Static and User-Extensible Proof Checking

Antonis Stampoulis Zhong Shao
Yale University

POPL 2012

Proof assistants are becoming popular in our community

— CompCert [Leroy et al.]

— seL4 [Klein et al.]

— Four-color theorem [Gonthier et al.]

… but they're still hard to use

— 1 to 1.5 weeks per paper proof page

— 4 pages of formal proof per 1 page of paper proof

[Asperti and Coen '10]

Formal proofs

—communicated to a **fixed** proof checker

—must spell out all details

—use domain-specific lemmas

≠

Informal proofs

—communicated to a person

—rely on **domain-specific intuition**

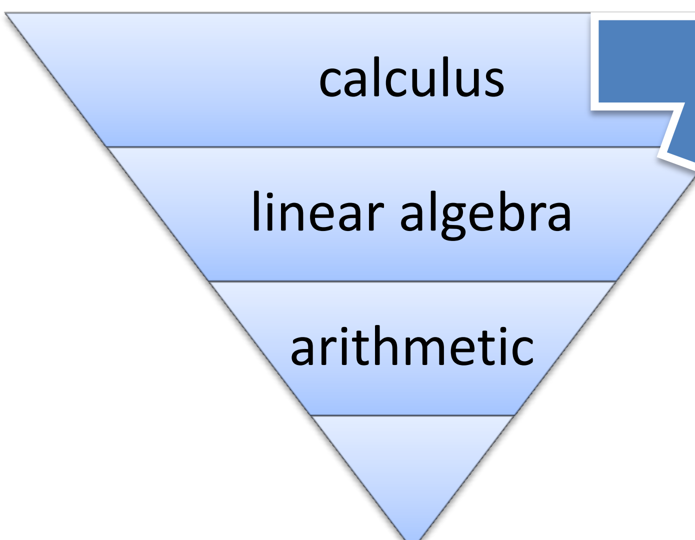
—use “obviously”

Formal proofs

—communicated to a **fixed** proof checker

—must spell out all details

—use domain-specific lemmas



≠

Informal proofs

–communicated to a person

—rely on **domain-specific intuition**

—use “obviously”

We need tactics to omit details

— procedures that produce proofs
— domain-specific tactics good in large developments
— but difficult to write!

We need tactics to omit details

—procedures that pro

—domain-specific tact
developments

—but difficult to write!

- untyped
- proofs within tactics can be wrong!

Proof assistants are hard to use because

1. cannot extend proof checker → lots of details

2. no checking for tactics → lots of potential errors

These are architectural issues

Our contribution:
A new architecture for proof assistants

1. cannot extend proof checker → lots of details
2. no checking for tactics → lots of potential errors

Our contribution:
A new architecture for proof assistants

1. extensible proof checker → omit lots of details

1. ~~cannot extend proof checker → lots of details~~

2. no checking for tactics → lots of potential errors

full programming model
soundness guaranteed

contribution:
ture for proof assistants

1. extensible proof checker → omit lots of details

1. ~~cannot extend proof checker → lots of details~~

2. no checking for tactics → lots of potential errors

Our contribution:
A new architecture for proof assistants

1. extensible proof checker → omit lots of details

1. ~~cannot extend proof checker → lots of details~~

2. ~~no checking for tactics → lots of potential errors~~

2. extensible checking for tactics → lots of errors avoided

Our contribution:
A new architecture for proof assistants

1. extensible proof checker → omit lots of details

1. ~~cannot extend proof checker → lots of details~~

2. ~~no checking for tactics → lots of potential errors~~

2. extensible checking for tactics → lots of errors avoided

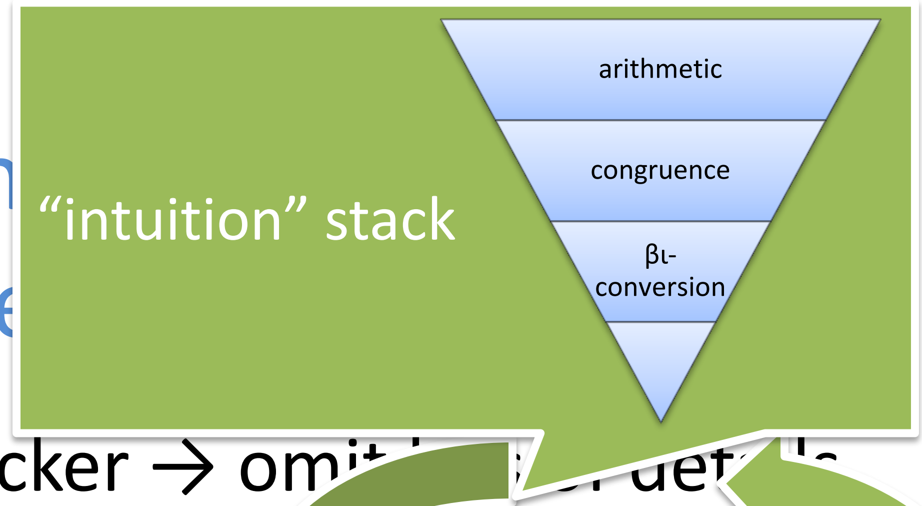
static checking of contained proofs

Our contribution:
A new architecture for proof assistants

1. extensible proof checker → omit lots of details
1. ~~cannot extend proof checker → lots of details~~

2. ~~no checking for tactics → lots of potential errors~~
2. extensible checking for tactics → lots of errors avoided

Our con…
A new architecture…



1. extensible proof checker → omit lots of details
1. ~~cannot extend proof checker → lots of details~~

2. ~~no checking for tactics → lots of potential errors~~
2. extensible checking for tactics → lots of errors avoided

Our contribution:
A new architecture for proof assistants

1. extensible proof checker → omit lots of details
1. ~~cannot extend proof checker → lots of details~~

2. ~~no checking for tactics → lots of potential errors~~
2. extensible checking for tactics → lots of errors avoided

More specifically:

- **a new language design**
- **a new implementation**
- **and a new metatheory**

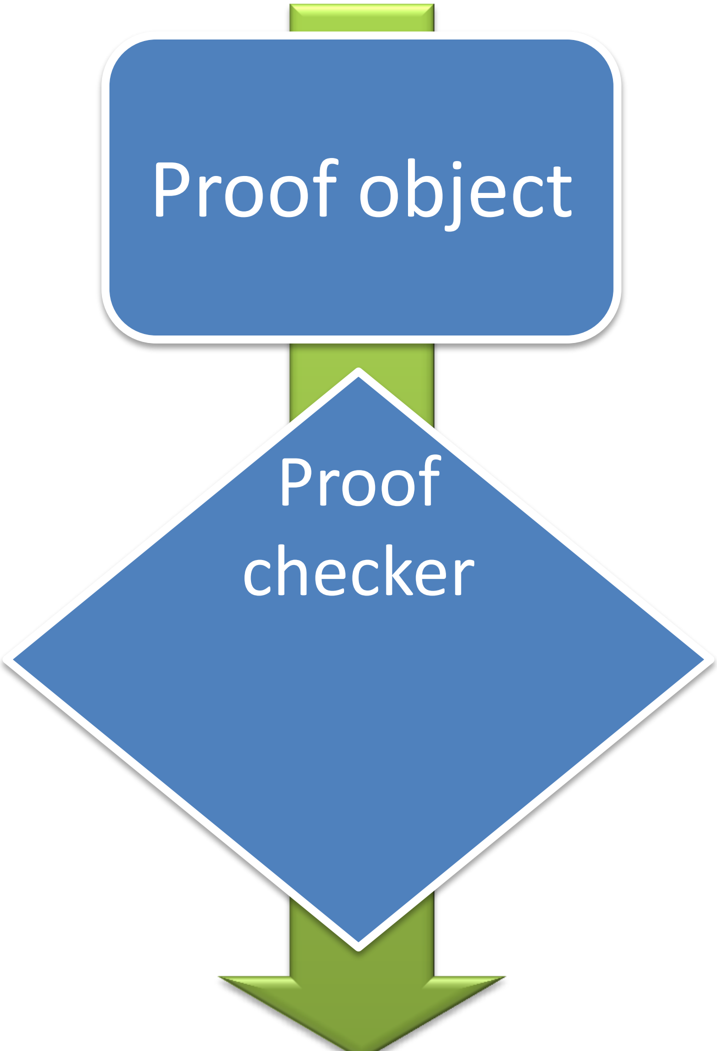
based on VeriML [ICFP'10]

Architecture of proof assistants

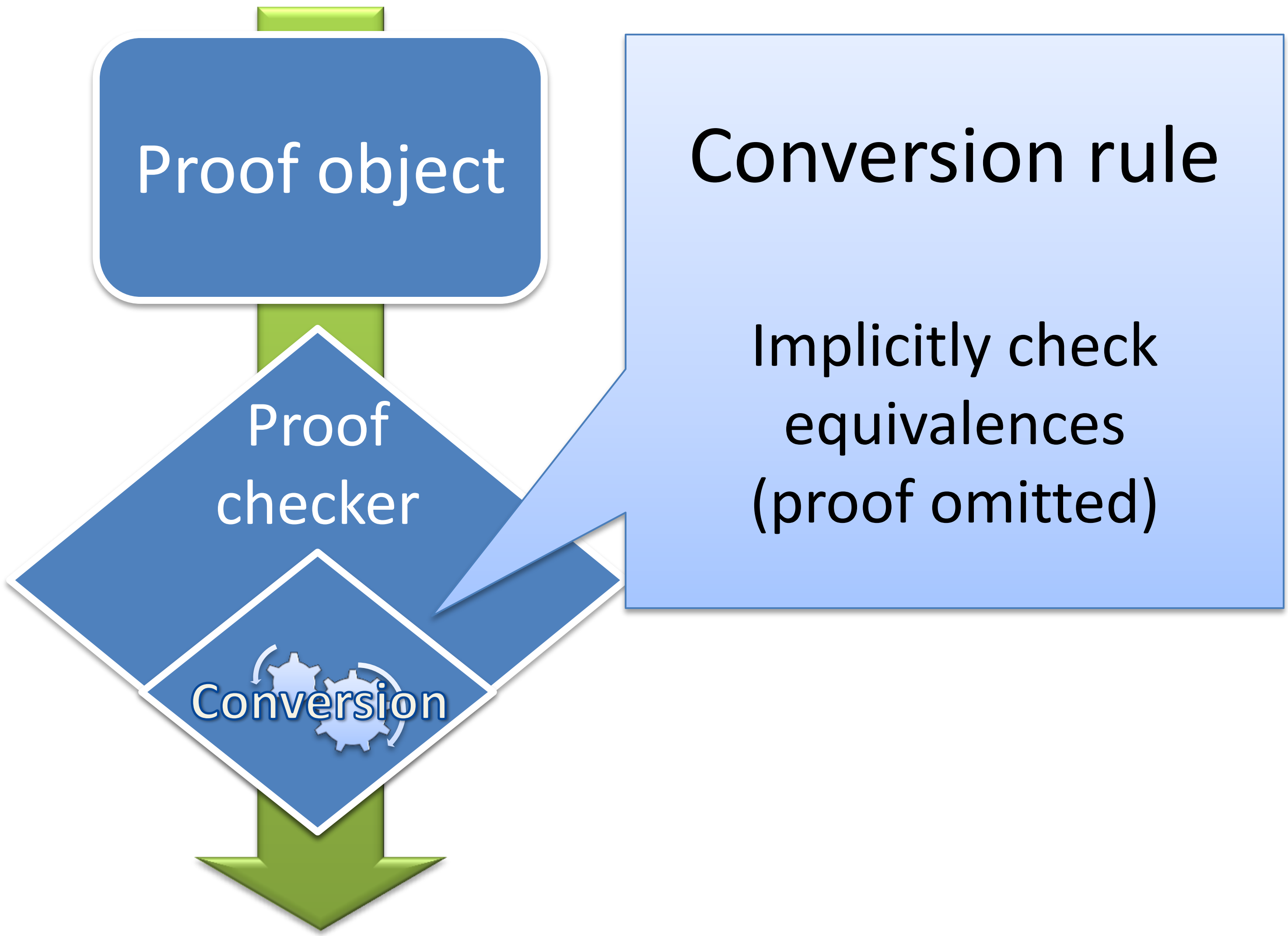
Architecture of proof assistants: main notions

Proof object	Derivation in a logic
Proof checker	Checks proof objects
Tactic	Function producing proof objects
Proof script	Combination of tactics; program producing a proof object

