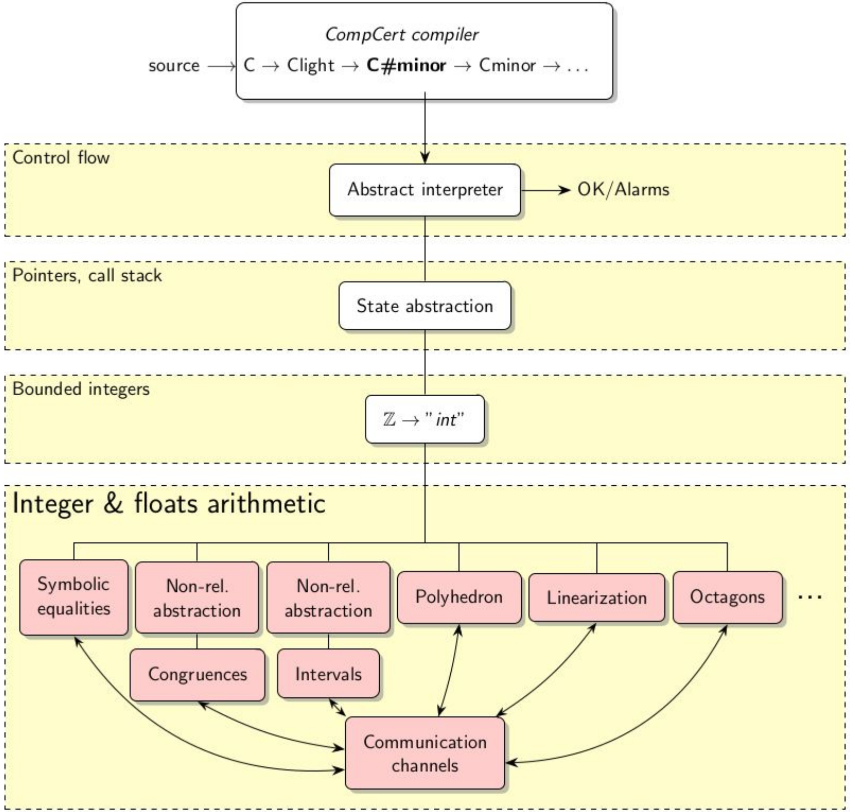
Modular architecture



State abstract domain

1. Solves pointer references
   a. Points-to domain
2. Checks type and memory safety
   a. Types domain
   b. Permissions domain
3. Parameterized by a numerical abstract domain

