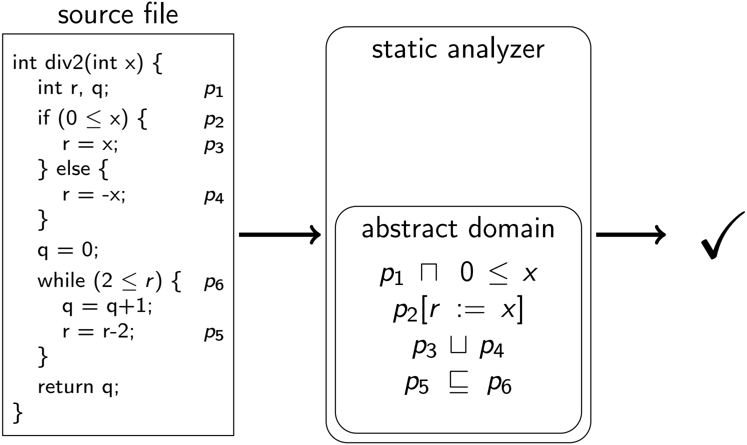
Modular and lightweight certification of polyhedral abstract domains

Alexis Fouilhe   Sylvain Boulmé   Michaël Périn

Verimag, Grenoble

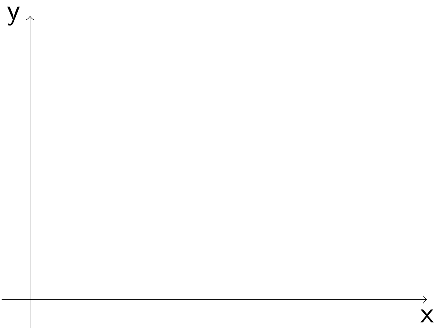
May 14, 2014

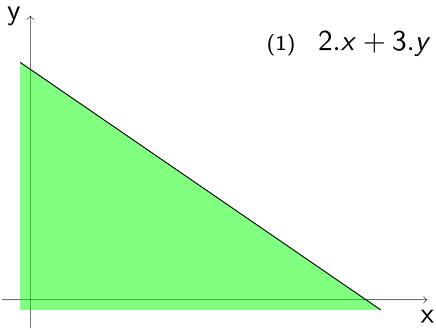
Modular and lightweight certification of polyhedral abstract domains



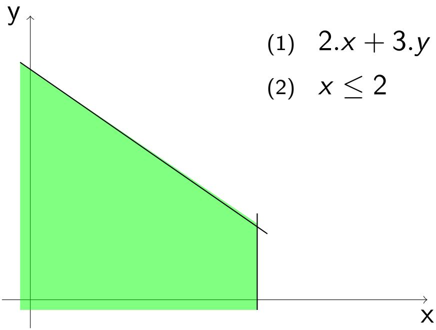
source file
int div2(int x) {
int r, q; p_1
if (0 ≤ x) { p_2
r = x; p_3
} else {
r = -x; p_4
}
q = 0;
while (2 ≤ r) { p_6
q = q+1;
r = r-2; p_5
}
return q;
}
static analyzer
abstract domain
$p_1 \sqcap 0 \leq x$
$p_2[r := x]$
$p_3 \sqcup p_4$
$p_5 \sqsubseteq p_6$
✓