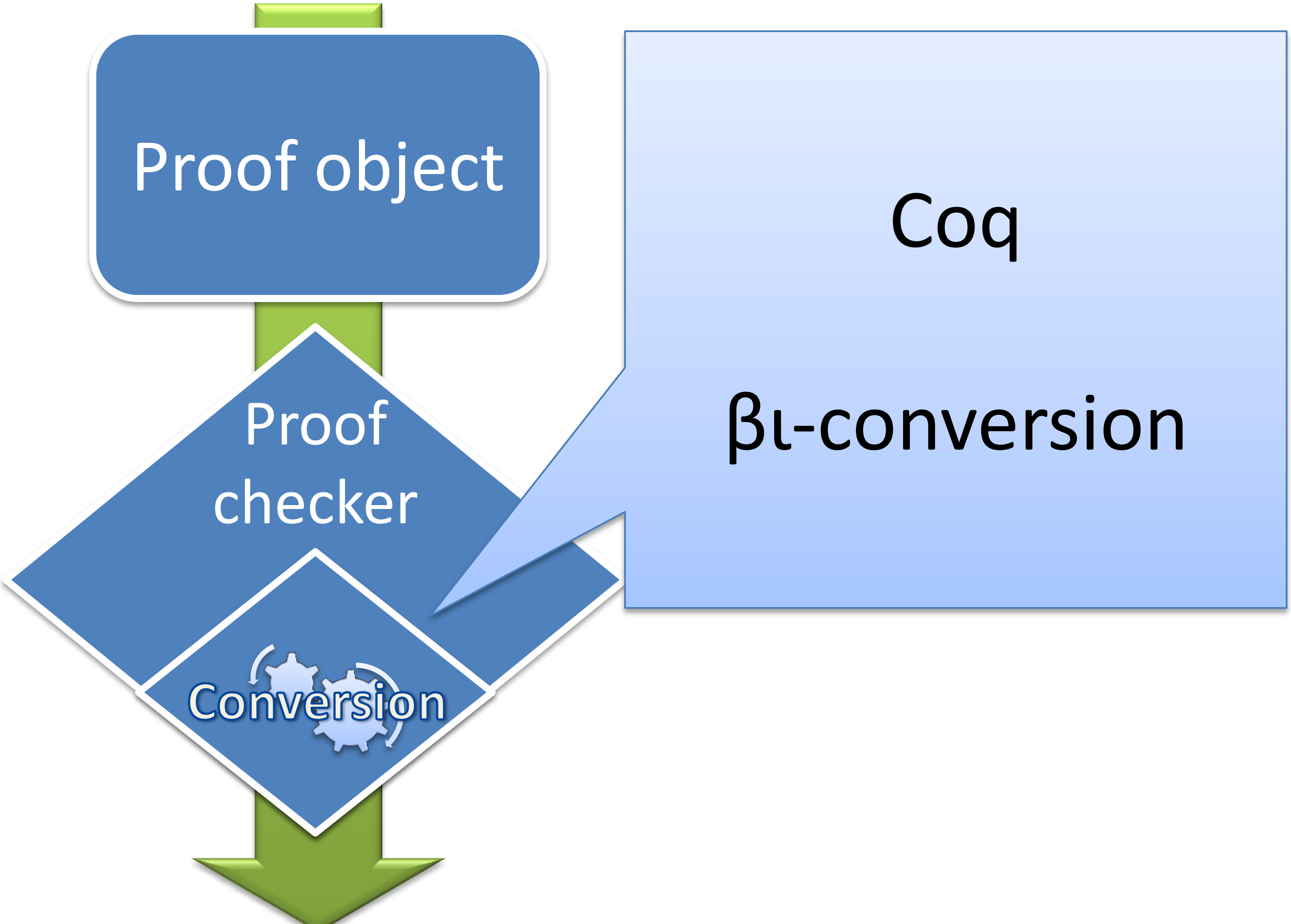
Architecture of proof assistants: Checking proof objects



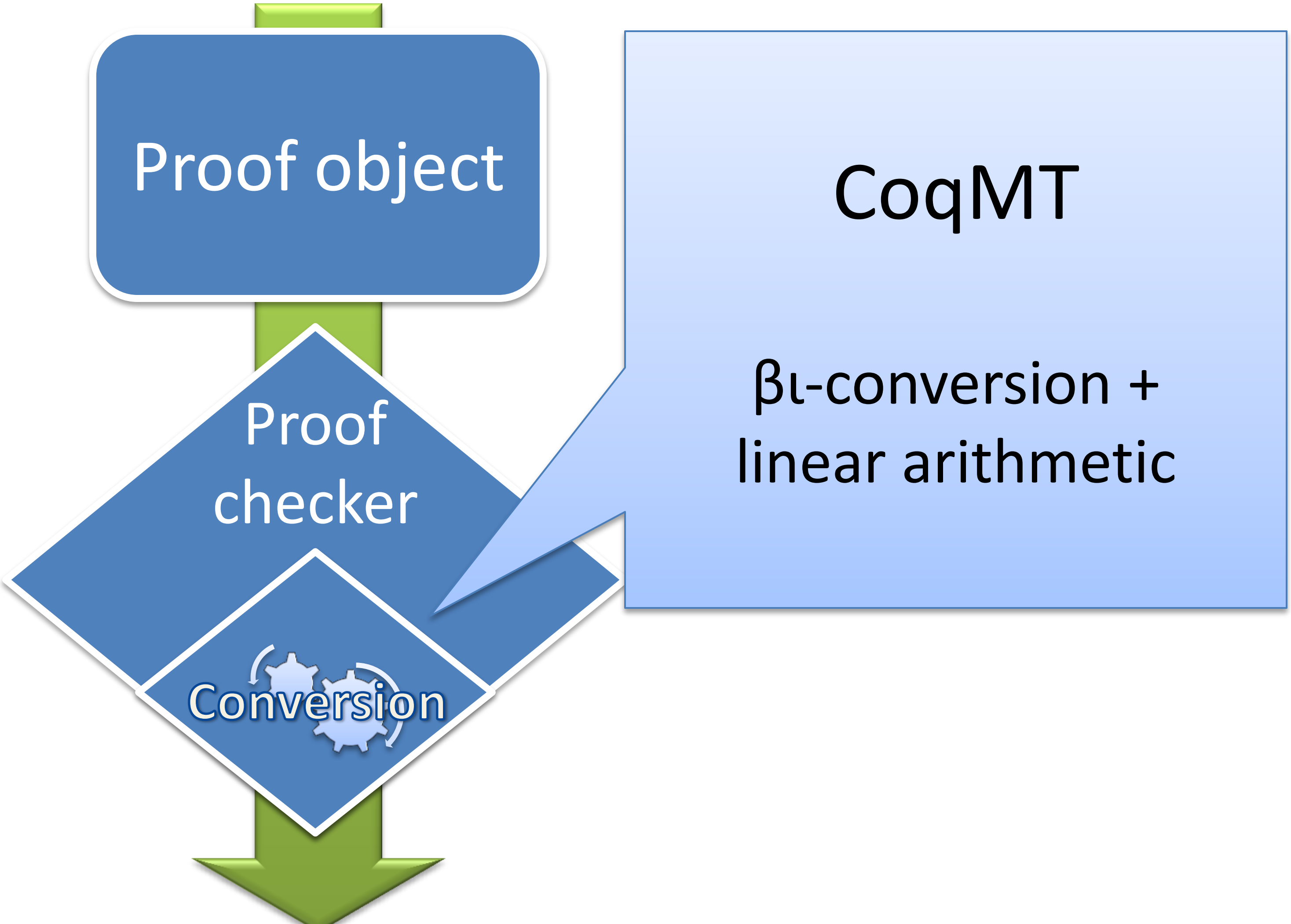
Architecture of proof assistants: Checking proof objects



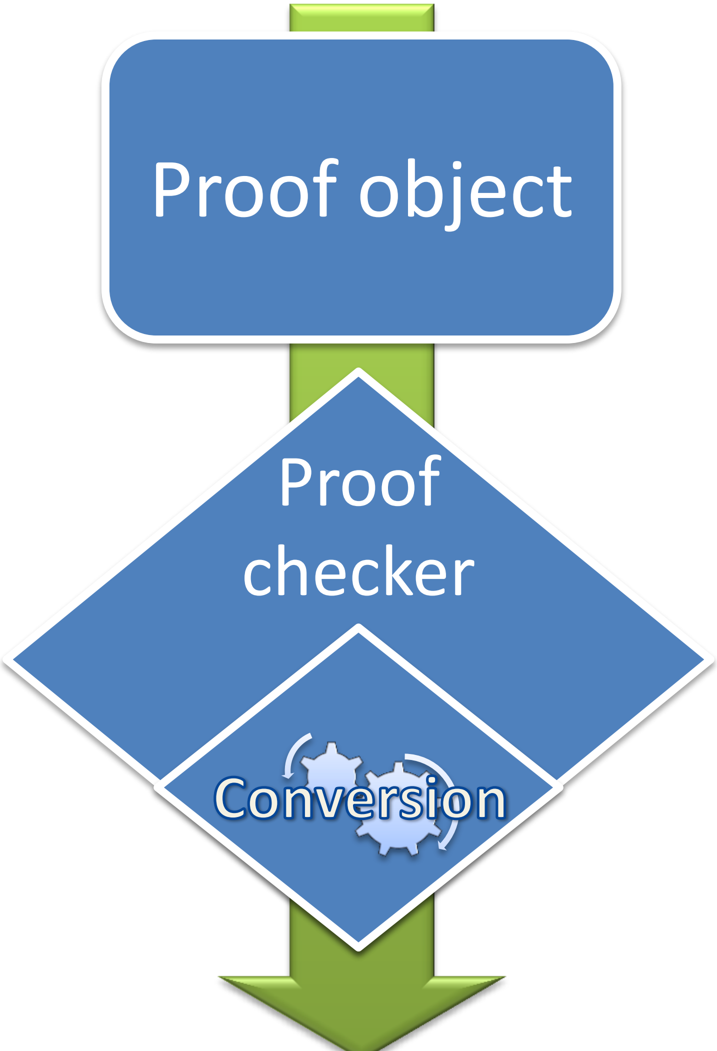
Architecture of proof assistants: Checking proof objects



Architecture of proof assistants: Checking proof objects



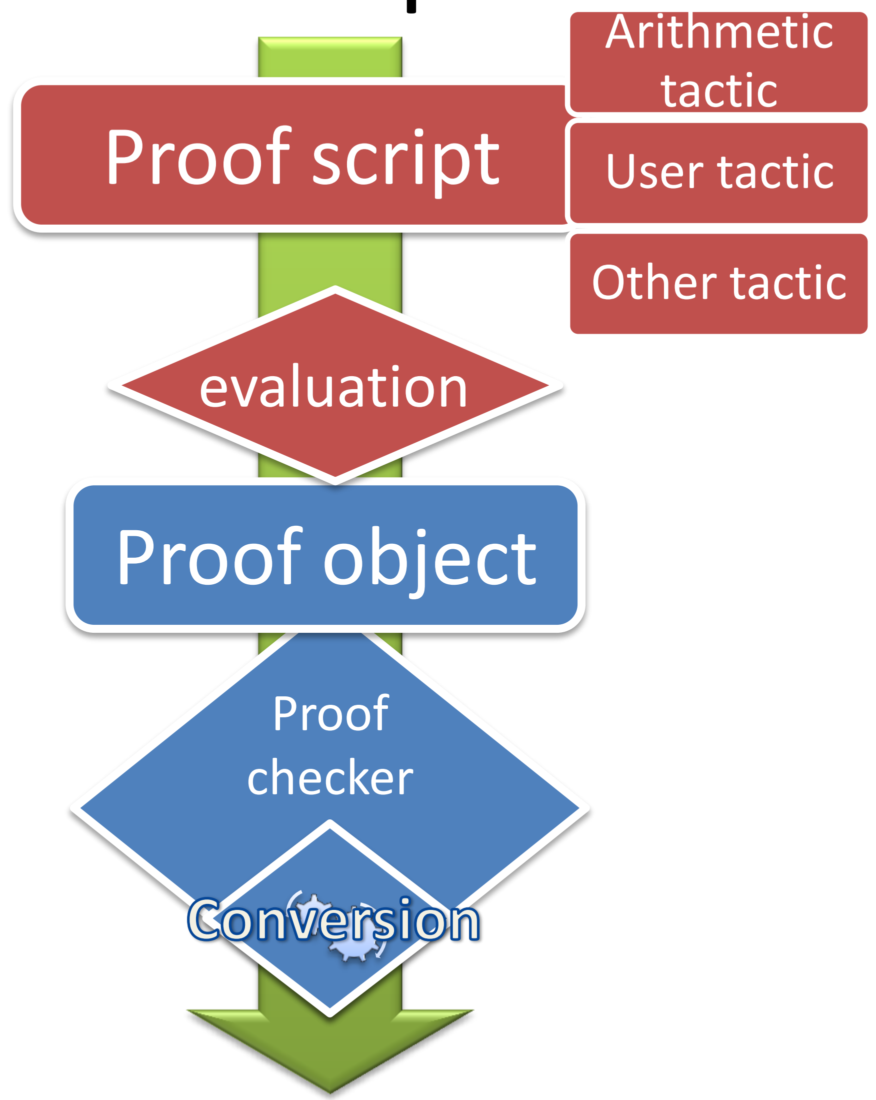
Architecture of proof assistants: Checking proof objects



- rich static information
- (de)composable
- checking not extensible

Architecture of proof assistants: Validating proof scripts

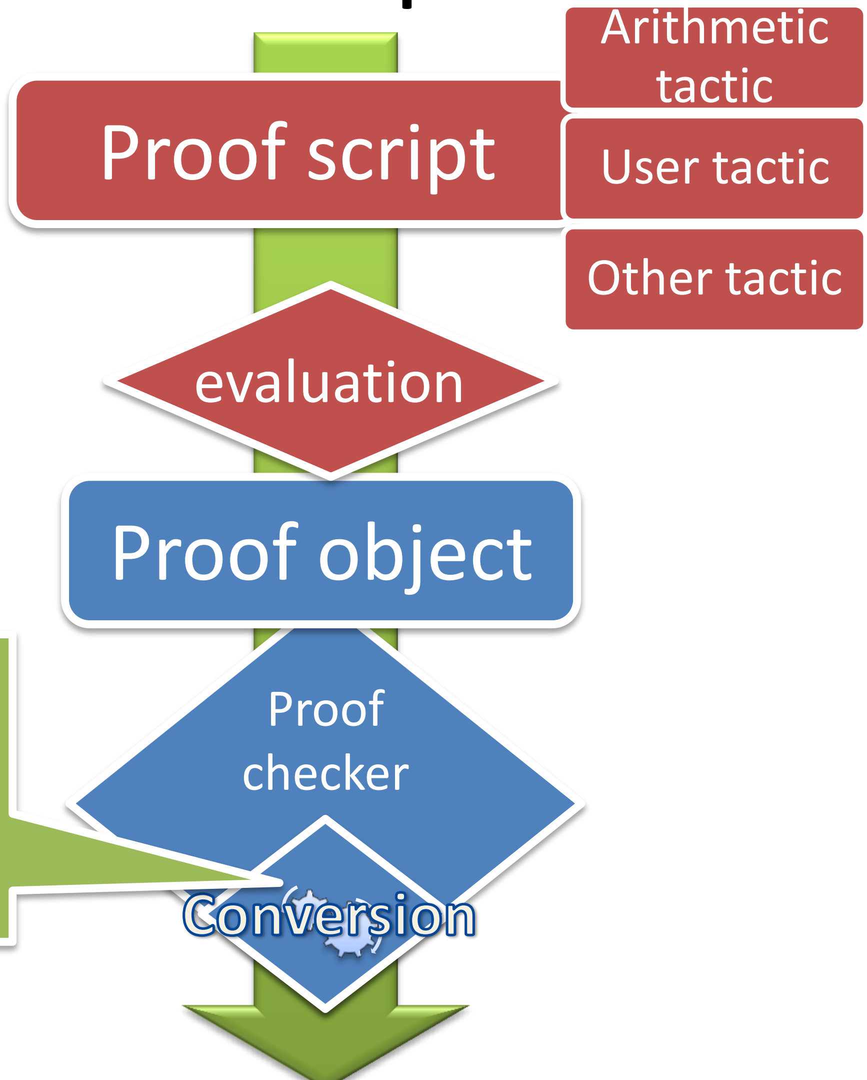
- extensible through tactics
- rich programming model
- no static information
- not (de)composable
- hidden proof state