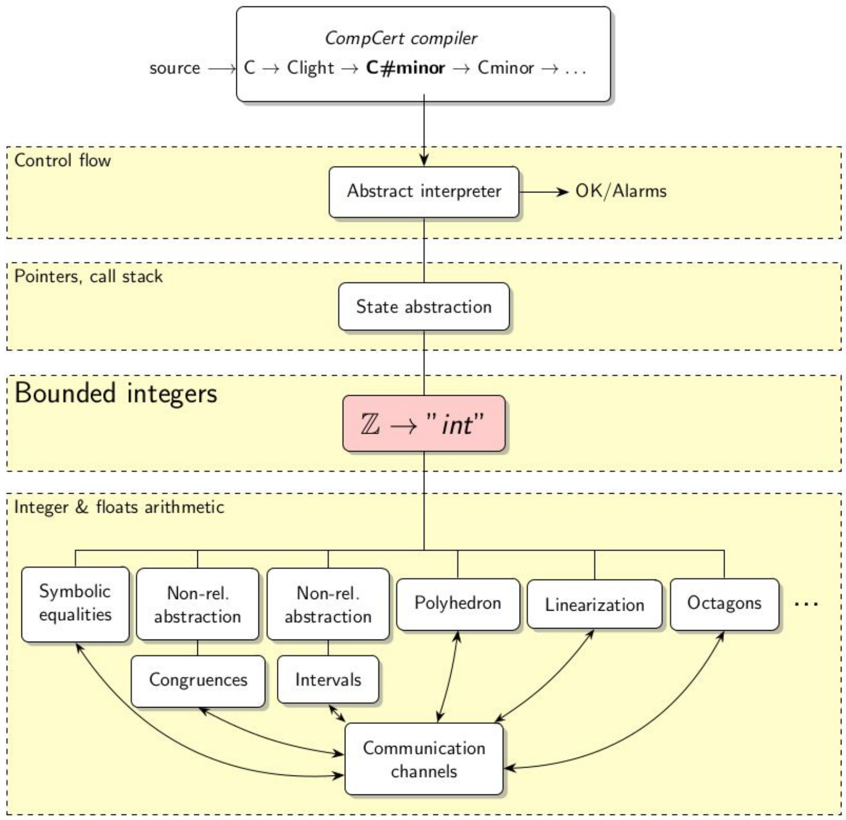
Modular architecture



Several numerical domains

1. Intervals over **Z** and **floats**
2. Arithmetical congruences
3. Symbolic equalities
4. Octagons
5. Convex polyhedra
6. Modular communication system using channels

Modular architecture



Bounded machine integers analysis

1. Wrap-around when overflow
2. Checks numerical errors
   a. Division by 0
   b. Undefined overflows
3. Parameterized by an abstract domain for **Z**

Example 1

State & numerical domains cooperation

```
const double sqrt_tab[128] = { ... };

int main() {
  double x = verasco_any_double();
  verasco_assume(0.3 < x && x < 0.7);
  double y = sqrt_tab[ (int)(x*128.) ];
  verasco_assert(0.54 < y && y < 0.84);
  return 0;
}
```

Integer & float intervals

Precise bounds on result

Example 2

```
int src[10] = { ... };
int dst[10];

int main() {
  int i_src = 0, i_dst = 9;
  while(i_dst >= 0) {
    dst[i_dst] = src[i_src];
    i_dst--; i_src++;
  }
  verasco_assert(i_src == 10);
  return 0;
}
```

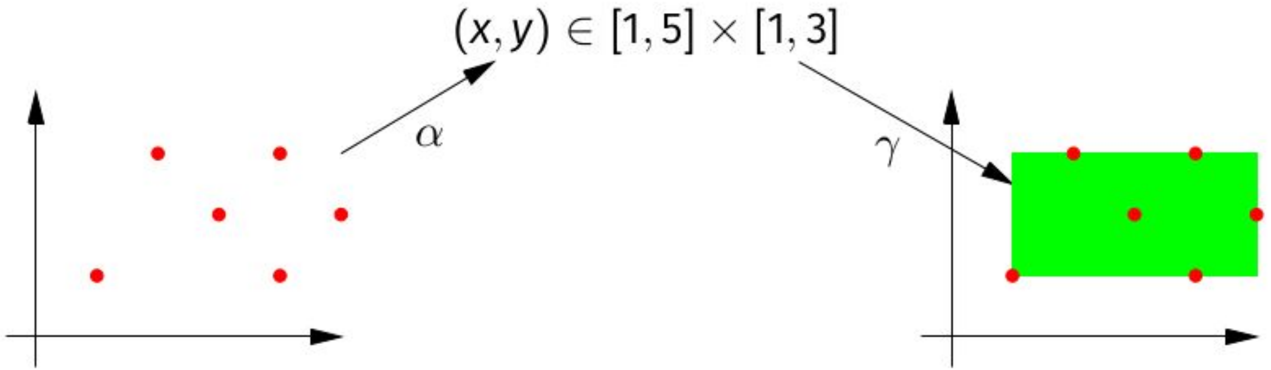
Octagons prove the **relational** invariant $\mathtt{i\_src} + \mathtt{i\_dst} = 9$ ...

... and deduce precise bounds for i_src

A Galois connection

A semi-lattice $\mathcal{A}, \sqsubseteq$ of abstract states and two functions:

- Abstraction α : set of concrete states $\rightarrow$ abstract state
- Concretization γ : abstract state $\rightarrow$ set of concrete states



For intervals, $\alpha(S) = [\inf S, \sup S]$ and $\gamma([a, b]) = \{x \mid a \leq x \leq b\}$.

