Deductive runtime certification

Konstantine Arkoudas & Martin Rinard

Result correctness

We have a program P and an input x.

We run it: $P(x) \leadsto$ *result* r.

How do we know that r is correct?

Two prominent approaches:

1. *Verify* P. I.e., prove:

$$\forall\, x \,.\, P(x)\downarrow \Rightarrow \mathit{Correct}(P(x), x)$$

2. *Test* the result r.

Verification is powerful but difficult.

Testing is easier but has limited scope.

Another approach

Express the program P as a theorem prover.

Instead of producing a data value

$$P(x) \leadsto r$$

we prove a theorem

$$P(x) \leadsto \vdash \mathit{Correct}(r, x)$$

If we trust the axioms and inference rules, we can trust t

Pros:

- Potentially large reduction of the trusted base.
- Much easier than total verification.

Cons:

- Guarantee applies only to particular results.

Example: Euclid's algorithm

A conventional (recursive) formulation:

$$\begin{aligned} euclid(a,b) = {} & \text{If } b \text{ is 0 then return } a \\ & \text{else return } euclid(b, a \bmod b). \end{aligned}$$

Suppose we get euclid$(784, 512) = 16$.

Is that correct??

We can test the result 16.

Or we can *verify* the algorithm *euclid.*

For verification, we need to use strong induction to prov

$$\forall\, x, y \;.\; euclid(x,y) = gcd(x,y)$$

The proof relies on two results:

A1. $\forall\, x \;.\; gcd(x,0) = x$

A2. $\forall\, x, y \;.\; y > 0 \Rightarrow gcd(x,y) = gcd(y, x \bmod y)$

A theorem-proving formulation

A1. $\forall\ x\ .\ gcd(x, 0) = x$

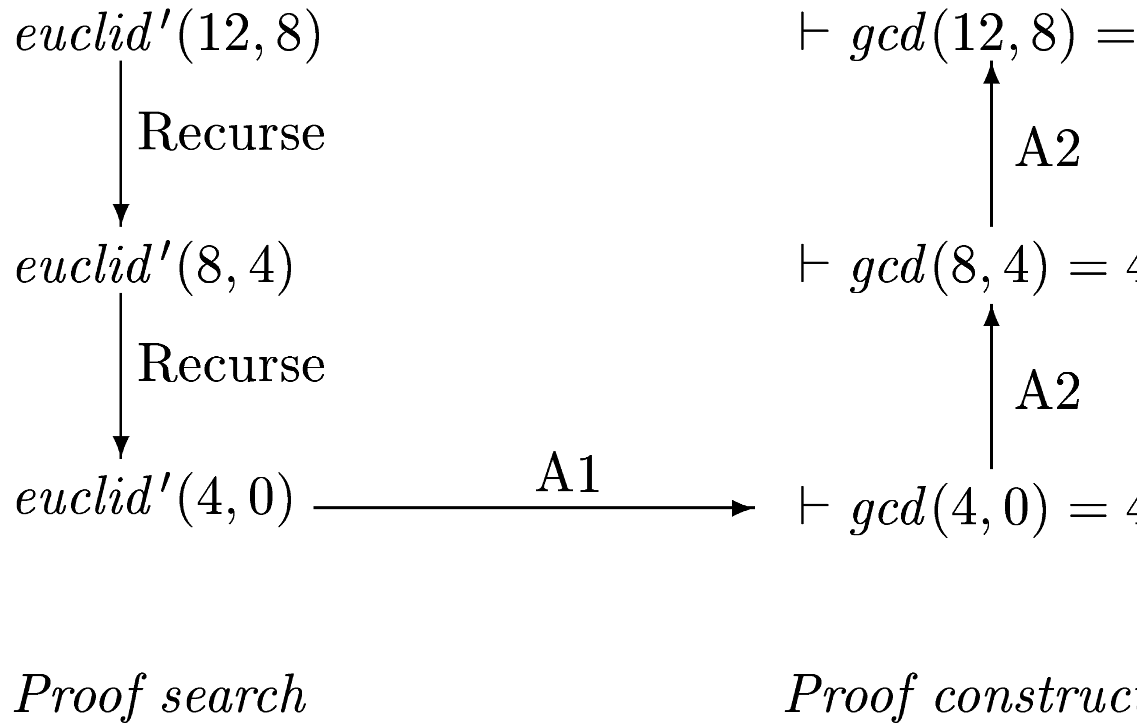
A2. $\forall\ x, y\ .\ y > 0 \Rightarrow gcd(x, y) = gcd(y, x \bmod y)$