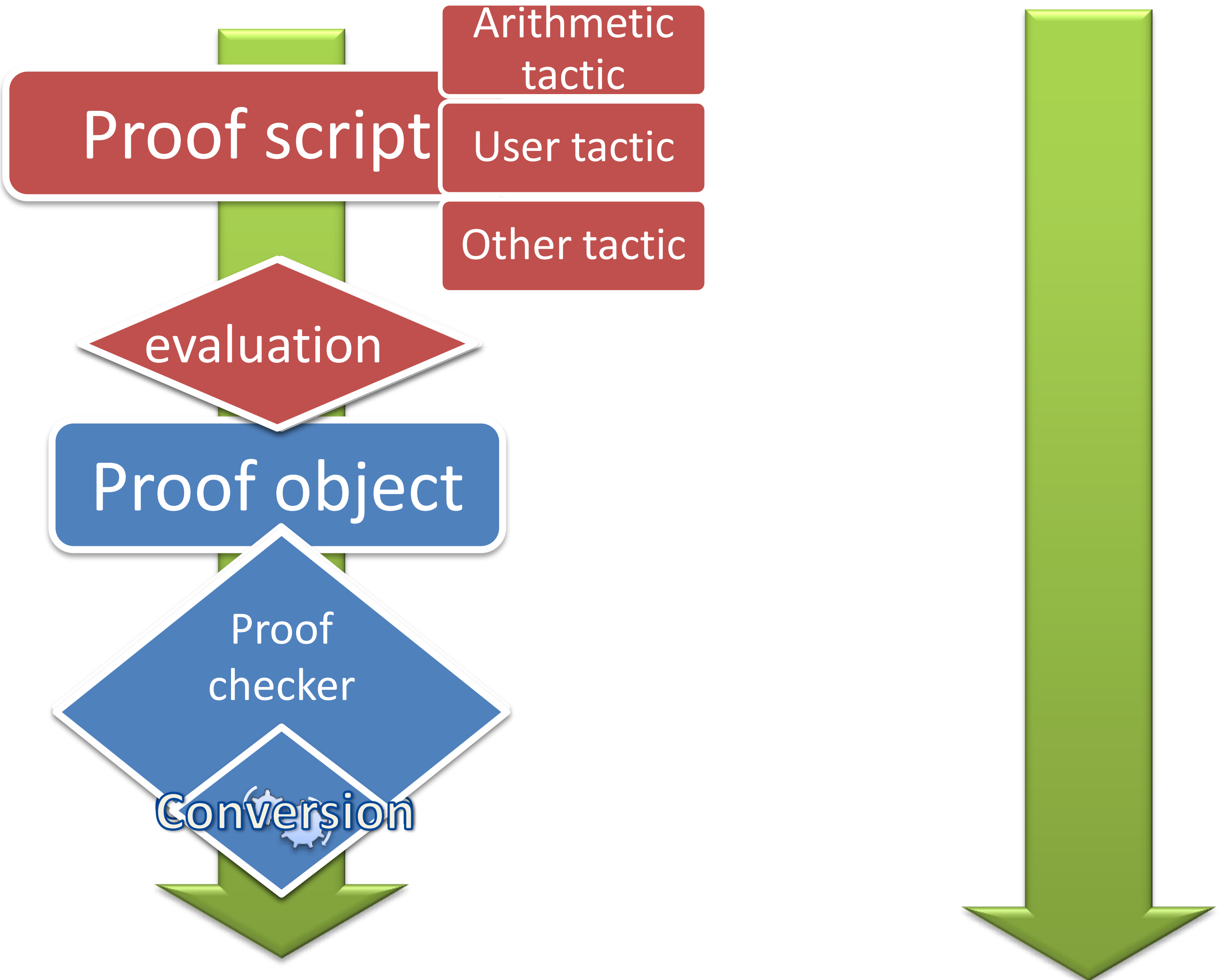
Architecture of proof assistants: Validating proof scripts

- extensible through tactics
- rich programming model
- no static information
- not (de)composable
- hidden proof state