Calculating abstract operators

For any concrete operator $F : C \to C$ we define its abstraction $F^{\#} : A \to A$ by

$$F^{\#}(a) = \alpha\{F(x) \mid x \in \gamma(a)\}$$

This abstract operator is:

- Sound: if $x \in \gamma(a)$ then $F(x) \in \gamma(F^{\#}(a))$.
- Optimally precise: every a' such that $x \in \gamma(a) \Rightarrow F(x) \in \gamma(a')$ is such that $F^{\#}(a) \sqsubseteq a'$.

Moreover, an algorithmic definition of $F^{\#}$ can be calculated from the definition above.

Getting rid of alpha

1. Soundness: if $x \in \gamma(a)$ then $F(x) \in \gamma(F^{\#}(a))$.
2. Optimality: every a' such that $x \in \gamma(a) \Rightarrow F(x) \in \gamma(a')$ is such that $F^{\#}(a) \sqsubseteq a'$.

Instead of calculating $F^{\#}$, we can guess a definition for $F^{\#}$, then verify

- property 1: soundness (mandatory!)
- possibly property 2: optimality (optional sanity check).

These proofs only need the concretization relation γ, which is unproblematic.

Soundness first!

Having made optimality entirely optional, we can further simplify the analyzer and its soundness proof, while increasing its algorithmic efficiency:

- Abstract operators that return over-approximations (or just $\top$) in difficult / costly cases.
- Join operators $\sqcup$ that return an upper bound for their arguments but not necessarily the least upper bound.
- "Fixpoint" iterations that return a post-fixpoint but not necessarily the smallest (widening + return $\top$ when running out of fuel).

Example of interface (State abstract domain)

- Concrete states:

$$\underbrace{\texttt{list(}\ \underbrace{\texttt{ident}}_{\text{function}}\ \texttt{*}\ \underbrace{\texttt{env}}_{\text{local variables}}\ \texttt{)}}_{\text{call stack}}\ \texttt{*}\ \texttt{mem}$$

Operations

- Standard abstract interpretation operators:
  - $\sqsubseteq$: $\texttt{abstate} \rightarrow \texttt{abstate} \rightarrow \texttt{bool}$
  - $\sqcup$, ∇ : $\texttt{abstate} \rightarrow \texttt{abstate} \rightarrow \texttt{abstate}$
- Abstract transfer functions:
  - `assign`, `store`, `assume`, `push_frame`, `pop_frame`...

Example of interface (State abstract domain)

$$\gamma : \texttt{abstate} \rightarrow \texttt{concrete\_state} \rightarrow \texttt{Prop}$$

- Specifications:
  - $\forall\ a\ b,\ \gamma(a) \cup \gamma(b) \subseteq \gamma(a \sqcup b)$
  - $\forall\ x\ e\ a,\ \{\rho[x := v] \mid \rho \in \gamma(a)\ \wedge\ \rho \vdash e \Downarrow v\} \subseteq \gamma(\texttt{assign}\ x\ e\ a)$

Complex control flow

$\{\top\}$

```
x=0
```

$\{x = 0\}$

```
loop {
```

$\{x \in [0, 13] \wedge x \bmod 2 = 0\}$

```
  if x > 11
```

$\{x = 12\}$

```
    break
```

$\{x \in [0, 11] \wedge x \bmod 2 = 0\}$

```
  x+=2
```

$\{x \in [2, 13] \wedge x \bmod 2 = 0\}$

```
}
```

$\{x = 12\}$

Complex control flow

```
x=0

loop {

  if x > 11
    {x = 12}
    break
    {⊥, x = 12}
  x+=2

}
```

Dedicated program logic:

$$\{P\}\ s\ \{Q,\ Q_b\}$$

Complex control flow

```
x=0

loop {
```
$\{x \in [0, 13] \wedge x \bmod 2 = 0\}$
```
  if x > 11

    break

  x+=2
```
$\{x \in [2, 13] \wedge x \bmod 2 = 0,\ x = 12\}$
```
}
```