$euclid'(a, b) =$ If b is 0 derive $\vdash \mathrm{gcd}(a, b) = a$ from A1.	
If $b > 0$ then:	1. Let $m = a \bmod b$;
	2. Call $euclid'(b, m)$ to get $\vdash \mathrm{gcd}(b, m) =$
	3. Derive $\vdash \mathrm{gcd}(a, b) = \mathrm{gcd}(b, m)$ from A
	4. Derive $\vdash \mathrm{gcd}(a, b) = r$ from 3 and 2.

$euclid'$ interleaves computation and deduction.

It is a recipe for how to prove $\mathrm{gcd}(a, b) = r$ for any give

$euclid'(12, 8) \rightsquigarrow\ \vdash \mathrm{gcd}(12, 8) = 4.$


Control flow of $\boldsymbol{euclid'}(12,8)$
$euclid'(12,8)$
Recurse
$euclid'(8,4)$
Recurse
$euclid'(4,0)$
A1
$\vdash gcd(12,8) =$
A2
$\vdash gcd(8,4) =$
A2
$\vdash gcd(4,0) =$
Proof search
Proof construc

Trusted base reduction

Q: What do we need to trust in order to believe *euclid*?

A: The logic (A1, A2, and eq. trans.) *as well as* the co
analysis and the recursive call.

I.e., we need to trust *euclid*'s control structure: the com

Q: What do we need to trust in order to believe *euclid'*

A: The logic.

We throw away the computation (proof search): it's not
trusted base.

A bug in the conventional formulation

What does it mean to say "we must trust the control st
e.g. we must trust the recursive call?

Well, suppose we made a typo:

$$\begin{aligned} euclid(a, b) = & \text{ If } b \text{ is 0 then return } a \\ & \text{ else return } euclid(\underline{a}, a \bmod b). \end{aligned}$$

So we don't swap the second argument in the recursive

Let's see what happens:

$$euclid(12, 4) \leadsto 12.$$

Oops. We get a wrong result.

There are no checks and balances!

The same bug in the CC formulation

$euclid'(a, b) =$ If b is 0 derive $\vdash \mathrm{gcd}(a, b) = a$ from A1.
If $b > 0$ then: 1. Let $m = a \bmod b$;
2. Call $euclid'(\underline{a}, m)$ to get $\vdash \mathrm{gcd}(b, m) =$
3. Derive $\vdash \mathrm{gcd}(a, b) = \mathrm{gcd}(b, m)$ from A
4. Derive $\vdash \mathrm{gcd}(a, b) = r$ from 3 and 2.

What happens now?

$euclid'(12, 4) \rightsquigarrow$ Error: unable to derive the conc
$gcd(12, 4) = 12$ from the given premises.