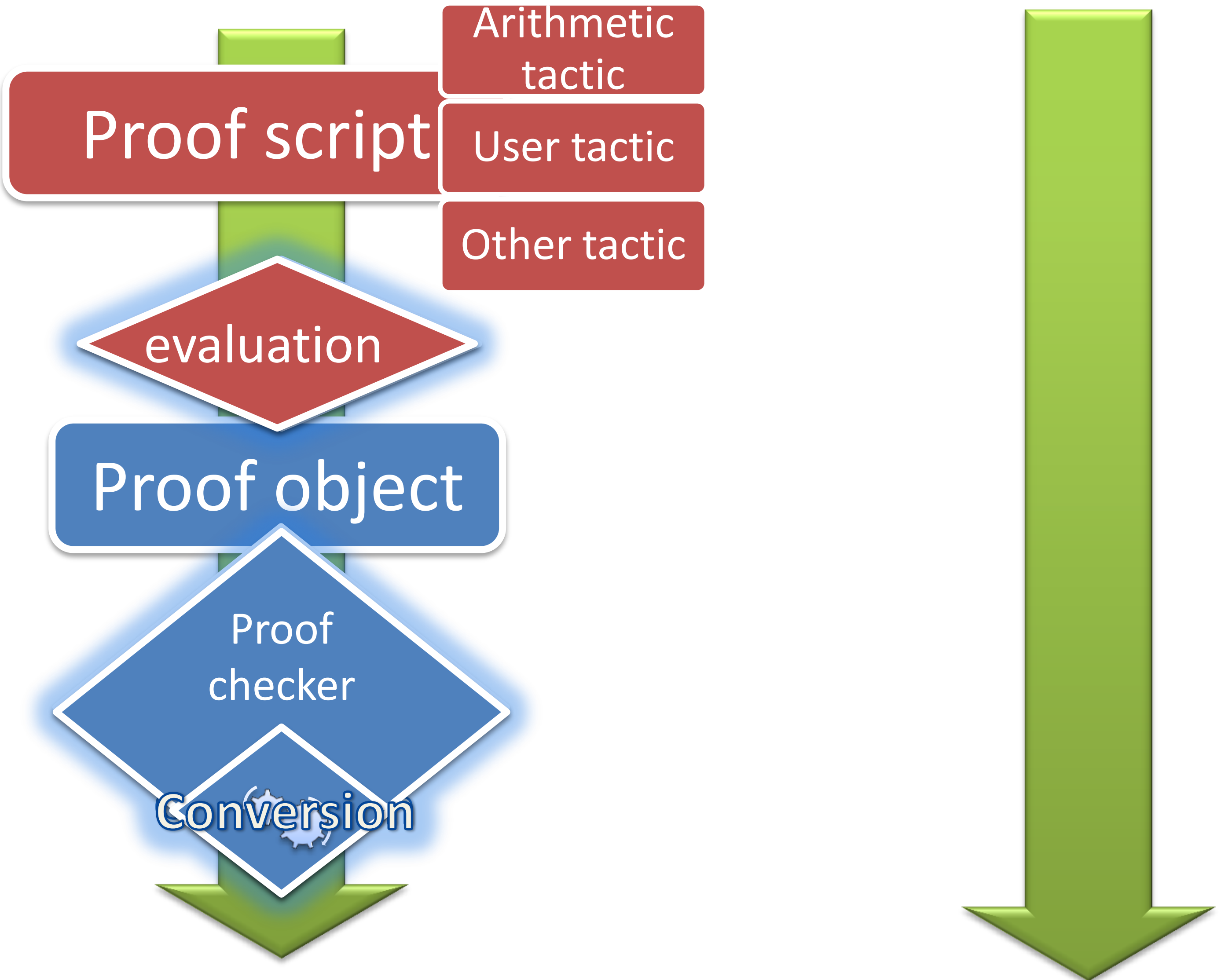
use conversion for more robust scripts



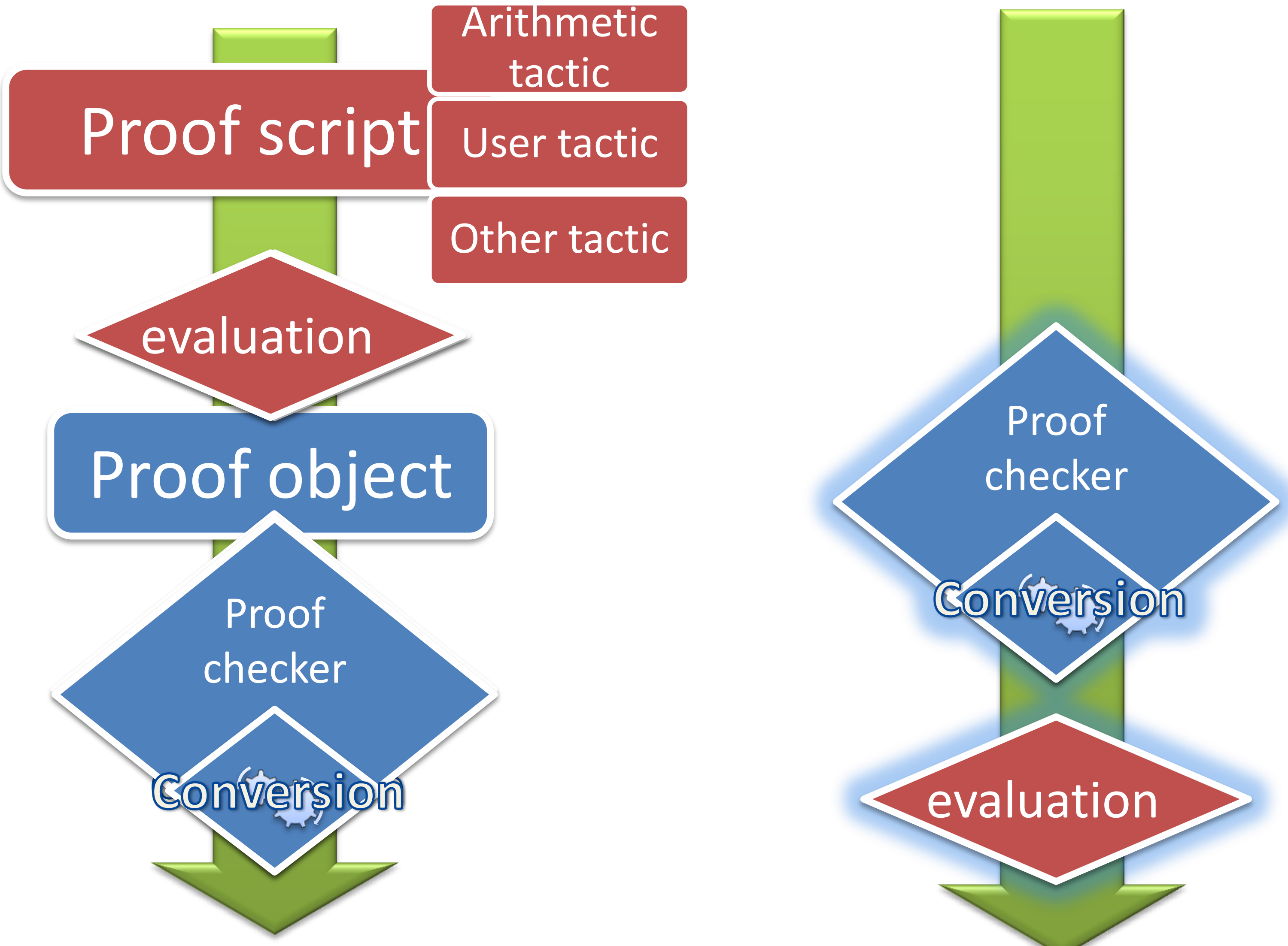
Moving to typed proof scripts



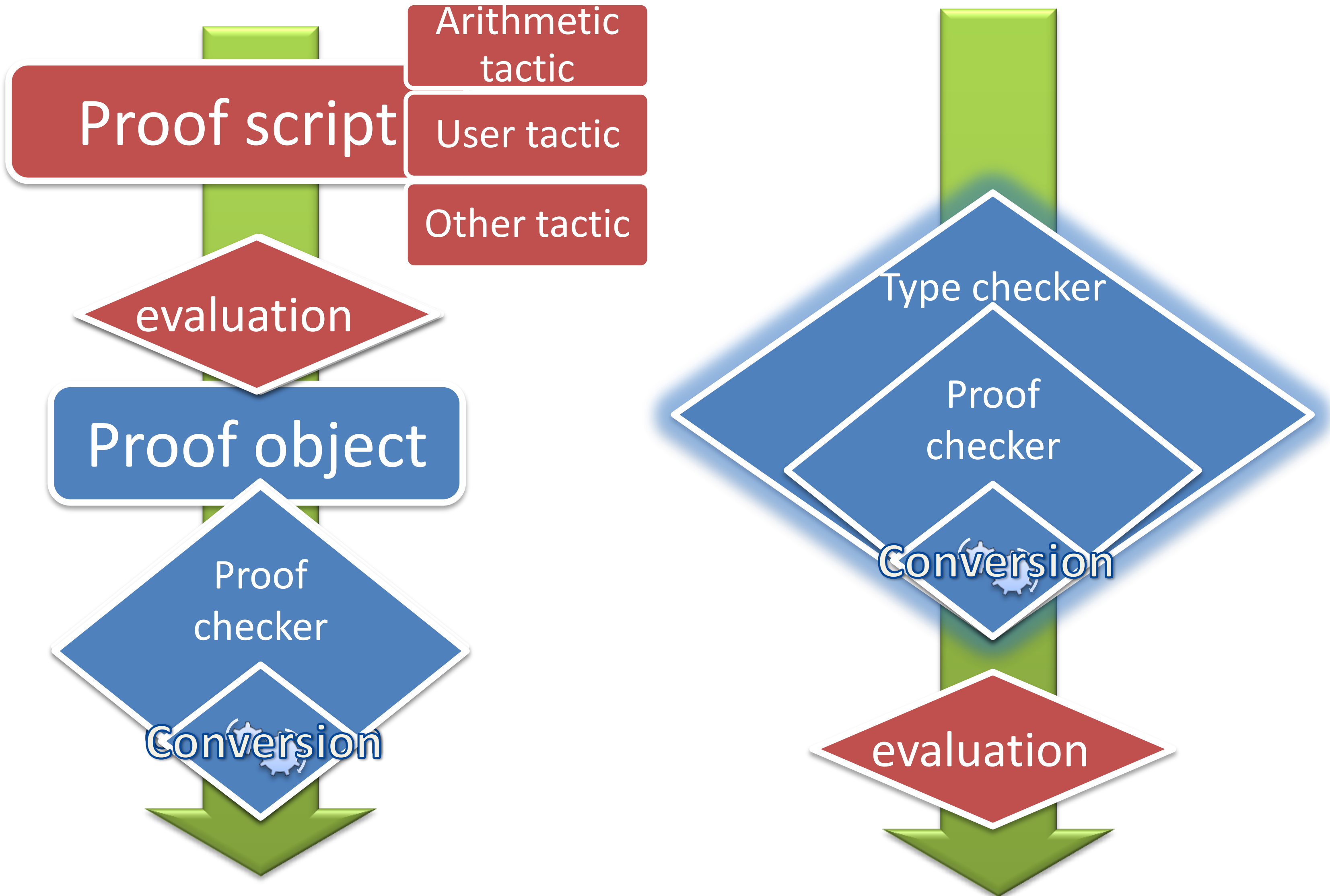
Moving to typed proof scripts



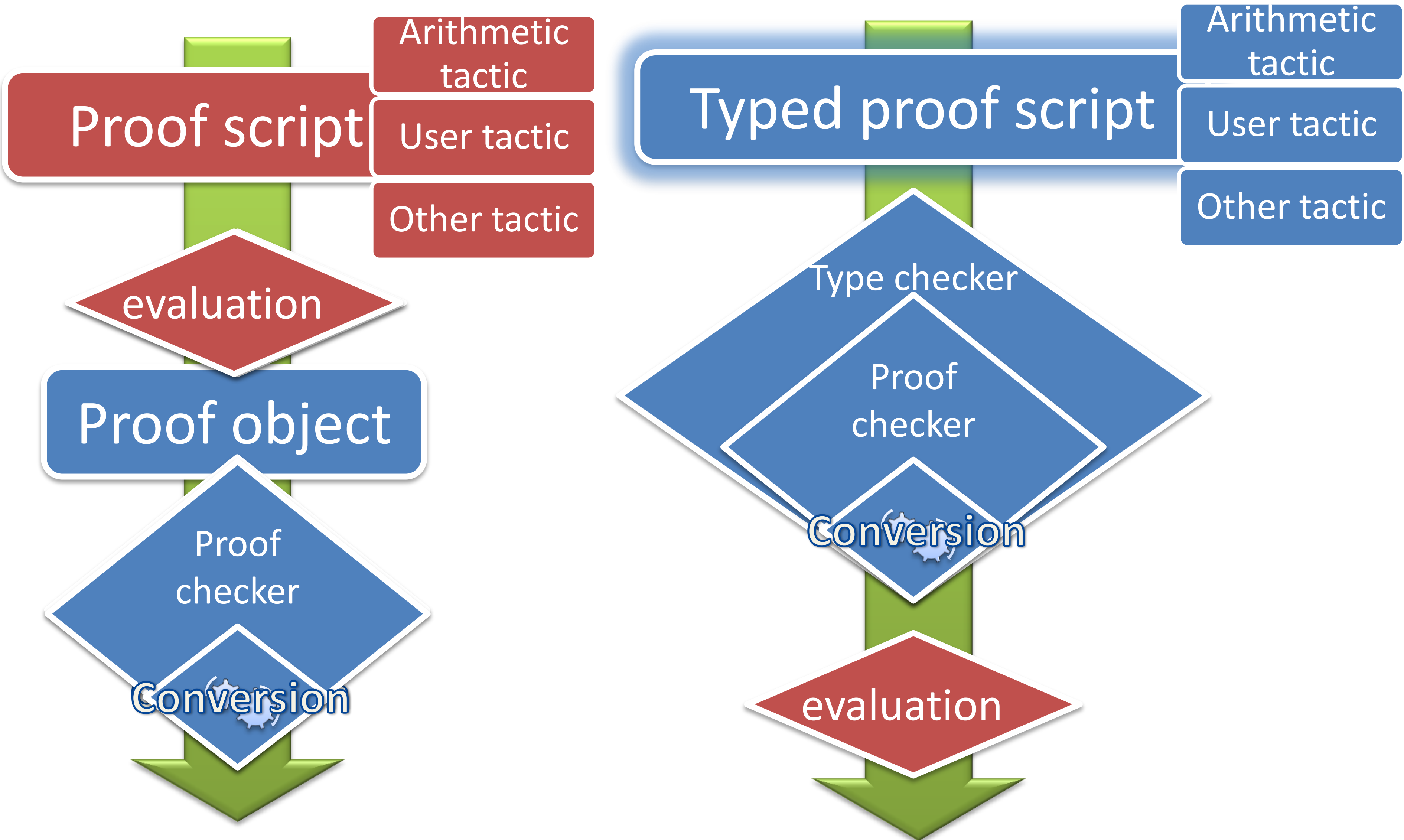
Moving to typed proof scripts



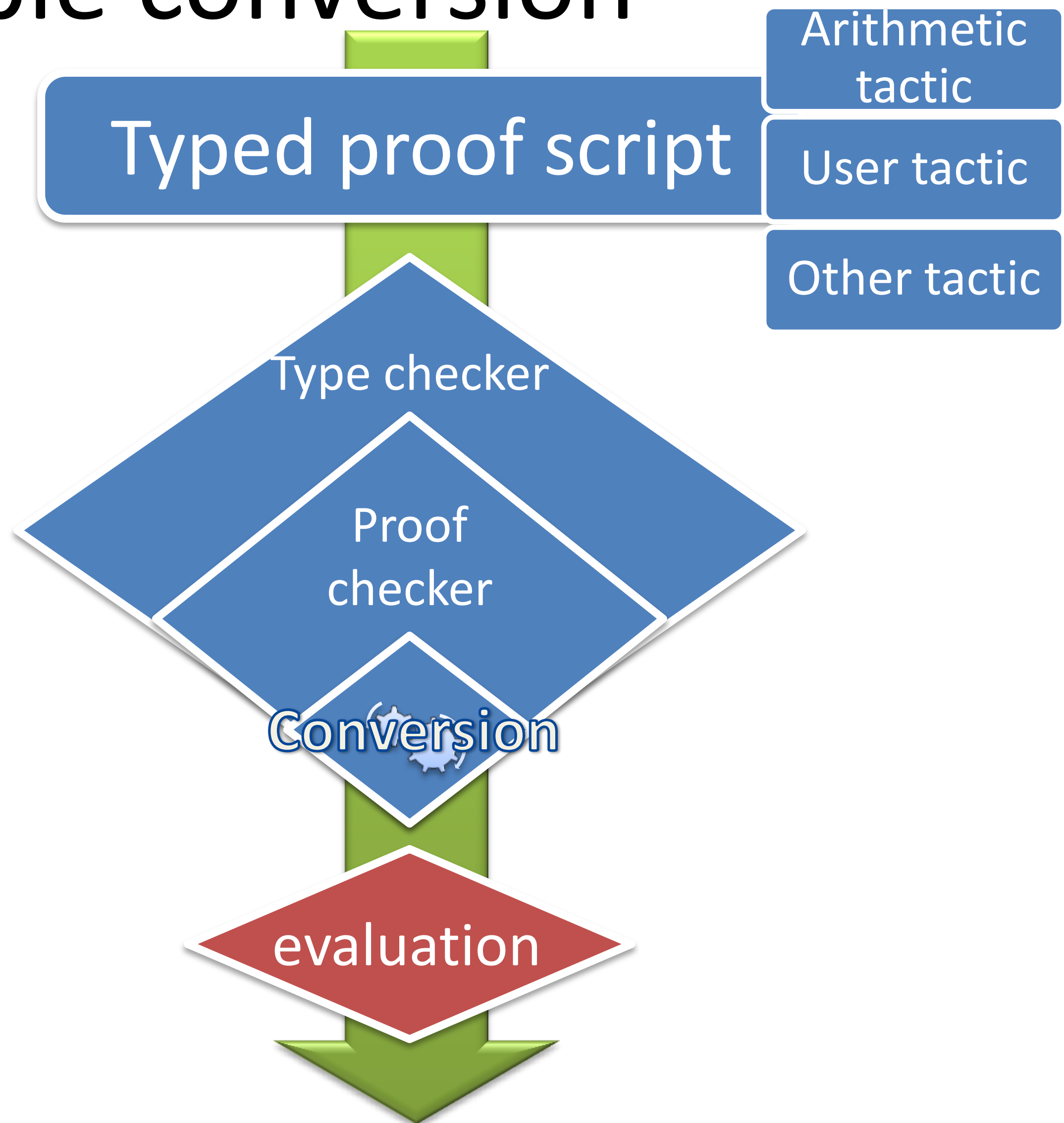
Moving to typed proof scripts



Moving to typed proof scripts

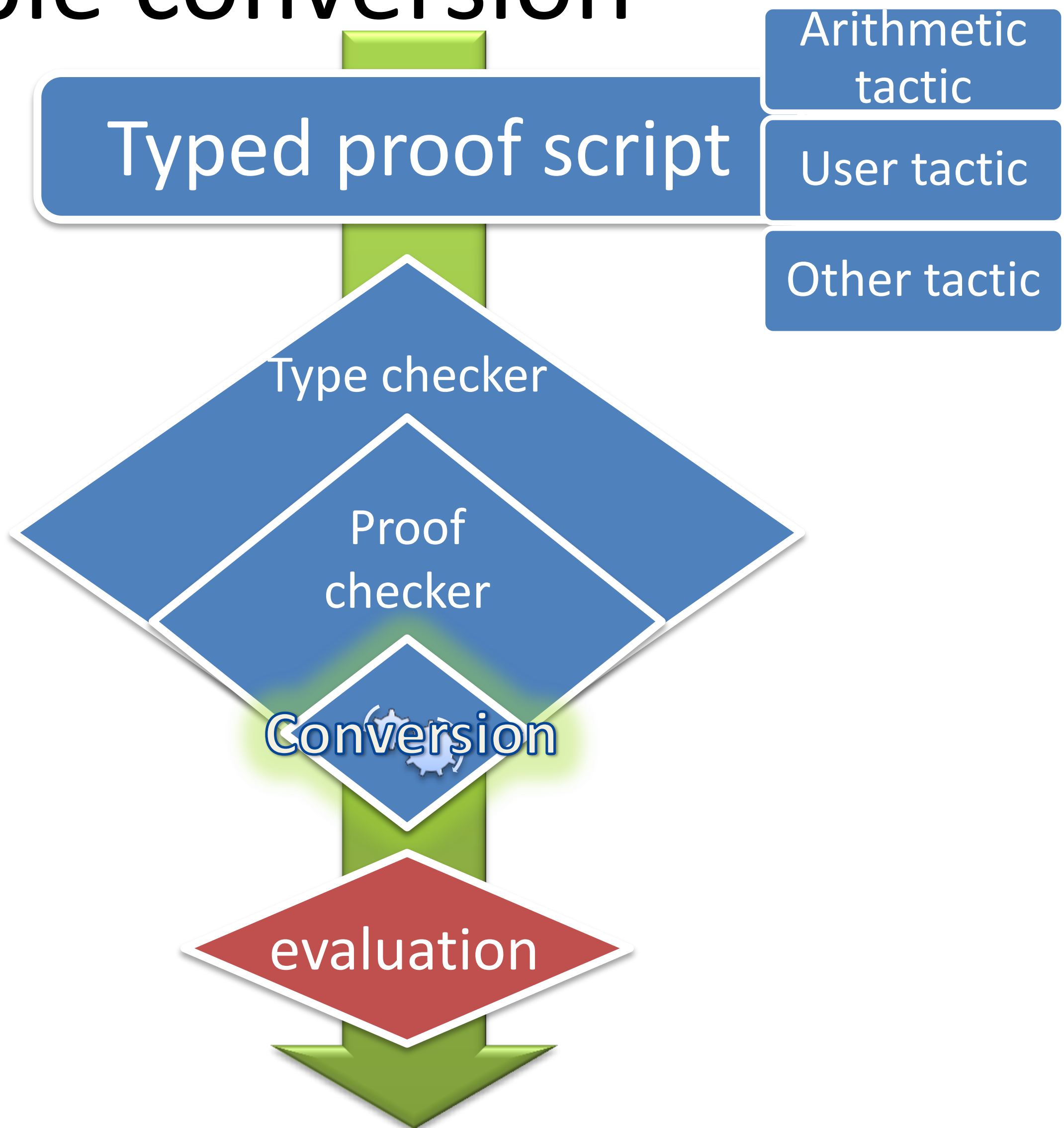


Moving to typed proof scripts + extensible conversion



Moving to typed proof scripts + extensible conversion

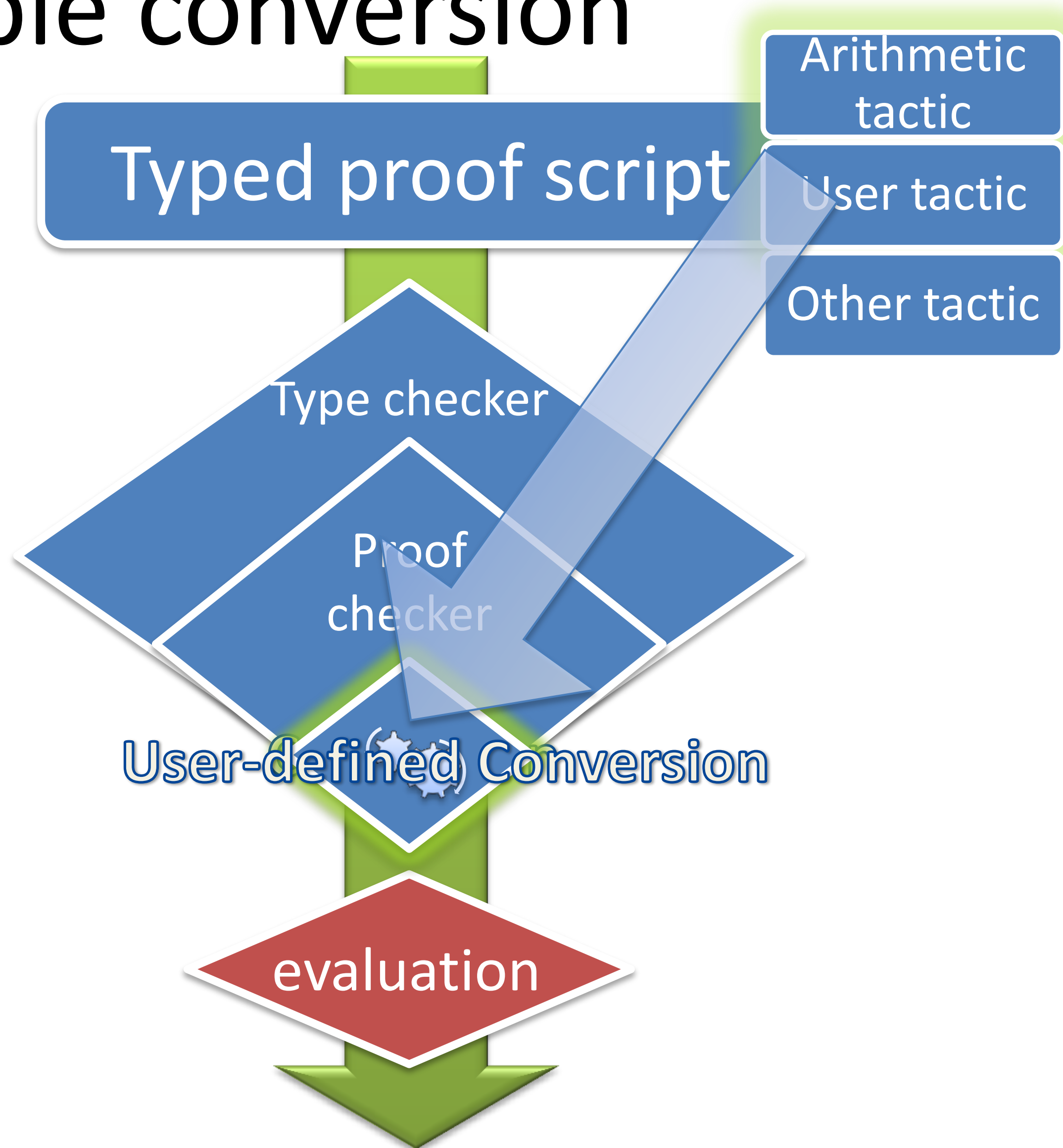
Key insight:
conversion is just a hardcoded trusted tactic



Moving to typed proof scripts + extensible conversion

Key insight:
conversion is just a hardcoded trusted tactic

but we can trust other tactics too if they have the right type

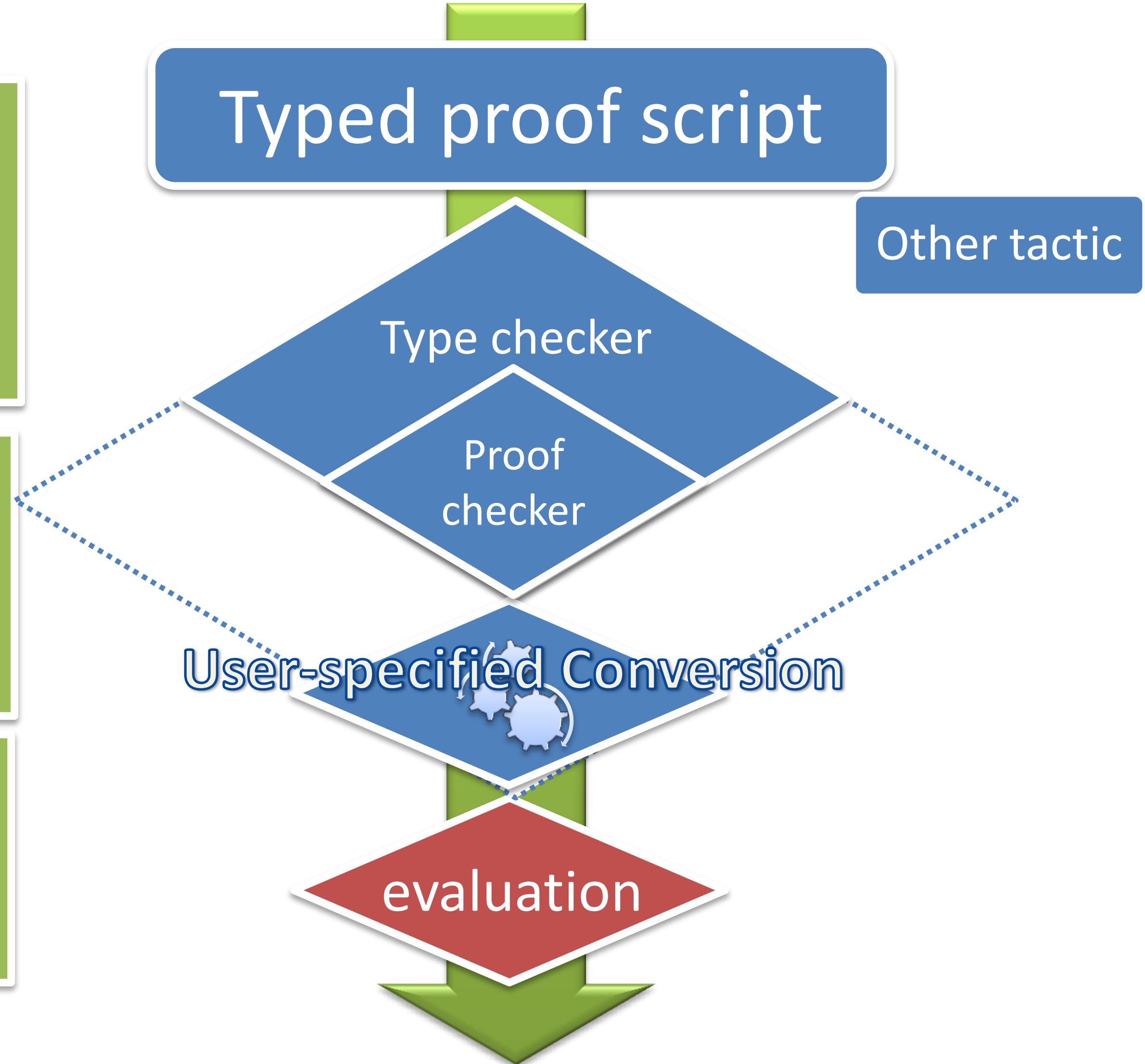


Moving to typed proof scripts + extensible conversion

Key insight:
conversion is just a hardcoded trusted tactic

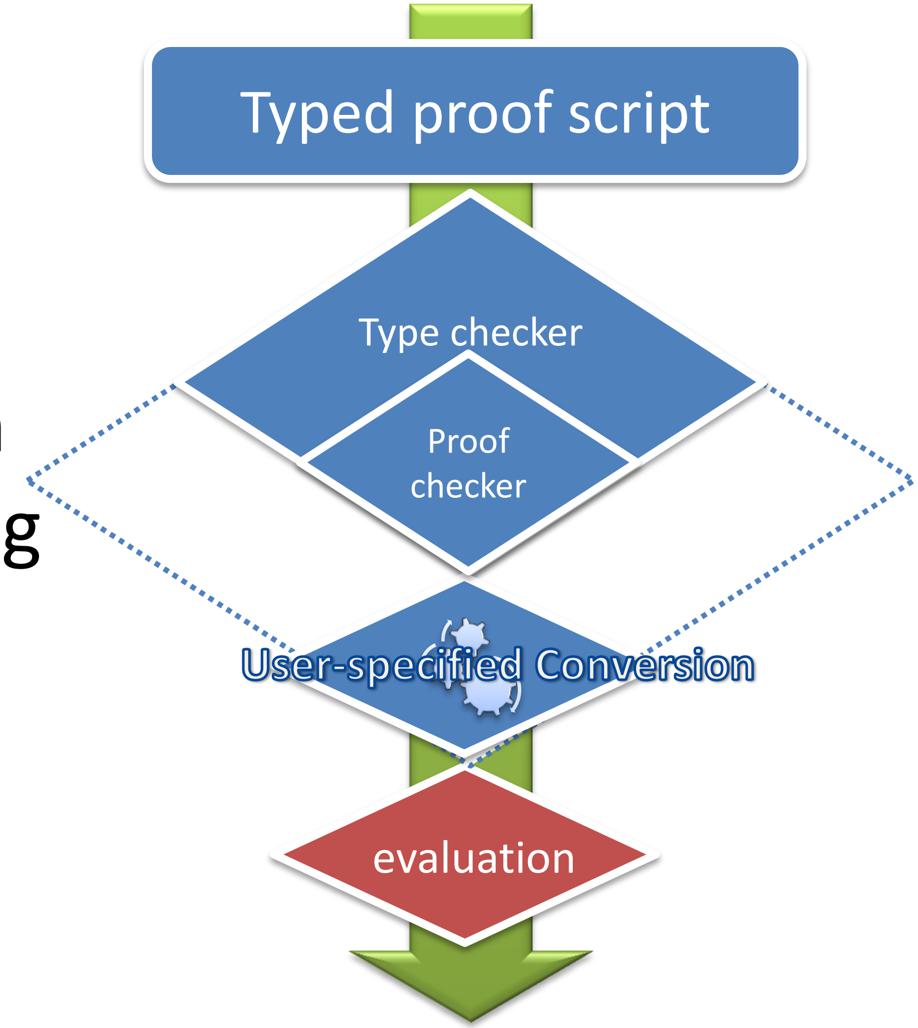
but we can trust other tactics too if they have the right type