Dedicated program logic:

$$\{P\}\ s\ \{Q,\ Q_b\}$$

Complex control flow

```
x=0
```
$\{x = 0\}$
```
loop {
  if x > 11
    break
  x+=2
}
```
$\{x = 12,\ \bot\}$

Dedicated program logic:

$$\{P\}\ s\ \{Q,\ Q_b\}$$

Abstract interpreter

```
Fixpoint iter (ab:abstate) (s:stmt) {struct s}
        : abstate * abstate := ...
```

Proof step 1: interpreter soundness

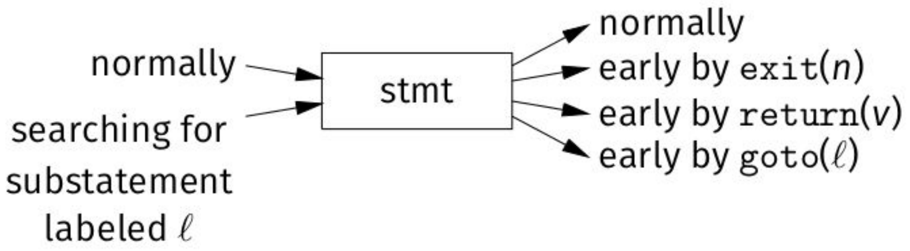
$$\texttt{Lemma iter\_ok: } \forall\ \texttt{ab\_pre stmt ab\_post ab\_break,}$$
$$\texttt{iter ab\_pre stmt} = \texttt{(ab\_post, ab\_break)} \rightarrow$$
$$\{\ \gamma(\texttt{ab\_pre})\ \}\ \texttt{stmt}\ \{\ \gamma(\texttt{ab\_post}), \gamma(\texttt{ab\_break})\ \}.$$

Proof step 2: program logic soundness

$$\{\cdot\}\ \cdot\ \{\cdot,\ \cdot\} \Longrightarrow \text{No undefined behavior}$$

Control flow in the C#minor language

Unlike in IMP, a C#minor statement can terminate in several different ways, and can also be entered in several ways:



The abstract interpreter becomes:

$$F(s, A_i, A_l) = (A_o, A_r, A_e, A_g) + \texttt{alarm}$$

A_i : abstract state (normal entry)
A_l : label $\to$ abstract state (incoming $\texttt{goto}$)
A_o : abstract state (normal termination)
A_r : abstract value $\times$ abstract state (early return)
A_e : exit level $\to$ abstract state
A_g : label $\to$ abstract state (outgoing $\texttt{goto}$)

NB: for C#, we need Hoare "7-tuples"
$\{\gamma(A_i), \gamma(A_l)\}\ s\ \{\gamma(A_o), \gamma(A_r), \gamma(A_e), \gamma(A_g)\}$.

Abstract interpreter

1. Handles all C#minor control constructs
    a. **2 pre-conditions, 4 post-conditions**
2. Works in a monad for alarms
3. Structural recursion on syntax tree
    a. Unfolding functions at call sites
4. Fixpoint iteration using **widening** and **narrowing**
    a. One fixpoint iteration per loop
    b. Gotos: one fixpoint iteration per function

Fixpoints

Theorem (Kleene)

Let $(A, \sqsubseteq, \bot)$ a partially ordered set such that $\sqsupset$ is well founded (no infinite increasing sequences).
Let $F : A \to A$ an increasing, continuous function.
Then F has a smallest fixpoint, obtained by finite iteration from $\bot$:

$$\exists n, \ \ \bot \sqsubset F(\bot) \sqsubset \ldots \sqsubset F^n(\bot) = F^{n+1}(\bot)$$

Fixpoints

Most abstract domains are not well founded. Example:

$$\text{Integer intervals: } [0, 0] \sqsubset [0, 1] \sqsubset [0, 2] \sqsubset \cdots \sqsubset [0, n] \sqsubset \cdots$$

Moreover, even when Kleene iteration converges, it converges too slowly:

```
x = 0;  while (x <= 10000) { x = x + 1; }
```

(Starting with $x^{\#} = [0, 0]$, it takes 10000 iterations to reach the fixpoint $x^{\#} = [0, 10000]$.)