Modular and lightweight certification of
polyhedral abstract domains


y
x

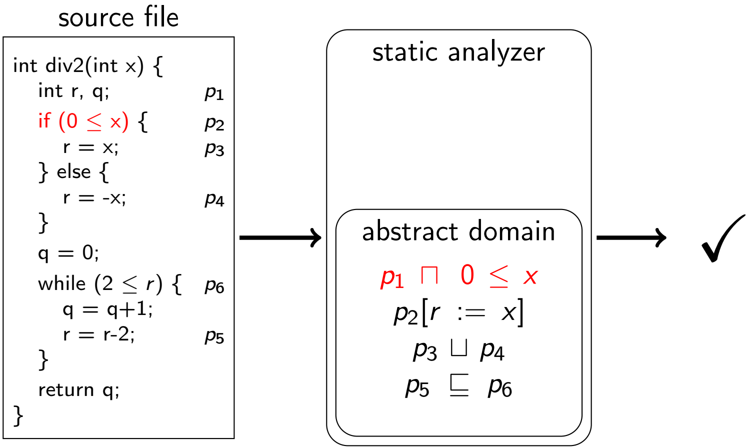


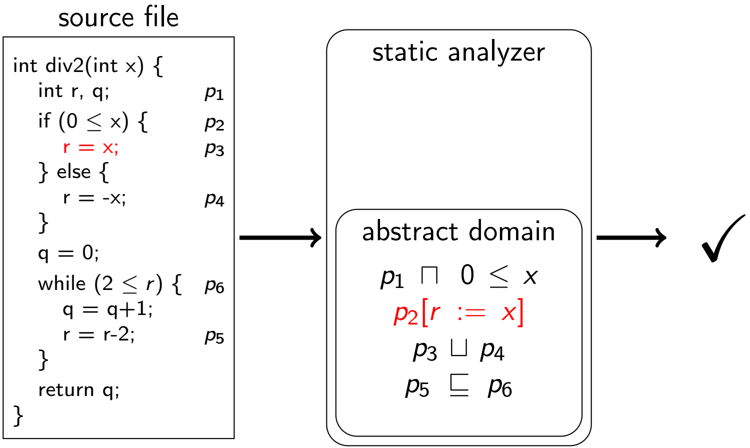
(1) $2.x + 3.y \leq 6$

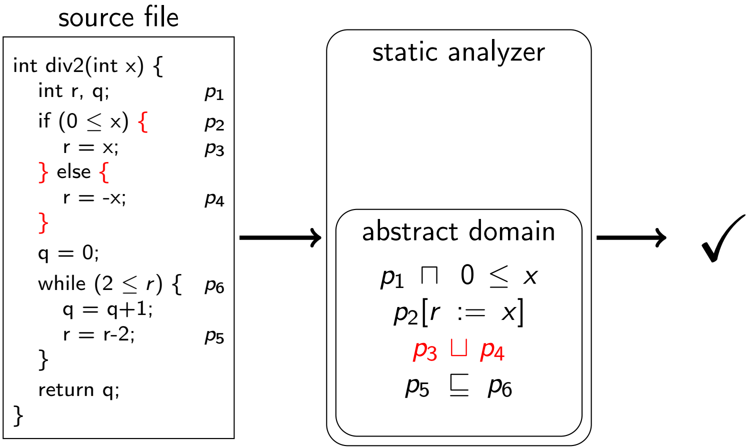


(1) $2.x + 3.y \leq 6$

(2) $x \leq 2$


source file
int div2(int x) {
int r, q; p_1
if (0 ≤ x) { p_2
r = x; p_3
} else {
r = -x; p_4
}
q = 0;
while (2 ≤ r) { p_6
q = q+1;
r = r-2; p_5
}
return q;
}
static analyzer
abstract domain
$p_1 \sqcap 0 \leq x$
$p_2[r := x]$
$p_3 \sqcup p_4$
$p_5 \sqsubseteq p_6$
✓


source file
int div2(int x) {
int r, q; p1
if (0 ≤ x) { p2
r = x; p3
} else {
r = -x; p4
}
q = 0;
while (2 ≤ r) { p6
q = q+1;
r = r-2; p5
}
return q;
}
static analyzer
abstract domain
p1 ⊓ 0 ≤ x
p2[r := x]
p3 ⊔ p4
p5 ⊑ p6
✓


source file
int div2(int x) {
int r, q; p_1
if (0 ≤ x) { p_2
r = x; p_3
} else {
r = -x; p_4
}
q = 0;
while (2 ≤ r) { p_6
q = q+1;
r = r-2; p_5
}
return q;
}
static analyzer
abstract domain
$p_1 \sqcap 0 \leq x$
$p_2[r := x]$
$p_3 \sqcup p_4$
$p_5 \sqsubseteq p_6$
✓

source file

```
int div2(int x) {
    int r, q;                 p1
    if (0 ≤ x) {              p2
        r = x;                p3
    } else {
        r = -x;               p4
    }
    q = 0;
    while (2 ≤ r) {           p6
        q = q+1;
        r = r-2;              p5
    }
    return q;
}
```