none of them needs to be hardcoded!



Typed proof scripts + extensible conversion

- rich static information
- user chooses conversion
- extensible static checking
- smaller proof checker
- can generate proof objects



Type checking tactics: an example

$$\mathsf{conversionCheck} \quad : \quad (P : \mathsf{Prop}) \rightarrow (P' : \mathsf{Prop}) \rightarrow (\mathsf{proof}{:}P = P')$$

- check propositions for equivalence
- return a proof if they are
- raise an exception otherwise

Metatheory result 1. **Type safety**

If evaluation succeeds,

the returned proof object is valid

T… an example

$$\text{conversionCheck} \quad : \quad (P : \text{Prop}) \to (P' : \text{Prop}) \to (\text{proof:} P = P')$$

- check propositions for equivalence
- return a proof if they are
- raise an exception otherwise

T... an example

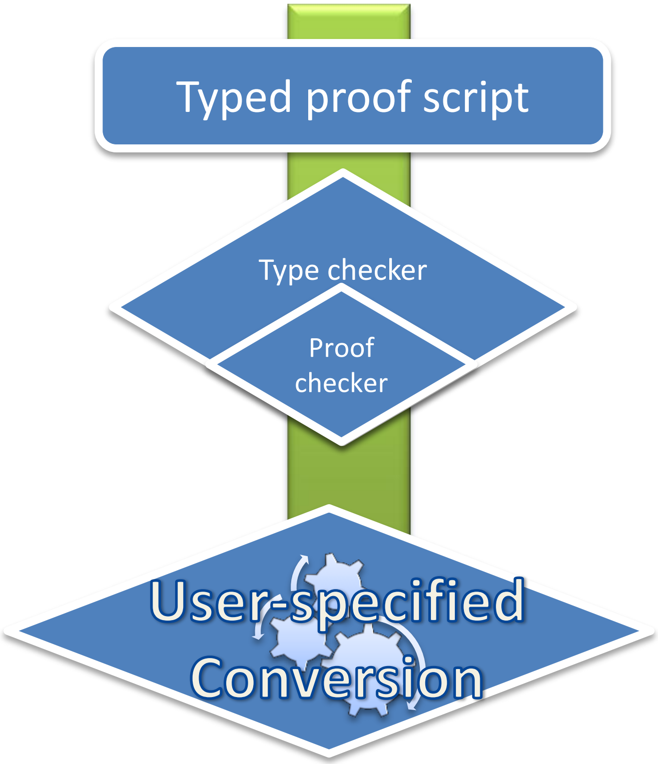
Metatheory result 1. **Type safety**
If evaluation succeeds,
the returned proof object is valid

$$\mathsf{conversionCheck} \quad : \quad (P : \mathsf{Prop}) \to (P' : \mathsf{Prop}) \to (\mathsf{proof}{:}P = P')$$