Fixpoints: widening

A widening operator $\nabla : A \to A \to A$ computes a majorant of its second argument in such a way that the following iteration converges always and quickly:

$$X_0 = \bot \qquad X_{i+1} = \begin{cases} X_i & \text{if } F(X_i) \sqsubseteq X_i \\ X_i \nabla F(X_i) & \text{otherwise} \end{cases}$$

The limit X of this sequence is a post-fixpoint: $F(X) \sqsubseteq X$.

Example: widening for intervals:

$$\begin{aligned}[l_1, u_1] \nabla [l_2, u_2] = \ & [\texttt{if } l_2 < l_1 \texttt{ then } -\infty \texttt{ else } l_1, \\ & \ \texttt{if } u_2 > u_1 \texttt{ then } \infty \texttt{ else } u_1]\end{aligned}$$

Fixpoints: narrowing

The quality of the post-fixpoint can be improved by iterating F some more:

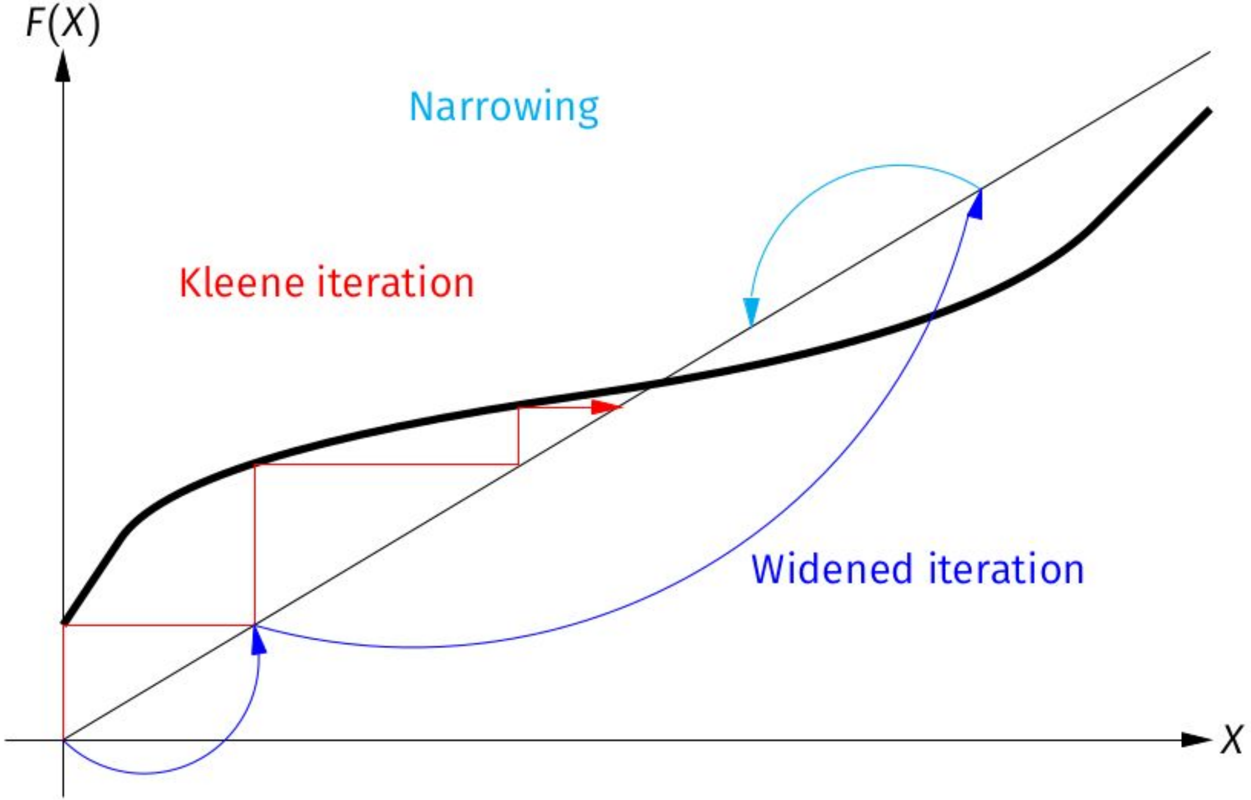
$$Y_0 = \text{a post-fixpoint} \qquad Y_{i+1} = F(Y_i)$$

If F is increasing, each Y_i is a post-fixpoint: $F(Y_i) \sqsubseteq Y_i$.