static analyzer

abstract domain

$p_1 \sqcap 0 \leq x$

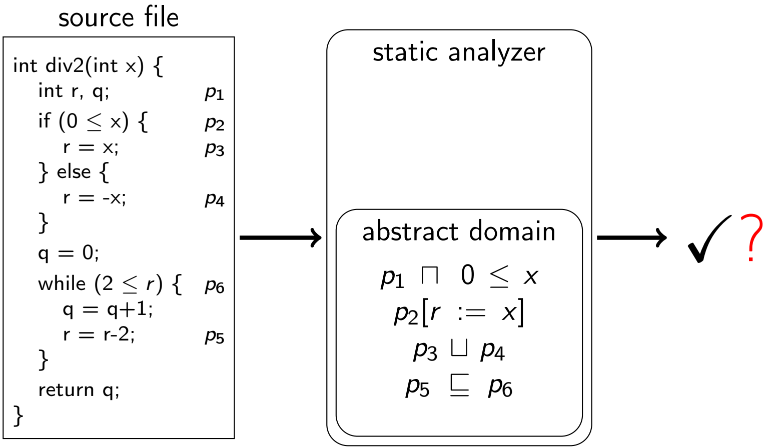
$p_2[r := x]$

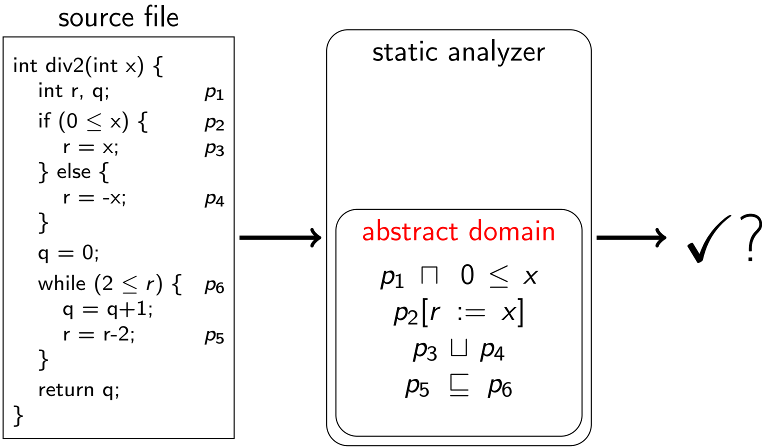
$p_3 \sqcup p_4$

$p_5 \sqsubseteq p_6$

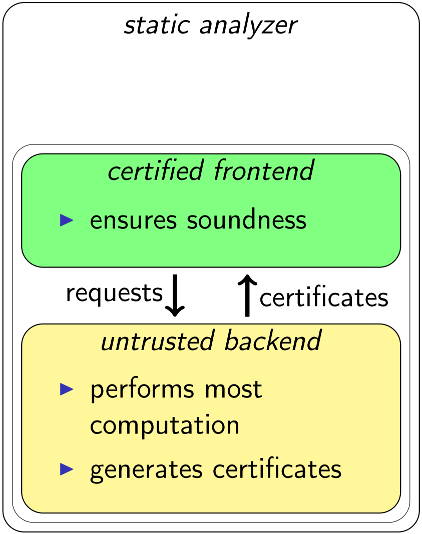
✓

Modular and lightweight certification of polyhedral abstract domains

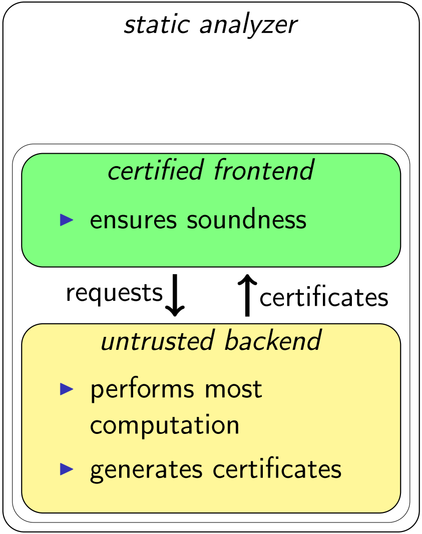

source file
int div2(int x) {
int r, q; p_1
if (0 ≤ x) { p_2
r = x; p_3
} else {
r = -x; p_4
}
q = 0;
while (2 ≤ r) { p_6
q = q+1;
r = r-2; p_5
}
return q;
}
static analyzer
abstract domain
$p_1 \sqcap 0 \leq x$
$p_2[r := x]$
$p_3 \sqcup p_4$
$p_5 \sqsubseteq p_6$
✓?


source file
int div2(int x) {
int r, q; p1
if (0 ≤ x) { p2
r = x; p3
} else {
r = -x; p4
}
q = 0;
while (2 ≤ r) { p6
q = q+1;
r = r-2; p5
}
return q;
}
static analyzer
abstract domain
p1 ⊓ 0 ≤ x
p2[r := x]
p3 ⊔ p4
p5 ⊑ p6
✓?

