

Stop testing, start proving

Valentin Bergeron

ScalaDays 2026

Hey Claude, can you implement partial refunds please ?
Make no mistakes !!

Hey Claude, can you implement partial refunds please ?
Make no mistakes !!

Result: **180k€ loss in one weekend**

Before

```
case class Order(id: OrderId, paid: Money, refund: Option[Refund])

def canRefund(o: Order, amount: Money): Boolean =
  o.refund.isEmpty || o.refund.head.amount + amount ≤ o.paid
```

After

```
case class Order(id: OrderId, paid: Money, refunds: List[Refund])

def canRefund(o: Order, amount: Money): Boolean =
  o.refunds.isEmpty || o.refunds.head.amount + amount ≤ o.paid
```

Vacuously True Driven Development

The test

```
test("a refund never exceeds what was paid") {  
  val order    = Fixtures.paidOrder(amount = 100)  
  val refunds = db.refunds.findByOrder(order.id)  
  refunds.foreach { r =>  
    assert(r.amount ≤ order.paid)  
  }  
}
```

Meanwhile, in `Fixtures.scala`

```
object Fixtures:  
  def paidOrder(amount: Money): Order =  
    db.orders.insert(Order(OrderId.random(), paid = amount, refunds = Nil))
```

Everything was green

Every test was green

Every pull request was reviewed and approved

The code did exactly what it said

Nobody wrote down **what must be true.**



It is all about proofs and trust.

Descartes as a shadow FP maximalist ?



Empiricists

Knowledge comes from **experience**

Locke, Berkeley, Hume

VS



Rationalists

Knowledge comes from **reason**

Descartes, Spinoza, Leibniz

Descartes as a shadow FP maximalist ?



Testers

Trust the CI bro

Locke, Berkeley, Hume

VS



Typers

If it compiles it works

Descartes, Spinoza, Leibniz

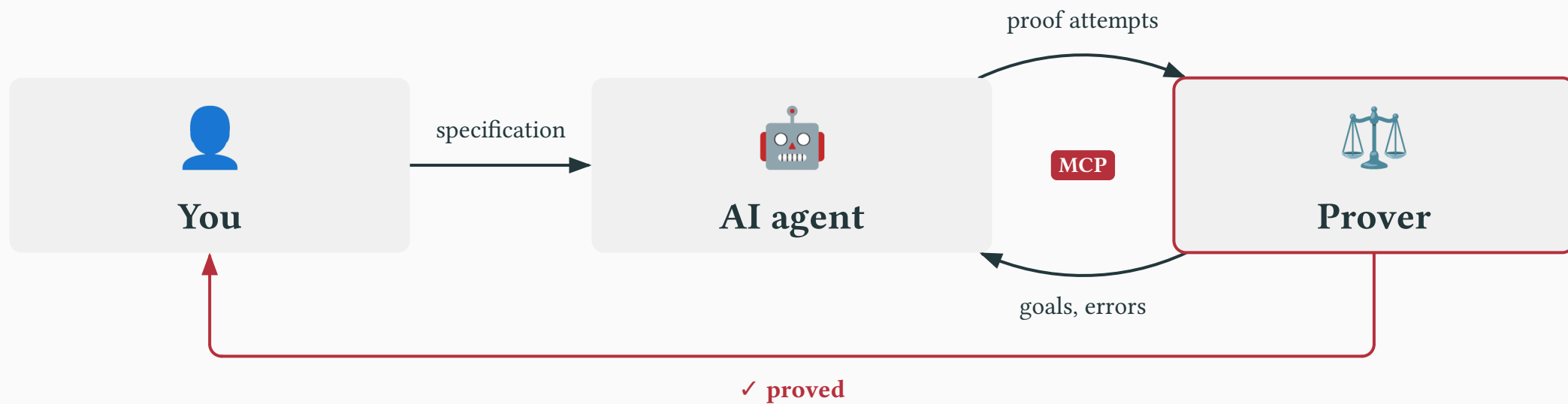
Curry–Howard: proofs are programs

Theorem (Curry 1934–1958, Howard 1969).

Proofs in intuitionistic logic are exactly the programs of the typed λ -calculus: a proposition is provable if and only if the corresponding type has a program.

Checking a proof = checking a program's type.

A theorem prover **is** a type checker
for a language whose type system is powerful enough
to express **arbitrary statements about its terms**.