Often, $Y_i \sqsubset Y_0$, improving the analysis quality.

Iteration can be stopped when Y_i is a fixpoint, or at any time.

Fixpoints: widening + narrowing



Some cases of the abstract interpreter F

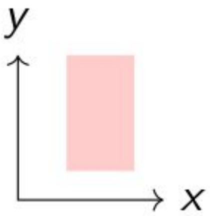
$$
\begin{aligned}
F((s_1; s_2), A) &= F(s_2, F(s_1, A)) \\
F((\texttt{IF}\ b\ \texttt{THEN}\ s_1\ \texttt{ELSE}\ s_2), A) &= F(s_1, A \wedge b) \sqcup F(s_2, A \wedge \neg b) \\
F((\texttt{WHILE}\ b\ \texttt{DO}\ s\ \texttt{DONE}), A) &= \texttt{pfp}\ (\lambda X.\ A \sqcup F(s, X \wedge b)) \wedge \neg b
\end{aligned}
$$

$$
\texttt{pfp}\ F\ A\ N = \begin{cases} \top & \text{if } N = 0 \\ \texttt{narrow}\ F\ A\ N_{\text{narrow}} & \text{if } A \sqsupseteq F\ A \\ \texttt{pfp}\ F\ (A\ \nabla\ F\ A)\ (N-1) & \text{otherwise} \end{cases}
$$

$$
\texttt{narrow}\ F\ A\ N = \begin{cases} A & \text{if } N = 0 \\ \texttt{narrow}\ F\ (F\ A)\ (N-1) & \text{if } A \sqsupseteq F\ A \\ A & \text{otherwise} \end{cases}
$$

Numerical abstract domains in Verasco

Intervals

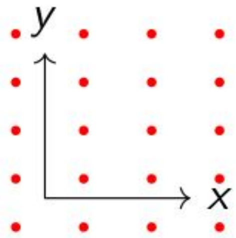


$x \in [l, h]$

Symbolic equalities

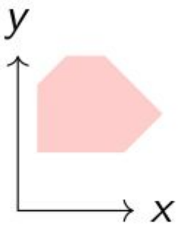
$$z \doteq y * y$$
$$c \doteq (x < 1 \ \&\& \ 2 \leq y)$$
$$(y \leq 2.5) \doteq \ \mathtt{true}$$

Congruences



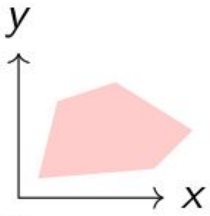
$x \bmod M = R$

Octagons



$\pm x \pm y \leq \mathsf{Cst}$

Polyhedron



$\sum_{i=0}^{n} \alpha_i x_i \leq \mathsf{Cst}$

Octagons

- Very popular **weakly relational domain**
  - Originally by Miné from Astrée
- Inequalities of the form: $\pm x \pm y \leq \text{Cst}$
- Interval constraints expressible:

$$x + x \leq 2h \qquad - x - x \leq -2l$$

- Data structure: **difference bound matrix** A_{xy} for x, y **signed variables**:

$$\forall x\ y,\ x + y \leq A_{xy}$$

Octagons

$$
\begin{array}{c} \\ +x \\ -x \\ +y \\ -y \\ +z \\ -z \end{array}
\begin{array}{c}
\begin{array}{cccccc} -x & +x & -y & +y & -z & +z \end{array} \\
\begin{pmatrix}
0 & 2 & & & & \\
0 & 0 & & & & \\
 & & 0 & 6 & & \\
 & & -2 & 0 & & \\
 & & & & 0 & 4 \\
 & & & & 0 & 0
\end{pmatrix}
\end{array}
$$