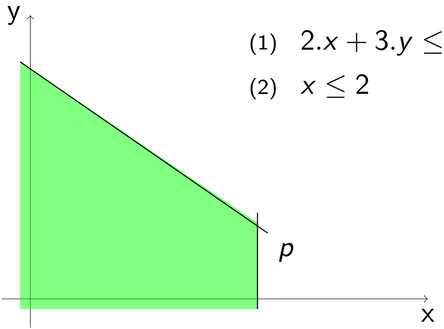
Modular and lightweight certification of polyhedral abstract domains



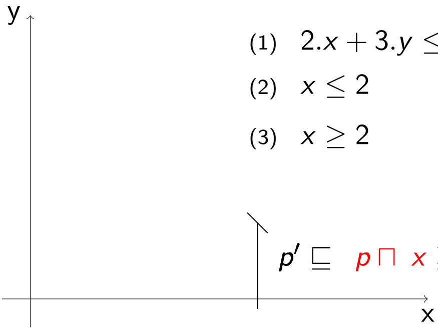
- perfect fit for result verification
- build results from certificates
- formalize impure external code



- perfect fit for result verification
- build results from certificates
- formalize impure external code



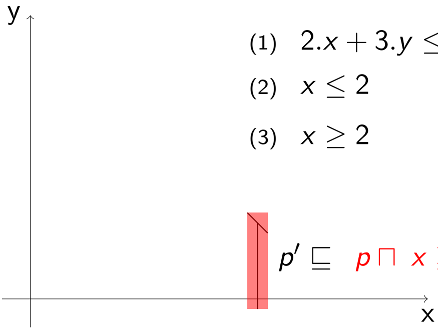
$$\left.\begin{array}{ll}(1) & 2.x + 3.y \leq 6 \\ (2) & x \leq 2\end{array}\right\} p$$



$$\left.\begin{array}{ll} (1) & 2.x + 3.y \leq 6 \\ (2) & x \leq 2 \end{array}\right\} p$$

$$(3) \quad x \geq 2$$

$$p' \sqsubseteq \ p \sqcap \ x \geq 2$$

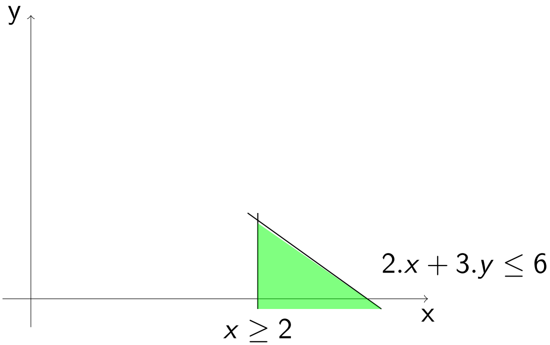


$$\left.\begin{array}{ll}(1) & 2.x + 3.y \leq 6 \\ (2) & x \leq 2\end{array}\right\} p$$

$$(3) \quad x \geq 2$$

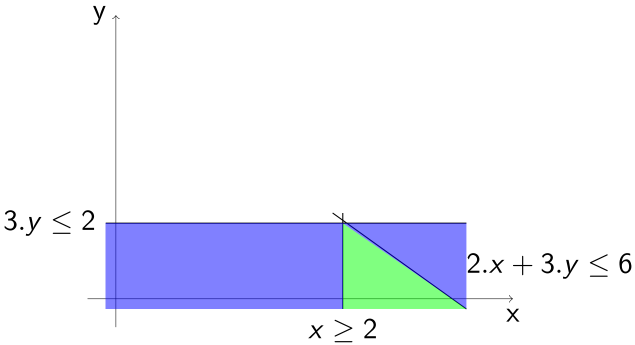
$$p' \sqsubseteq \; p \sqcap \; x \geq 2$$

Farkas's lemma:



Farkas's lemma:

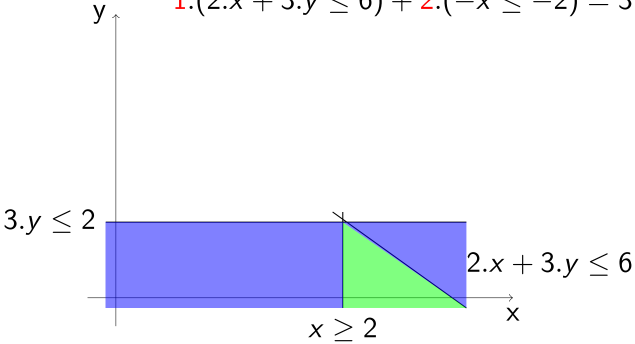
$\lambda_1.(2.x + 3.y \leq 6) + \lambda_2.(-x \leq -2) = 3.y \leq 2$

