Deductive runtime certification

Konstantine Arkoudas & Martin Rinard
Result correctness

We have a program $P$ and an input $x$.

We run it: $P(x) \leadsto \text{result } r$.

How do we know that $r$ is correct?

Two prominent approaches:

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

$$\forall x . P(x) \Downarrow \Rightarrow Correct(P(x), x)$$

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

$$P(x) \leadsto \vdash Correct(r, x)$$

If we trust the axioms and inference rules, we can trust us.

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:
\[
euclid(a, b) = \begin{cases} 
    a & \text{if } b = 0 \\
    \euclid(b, a \mod b) & \text{otherwise}
\end{cases}
\]

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 prove
\[
\forall x, y . \ euclid(x, y) = \gcd(x, y)
\]

The proof relies on two results:
A1. \( \forall x . \gcd(x, 0) = x \)
A2. \( \forall x, y . y > 0 \Rightarrow \gcd(x, y) = \gcd(y, x \mod y) \)
A theorem-proving formulation

A1. \( \forall x . \gcd(x, 0) = x \)
A2. \( \forall x, y . y > 0 \Rightarrow \gcd(x, y) = \gcd(y, x \mod y) \)

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

\( euclid' \) interleaves computation and deduction.
It is a recipe for how to prove \( \gcd(a, b) = r \) for any given \( a \) and \( b \).
\( euclid'(12, 8) \rightarrow \vdash \gcd(12, 8) = 4. \)
Control flow of \( \text{euclid}'(12, 8) \)

\[
\begin{align*}
\text{euclid}'(12, 8) & \quad \vdash \gcd(12, 8) = 4 \\
\text{Recurse} & \\
\text{euclid}'(8, 4) & \quad \vdash \gcd(8, 4) = 4 \\
\text{Recurse} & \\
\text{euclid}'(4, 0) & \quad \vdash \gcd(4, 0) = 4
\end{align*}
\]

Proof search

Proof construction

Deductive runtime certification

Arkoudas & Rinard
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 control analysis and the recursive call.

I.e., we need to trust $euclid$’s control structure: the computation.

Q: What do we need to trust in order to believe $euclid$’s runtime?
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 state”
e.g. we must trust the recursive call?

Well, suppose we made a typo:

$$euclid(a, b) = \text{If } b \text{ is 0 then return } a$$
$$\text{else return } euclid(a, a \ mod \ b).$$

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

\[ euclid'(a, b) = \text{If } b \text{ is 0 derive } \vdash \text{gcd}(a, b) = a \text{ from A1.} \]
\[ \text{If } b > 0 \text{ then: 1. Let } m = a \mod b; \]
\[ 2. \text{Call } euclid'(a, m) \text{ to get } \vdash \text{gcd}(b, m) = r \]
\[ 3. \text{Derive } \vdash \text{gcd}(a, b) = \text{gcd}(b, m) \text{ from A2 and } 2. \]
\[ 4. \text{Derive } \vdash \text{gcd}(a, b) = r \text{ from 3 and 2.} \]

What happens now?

\[ euclid'(12, 4) \leadsto \text{Error: unable to derive the conclusion } \]
\[ \text{gcd}(12, 4) = 12 \text{ from the given premises.} \]

The error gets caught—we can’t prove a wrong result right?
CC guarantee

When we run a certifying algorithm $P'$ on an input $x$, there are three possibilities:

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

We don’t know in advance which one of these will occur. But if 3 occurs, THEN we are assured that $r$ is correct for $x$—modulo our logic.

Thus deductive certification prevents incorrect results.
CC for reliable software components

Client gcd component

Accept/Reject

Proof checker

Certificate (Result + proof)

Inputs (e.g., \langle 12, 8 \rangle)

Server gcd component

Deductive runtime certification Arakoudas & Rinard
Steps in constructing a certifying algorithm:

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

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

3. Come up with a proof-search strategy: given any input, how can we derive an appropriate theorem (from the logic of the two previous steps)?

4. Implement the strategy.
Language issues

How do I write a certifying algorithm?

E.g. in what language do I write \textit{euclid}'?

Two approaches:

1. Use any language (e.g. C) but output a proof every \textit{in LF}).

2. Use a language that:
   - Has a built-in notion of statement 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, ...)

Deductive runtime certification

Arkoudas & Rinard