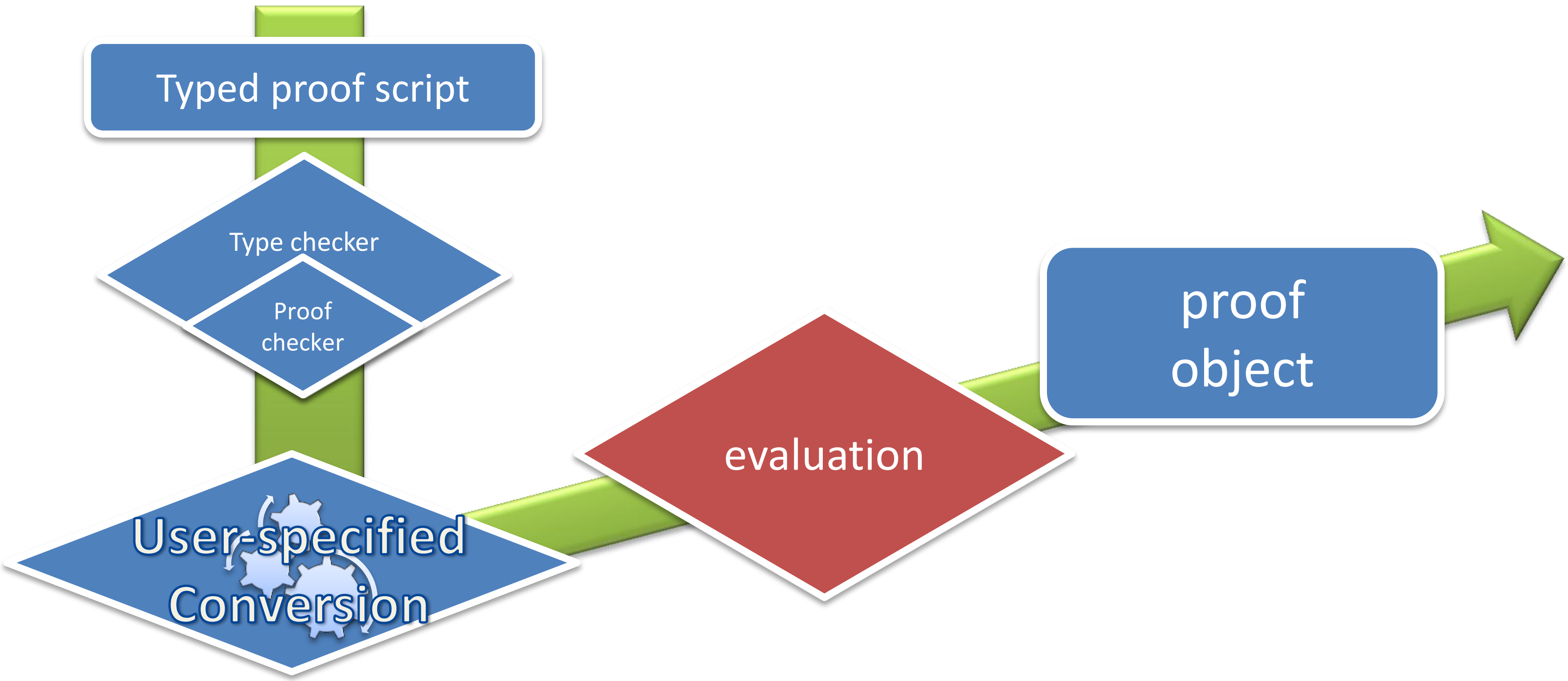
Metatheory result 2. **Proof erasure**
If evaluation succeeds, a valid proof object exists even if it's not generated