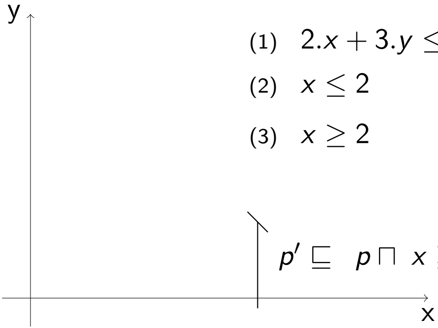


Farkas's lemma:

$$\lambda_1.(2.x + 3.y \leq 6) + \lambda_2.(-x \leq -2) = 3.y \leq 2$$

$$1.(2.x + 3.y \leq 6) + 2.(-x \leq -2) = 3.y \leq 2$$





$$\left.\begin{array}{ll}(1) & 2.x + 3.y \leq 6 \\ (2) & x \leq 2\end{array}\right\} p$$

$$(3) \quad x \geq 2$$

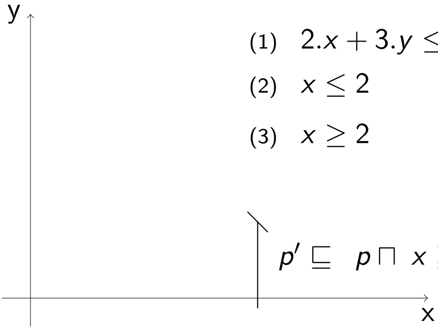
$$p' \sqsubseteq \; p \sqcap \; x \geq 2$$

$$1.(2), 1.(3) \quad x = 2$$

$$1.(1) + 2.(3) \quad 3.y \leq 2$$

$$\left.\begin{array}{ll}(1) & 2.x + 3.y \leq 6 \\ (2) & x \leq 2\end{array}\right\} p$$

$$(3) \quad x \geq 2$$



$$p' \sqsubseteq \; p \sqcap \; x \geq 2$$

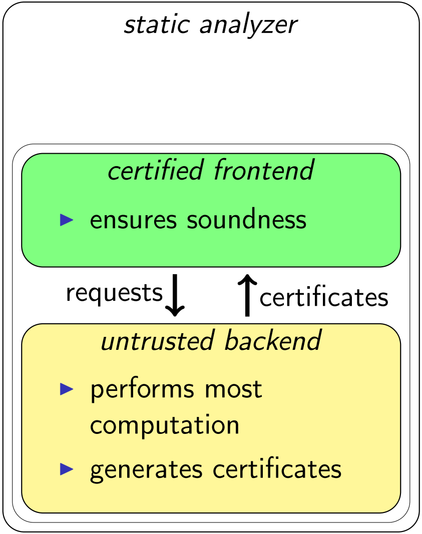
$1.(2), 1.(3) \quad x = 2$

$1.(1) + 2.(3) \quad 3.y \leq 2$

$1.(1) \quad 2.x + 3.y \leq 6$

$1.(2) \quad x \leq 2$

$1.(3) \quad x \geq 2$



- perfect fit for result verification
- build results from certificates
- formalize impure external code

Backend.mli

```
val f : nat → nat;;
```

```
Axiom f : nat → nat.
Extract Constant f ⇒ "Backend.f".
```

```
Goal ∀ n a b, f n = a → f n = b → a = b.
intros. subst. reflexivity. Qed.
```

Backend.mli

```
val f : nat → nat;;
```

```
Axiom f : nat → nat.
Extract Constant f ⇒ "Backend.f".

Goal ∀ n a b, f n = a → f n = b → a = b.
intros. subst. reflexivity. Qed.
```

Backend.ml

```
let c = ref 0;;

let f n = begin
   c := n + !c;
   !c;
   end;;
```

Our backend uses GMP.