Interval constraints:

$$x \in [0, 1]$$
$$y \in [1, 3]$$
$$z \in [0, 2]$$

Octagons

$$\begin{array}{c} \\ +x \\ -x \\ +y \\ -y \\ +z \\ -z \end{array}\begin{array}{c} \begin{array}{cccccc} -x & +x & -y & +y & -z & +z \end{array} \\ \begin{pmatrix} 0 & 2 & 0 & 4 & 1 & 3 \\ 0 & 0 & -1 & 3 & 0 & 2 \\ 3 & 4 & 0 & 6 & 3 & 5 \\ -1 & 0 & -2 & 0 & -1 & 1 \\ 2 & 3 & 1 & 5 & 0 & 4 \\ 0 & 1 & -1 & 3 & 0 & 0 \end{pmatrix} \end{array}$$

Strong saturation
$\Longrightarrow$ Dense matrix

Octagons

$$
\begin{array}{c}
\\
\begin{array}{c} +x \\ -x \\ +y \\ -y \\ +z \\ -z \end{array}
\end{array}
\begin{array}{c}
\begin{array}{cccccc} -x & +x & -y & +y & -z & +z \end{array} \\
\begin{pmatrix}
0 & 2 & & 2 & & \\
0 & 0 & & & & \\
 & 2 & 0 & 6 & & \\
 & & -2 & 0 & & \\
 & & & & 0 & 4 \\
 & & & & 0 & 0
\end{pmatrix}
\end{array}
$$

Adding the constraint:

$$x + y \leq 2$$

Octagons

$$
\begin{array}{c|cccccc}
 & -x & +x & -y & +y & -z & +z \\
\hline
+x & 0 & 2 & 0 & 2 & & \\
-x & 0 & 0 & -1 & 2 & & \\
+y & 2 & 2 & 0 & 4 & & \\
-y & -1 & 0 & -2 & 0 & & \\
+z & & & & & 0 & 4 \\
-z & & & & & 0 & 0
\end{array}
$$

Weak saturation:

$$y + y \leq 4$$

$$y - x \leq 2$$

$$-y - x \leq -1$$

$$x - y \leq 0$$

Abstract domain of symbolic equalities

CompCert front-end transformation:

```
if(0 <= a && a < 10) {
  ...
}
```

$\Longrightarrow$

```
if(0 <= a)
  tmp = a < 10;
else
  tmp = 0;
if(tmp) {
  ...
}
```

Need to reason on the **relation** between a and `tmp`.

Abstract domain of symbolic equalities

- Two kinds of equalities:
  - $\texttt{var} \doteq \texttt{expr}$
  - $\texttt{expr} \doteq \texttt{bool}$

- Example:

`if(0 <= a)`	$(\texttt{0 <= a}) \doteq \texttt{true}$	
`  tmp = a < 10;`	$(\texttt{0 <= a}) \doteq \texttt{true}$	$\texttt{tmp} \doteq \texttt{a < 10}$
`else`	$(\texttt{0 <= a}) \doteq \texttt{false}$	
`  tmp = 0;`	$(\texttt{0 <= a}) \doteq \texttt{false}$	$\texttt{tmp} \doteq 0$
		$\texttt{tmp} \doteq (\texttt{0 <= a})\ \texttt{?}\ \texttt{a < 10} : 0$
`if(tmp) {`	Unfolding `tmp`	
`  ...`	$a \in [0, 9]$	
`}`		

Final theorem

```
Definition vanalysis prog = ... iter ...

Theorem vanalysis_correct:
 ∀ prog res tr,
  vanalysis prog = (res, nil) →
  program_behaves (semantics prog) (Goes_wrong tr) →
  False.
```

Empty alarm list

No wrong behavior

Reference

Talk:

https://gdr-gpl.cnrs.fr/sites/default/files/documentsGPL/JourneesNationales/GPL2017/Jourdan_talk.pdf

https://chocola.ens-lyon.fr/media/talk212/Chocola-2019-04-Leroy.pdf

Paper:

A Formally-Verified C Static Analyzer https://xavierleroy.org/publi/verasco-popl2015.pdf

Vedio:

Verasco, a formally verified C static analyzer - YouTube