The error gets caught—we can't prove a wrong result ri

CC guarantee

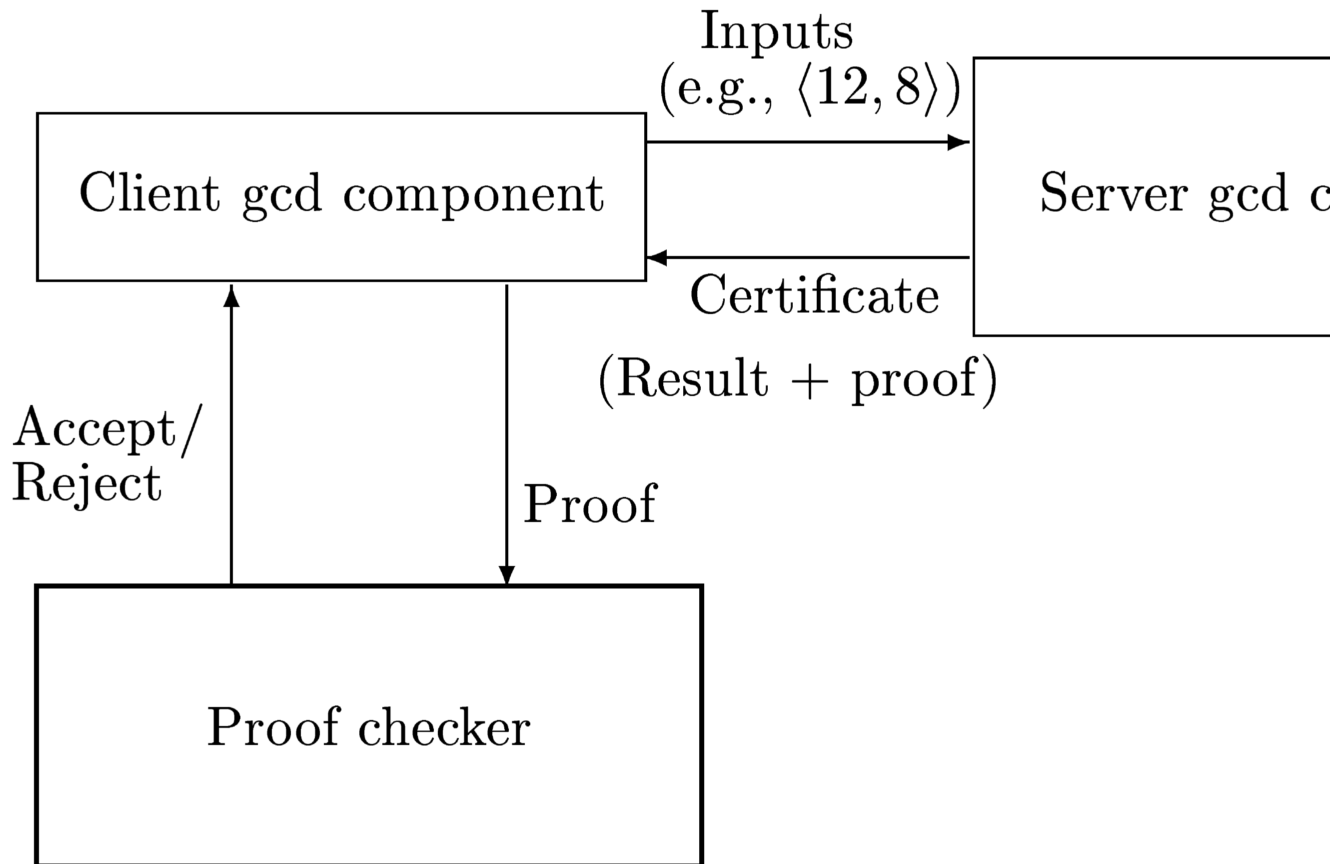
When we run a certifying algorithm P' on an input x, t
three possibilities:

1. Diverge
2. Halt in error
3. Get a theorem $\vdash \mathit{Correct}(r, x)$.

We don't know in advance which one of these will occur

But IF 3 occurs, THEN we are assured that r is correct
x—modulo our logic.

Thus deductive certification prevents incorrect results.


CC for reliable software components
Inputs
(e.g., ⟨12, 8⟩)
Client gcd component
Server gcd c
Certificate
(Result + proof)
Accept/
Reject
Proof
Proof checker

Steps in constructing a certifying algorit

1. Determine the form of the theorems to be derived.

2. Provide a logic: axioms and inference rules for estab theorems of the desired form. (Must be sound; hopefully

3. Come up with a proof-search strategy: given any in can we derive an appropriate theorem (from the logic of

4. Implement the strategy.

Language issues

How do I write a certifying algorithm?

E.g. in what language do I write $euclid'$?

Two approaches:

1. Use any language (e.g. C) but output a proof every
   in LF).
2. Use a language that:
   - Has a built-in notion of statemement and proof.
   - Integrates computation and deduction.
   - Allows for trusted "tactics".
   - Provides a truly natural deduction style.

Athena is one such language.

Other applications of CC

Sorting, searching, min-max problems

Credible compilation

Hindley-Milner type inference

Prolog engines

Related work

- Credible compilation (Pnueli, Rinard, ...)
- Program verification (Hoare, Floyd, ...)
- Result checking (Blum, ...)
- Logic programming (Kowalski, ...)
- LCF-style theorem-producing computations (Milner
- Model checking (Clarke, ...)
- Runtime assertions (Meyer, ...)