Inria, 1984 · **Coq** → **Rocq** in 2025

A theorem and its proof

```
Theorem never_refund_more_than_paid : forall (events : list event),  
  refunded (replay events) ≤ paid (replay events).
```

A theorem and its proof

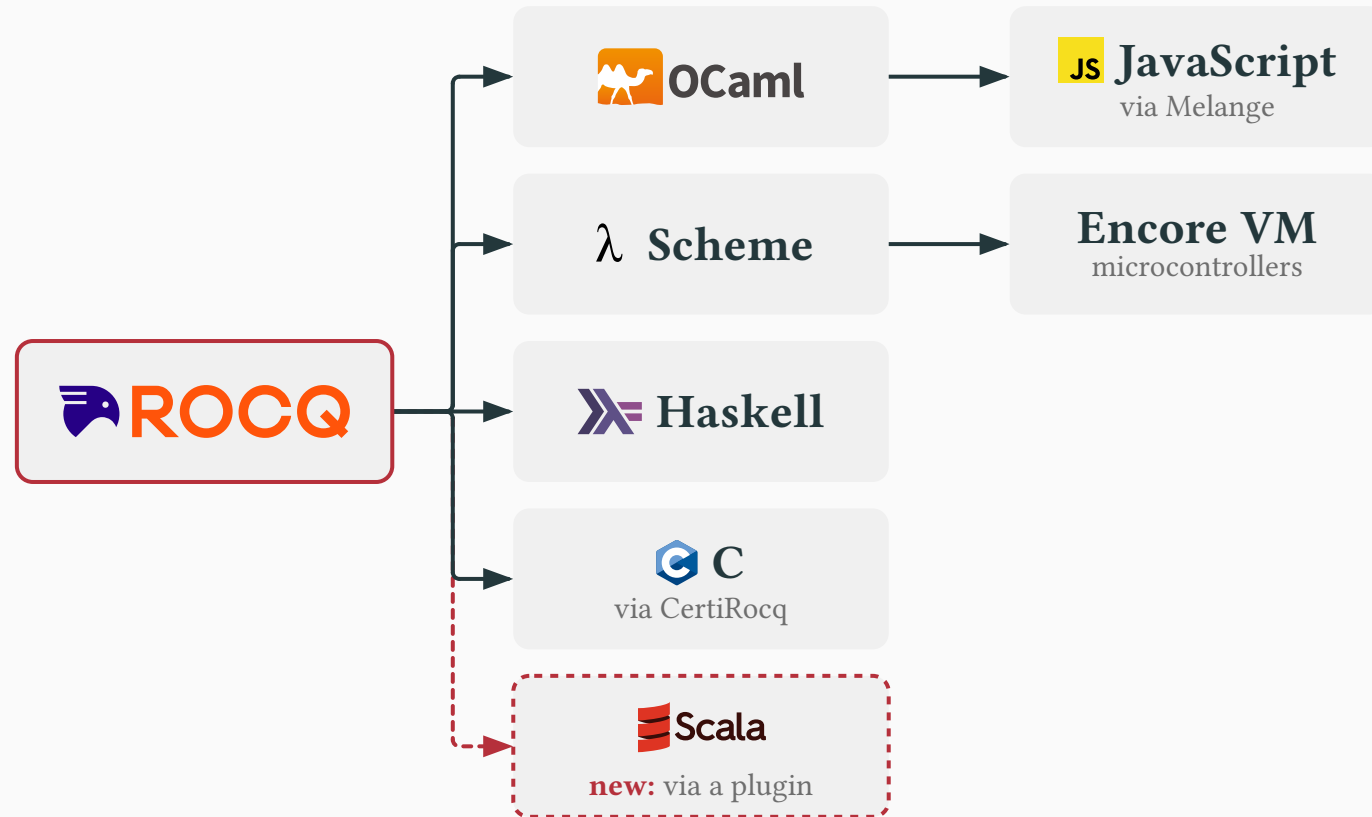
```
Theorem never_refund_more_than_paid : forall (events : list event),  
  refunded (replay events) ≤ paid (replay events).
```

Proof.

```
  intro events. unfold replay.  
  assert (H : refunded empty ≤ paid empty) by (cbn; lia).  
  revert H. generalize empty as o.  
  induction events as [| e es IH]; intros o H; cbn; [exact H|].  
  apply IH, handle_keeps_refunds, H.
```

Qed.

Extraction: from proof to production



Be the one who writes down **what must be true.**



<https://vbergeron.github.io/rocq-solid/>