- chec...
- retu...
- raise an exception otherwise

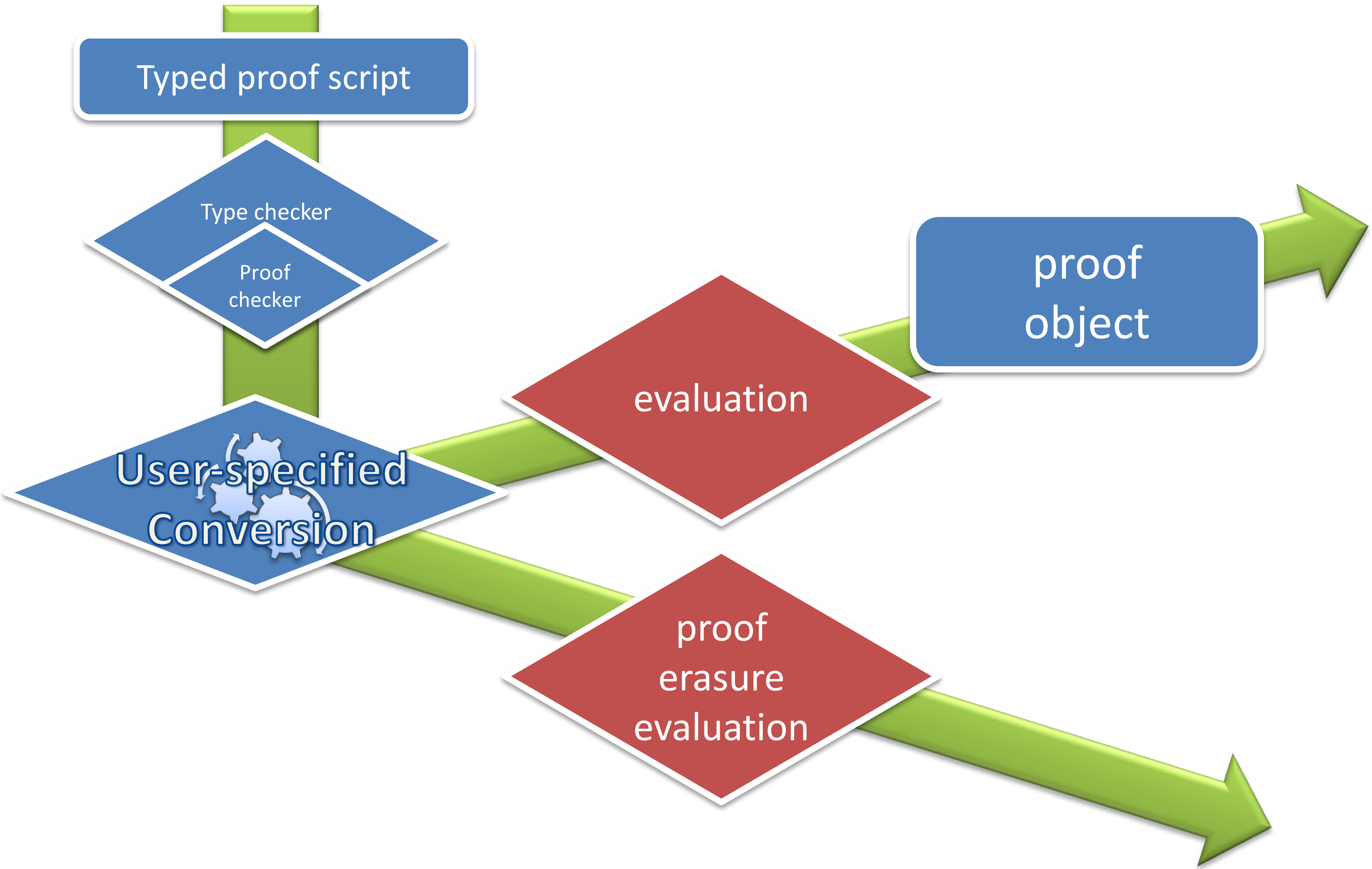
Two modes of evaluation



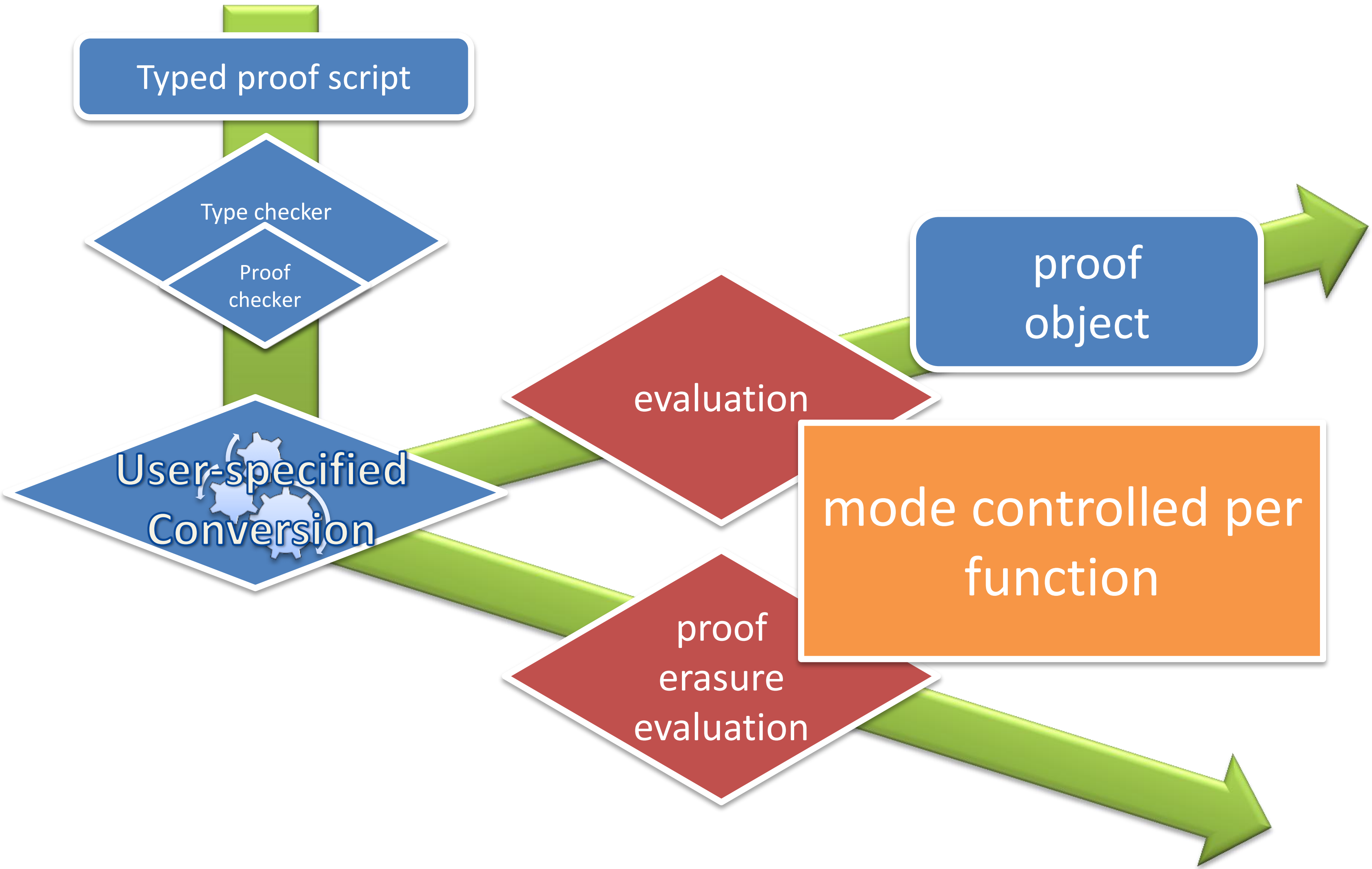
Two modes of evaluation



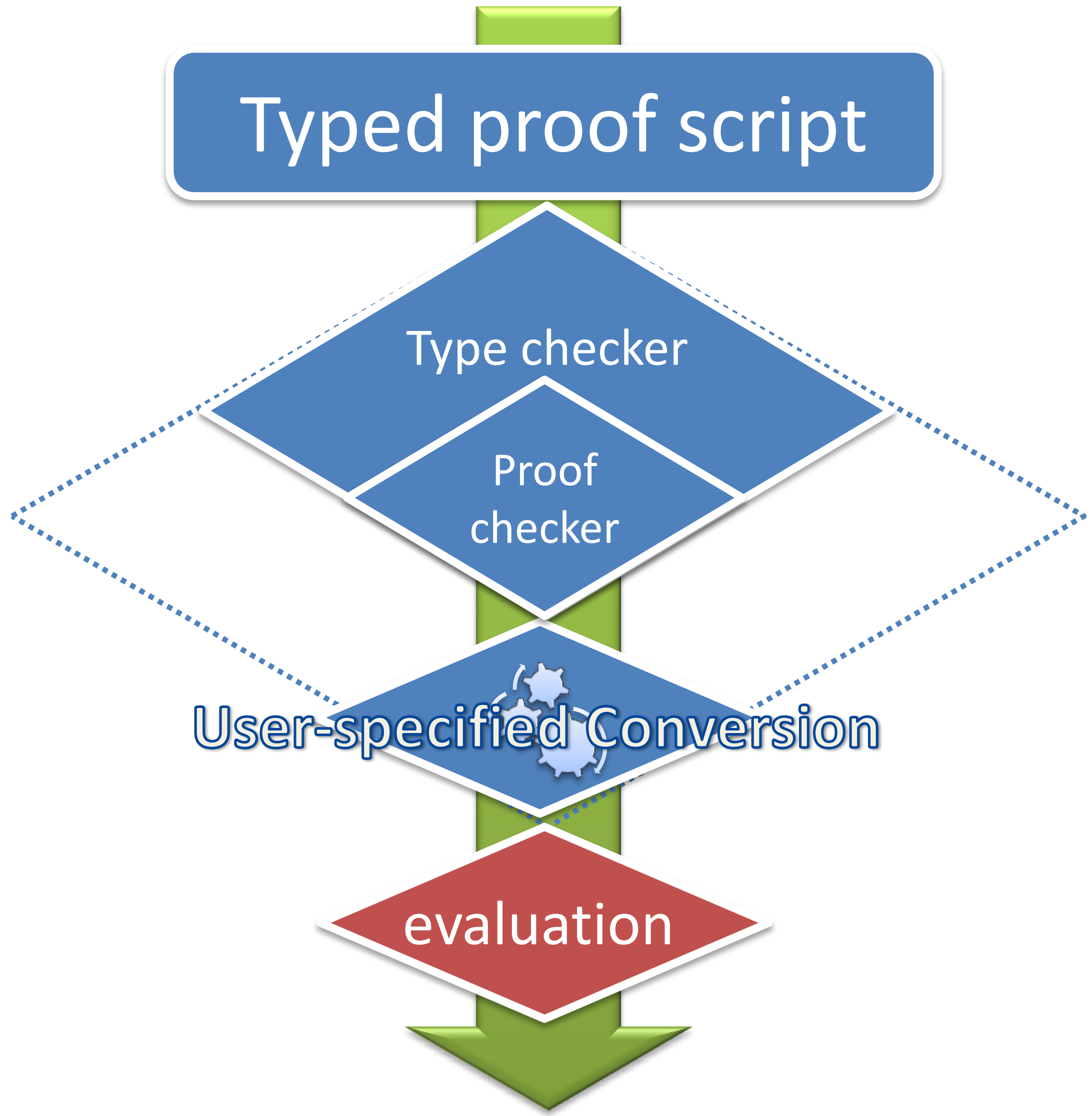
Two modes of evaluation



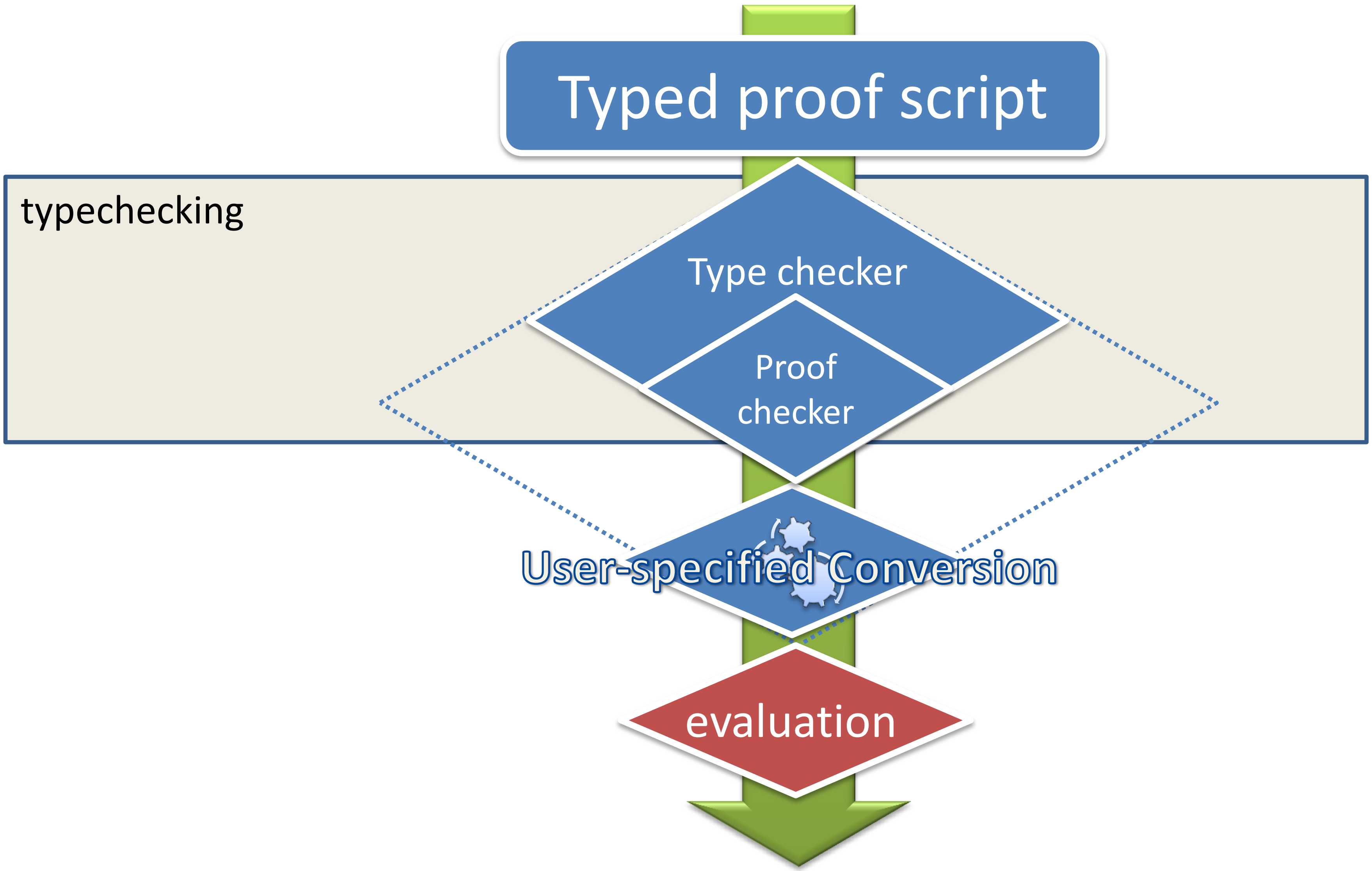
Two modes of evaluation



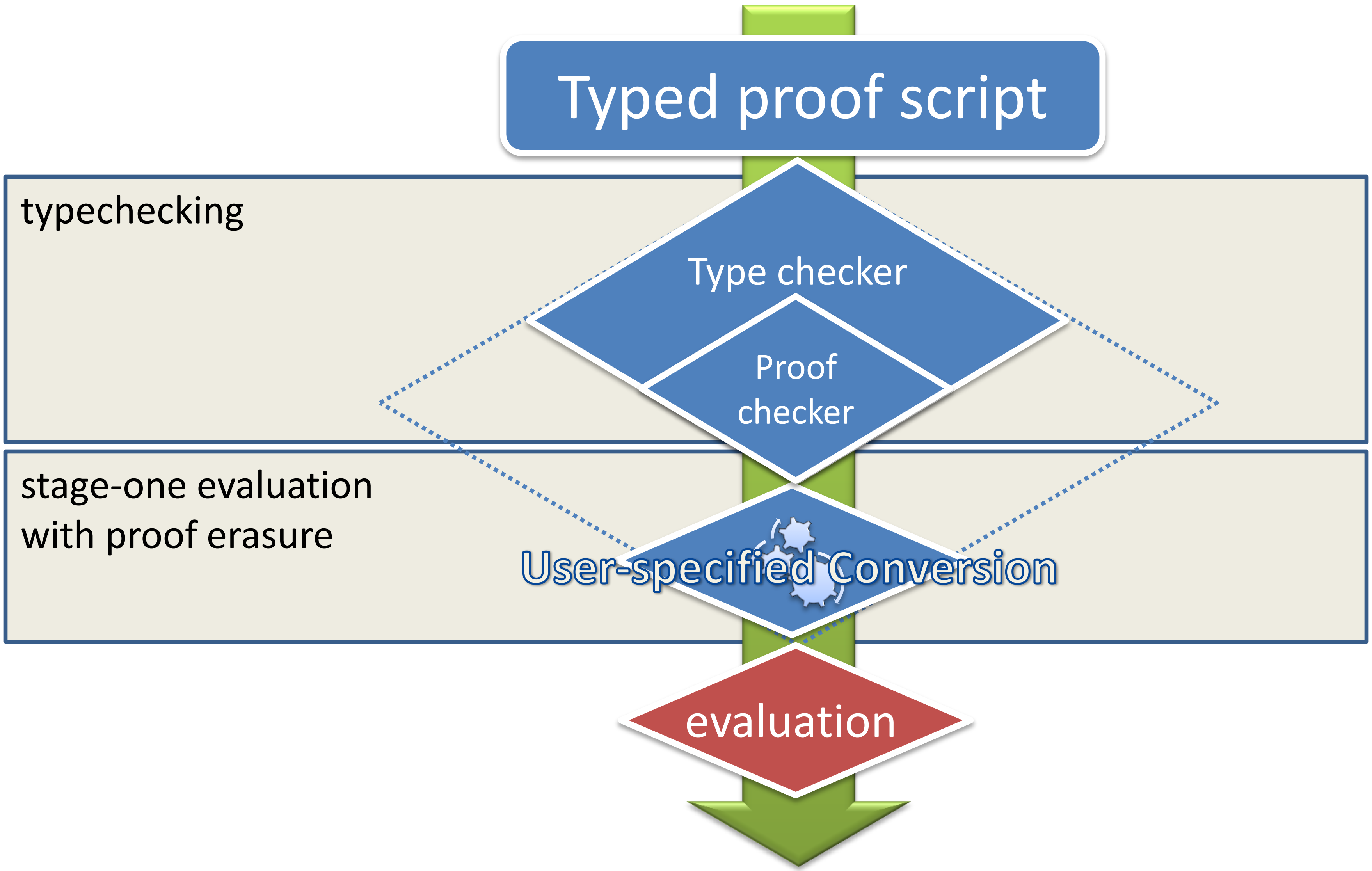
Static checking = type checking + staging under proof-erasure



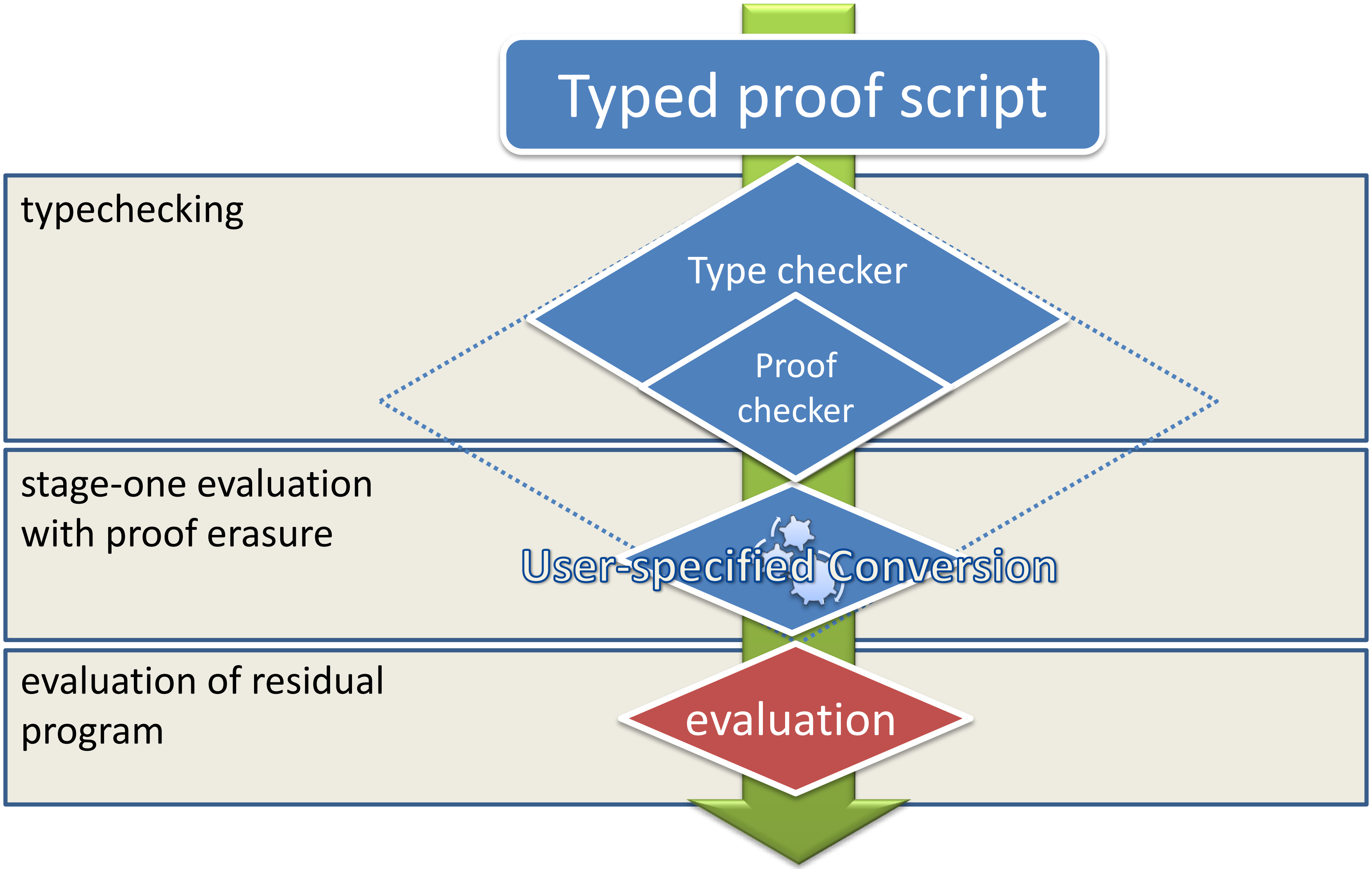
Static checking = type checking + staging under proof-erasure