Backend.mli

```
val f : nat → nat;;
```

```
Axiom f : nat → ?nat.
Extract Constant f ⇒ "Backend.f".

Goal ∀ n a b, f n ⇝ a → f n ⇝ b → a = b.
(* can't prove it *)
```

Backend.ml

```
let c = ref 0;;

let f n = begin
    c := n + !c;
    !c;
    end;;
```

Our backend uses GMP.

may-return monad

$$
\begin{aligned}
\mathtt{?} &: \mathtt{Type} \to \mathtt{Type} \\
\mathtt{unit} &: \mathtt{A} \to \mathtt{?A} \\
\mathtt{bind} &: \mathtt{?A} \to (\mathtt{A} \to \mathtt{?B}) \to \mathtt{?B} \\
\rightsquigarrow &: \mathtt{?A} \to \mathtt{A} \to \mathtt{Prop}
\end{aligned}
$$

$\mathtt{unit}\ \mathtt{a}_1 \rightsquigarrow \mathtt{a}_2 \Rightarrow \mathtt{a}_1 = \mathtt{a}_2$

$\mathtt{bind}\ \mathtt{k}_1\ \mathtt{k}_2 \rightsquigarrow \mathtt{b} \Rightarrow \exists\ \mathtt{a},\ \mathtt{k}_1 \rightsquigarrow \mathtt{a} \wedge \mathtt{k}_2\ \mathtt{a} \rightsquigarrow \mathtt{b}$

may-return monad

$$
\begin{array}{lcl}
\texttt{?} & : & \texttt{Type} \to \texttt{Type} \\
\texttt{unit} & : & \texttt{A} \to \texttt{?A} \\
\texttt{bind} & : & \texttt{?A} \to (\texttt{A} \to \texttt{?B}) \to \texttt{?B} \\
\leadsto & : & \texttt{?A} \to \texttt{A} \to \texttt{Prop}
\end{array}
$$

$$\texttt{unit}\ \texttt{a}_1 \leadsto \texttt{a}_2 \Rightarrow \texttt{a}_1 = \texttt{a}_2$$

$$\texttt{bind}\ \texttt{k}_1\ \texttt{k}_2 \leadsto \texttt{b} \Rightarrow \exists\ \texttt{a},\ \texttt{k}_1 \leadsto \texttt{a} \wedge \texttt{k}_2\ \texttt{a} \leadsto \texttt{b}$$

one implementation: the state monad

$$
\begin{array}{lcl}
\texttt{?A} & = & \texttt{S} \to \texttt{A} \times \texttt{S} \\
\texttt{unit a} & = & \lambda\ \texttt{s.(a,s)} \\
\texttt{bind}\ \texttt{k}_1\ \texttt{k}_2 & = & \lambda\ \texttt{s}_0,\ \texttt{let (a, s}_1\texttt{) = k}_1\ \texttt{s}_0\ \texttt{in k}_2\ \texttt{a s}_1 \\
\texttt{k} \leadsto \texttt{a} & = & \exists\ \texttt{s, fst (k s) = a}
\end{array}
$$

may-return monad

$$
\begin{array}{lcl}
\texttt{?} & : & \texttt{Type} \rightarrow \texttt{Type} \\
\texttt{unit} & : & \texttt{A} \rightarrow \texttt{?A} \\
\texttt{bind} & : & \texttt{?A} \rightarrow (\texttt{A} \rightarrow \texttt{?B}) \rightarrow \texttt{?B} \\
\leadsto & : & \texttt{?A} \rightarrow \texttt{A} \rightarrow \texttt{Prop}
\end{array}
$$

$$\texttt{unit } \texttt{a}_1 \leadsto \texttt{a}_2 \Rightarrow \texttt{a}_1 = \texttt{a}_2$$

$$\texttt{bind } \texttt{k}_1\ \texttt{k}_2 \leadsto \texttt{b} \Rightarrow \exists\ \texttt{a},\ \texttt{k}_1 \leadsto \texttt{a} \land \texttt{k}_2\ \texttt{a} \leadsto \texttt{b}$$

for extracting: the identity monad

$$
\begin{array}{lcl}
\texttt{?A} & = & \texttt{A} \\
\texttt{unit a} & = & \texttt{a} \\
\texttt{bind } \texttt{k}_1\ \texttt{k}_2 & = & \texttt{k}_2\ \texttt{k}_1 \\
\texttt{k} \leadsto \texttt{a} & = & \texttt{k} = \texttt{a}
\end{array}
$$

\+ inlining

Perfect fit for result verification

- simple maths: easy COQ proofs
- build results from certificates: efficient communication
- complex result search

Formalization of external code

- COQ checks we don't use the purity assumption
- may-return monad
  - no runtime overhead
  - low proof overhead

Engineering: a certified abstract domain

- simple/modular formalization
- generic w.r.t. the backend
- experiments show reasonable performance
- integration in the VERASCO analyzer