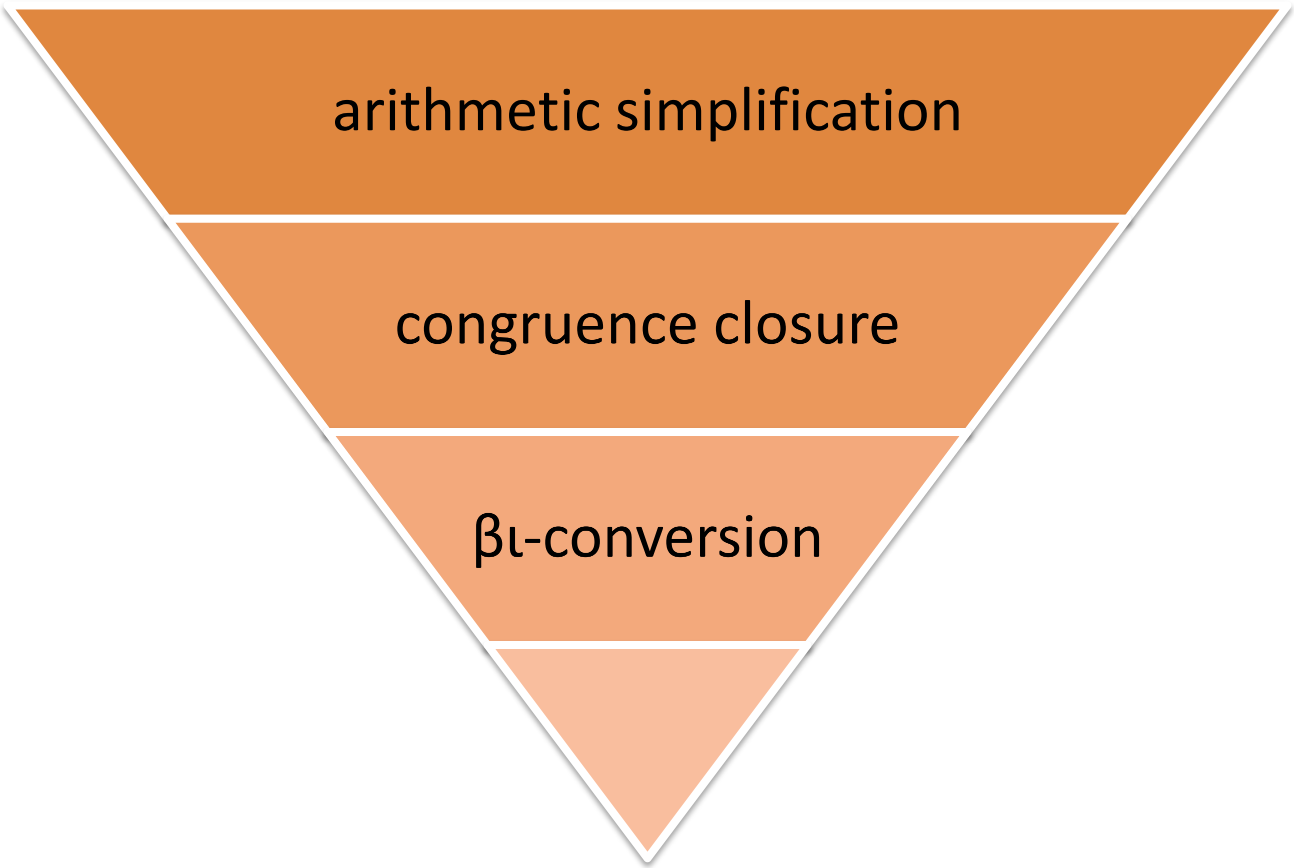
Static checking = type checking + staging under proof-erasure



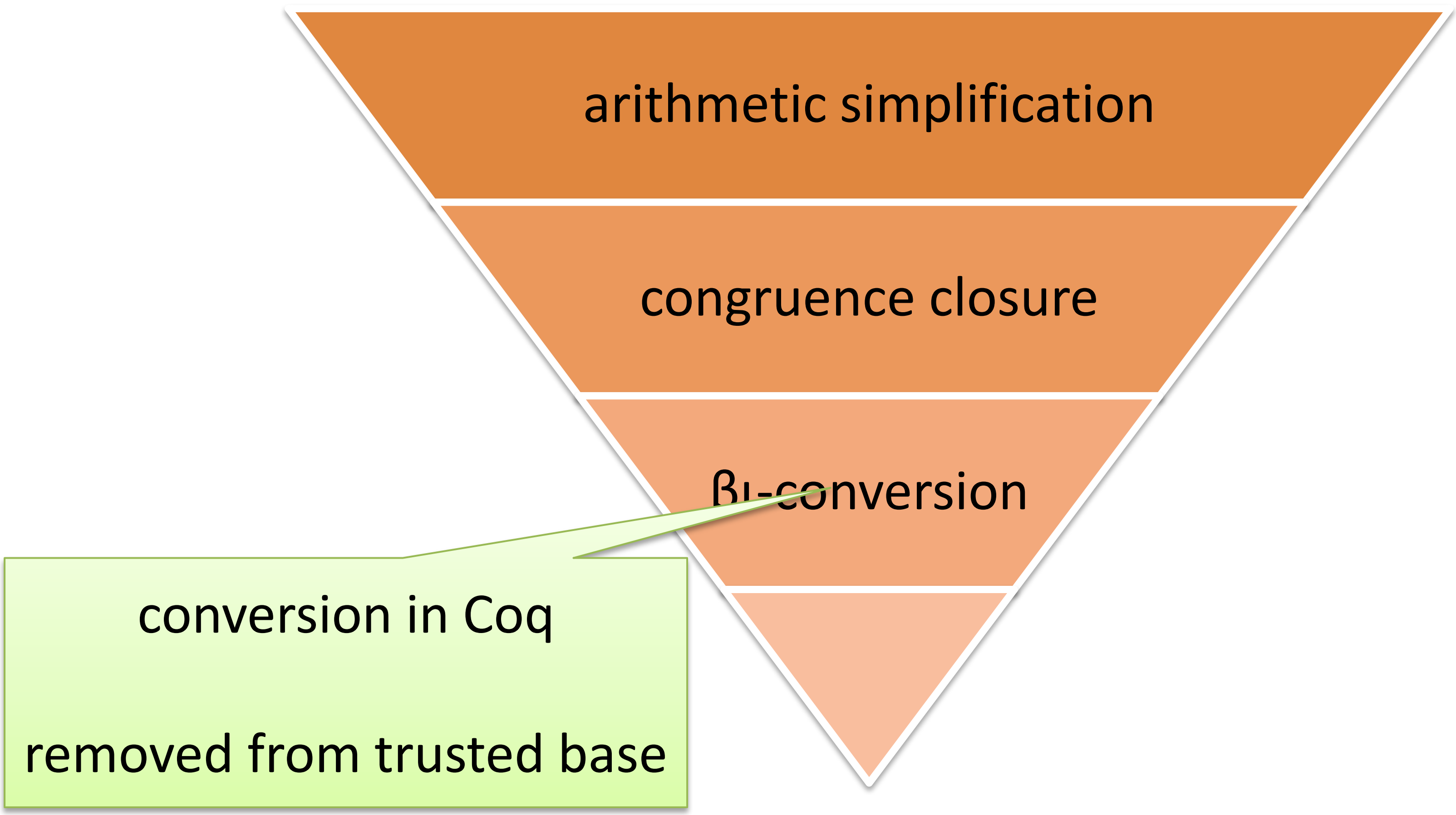
Static checking = type checking + staging under proof-erasure



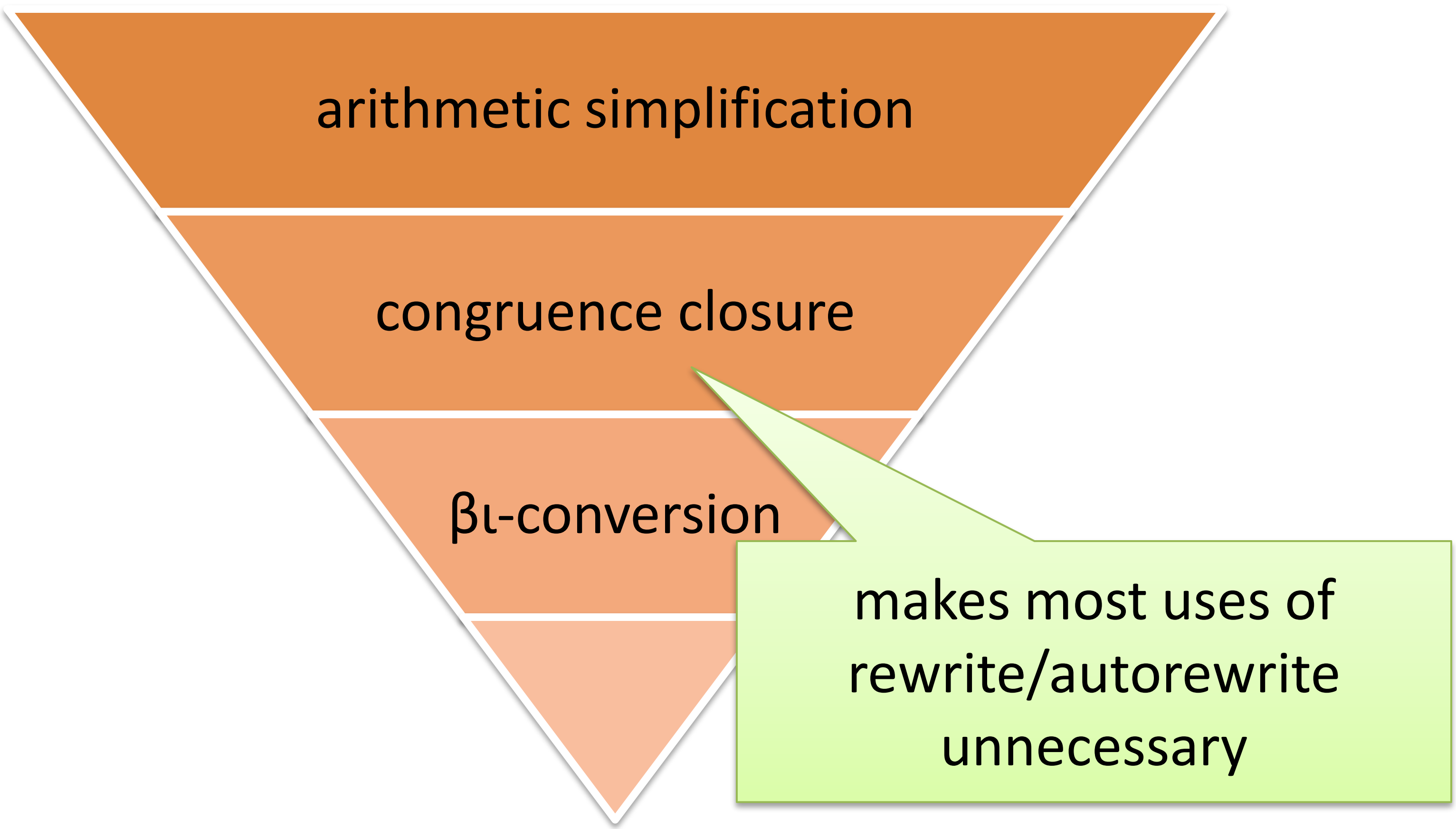
A stack of conversion rules



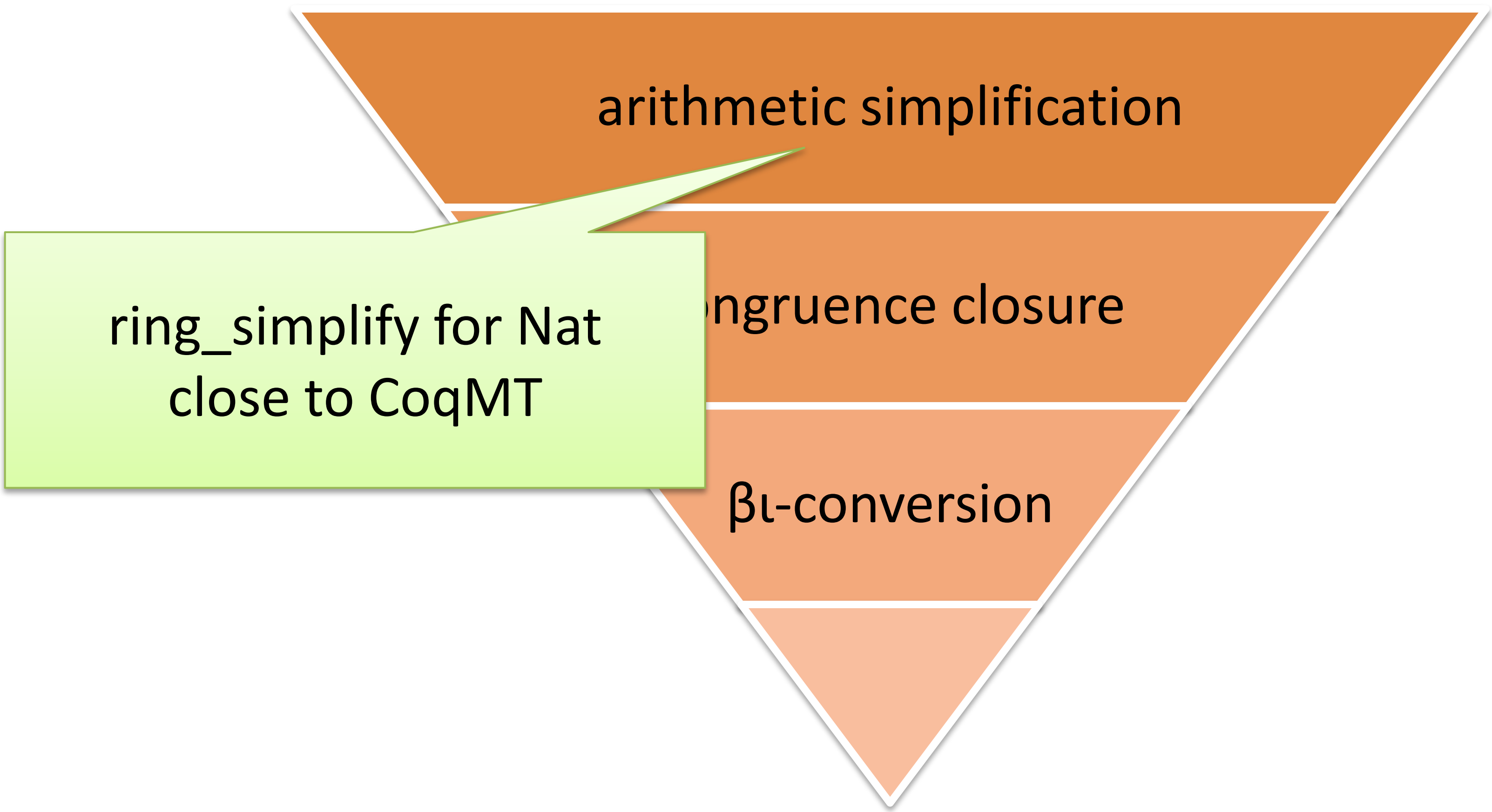
A stack of conversion rules



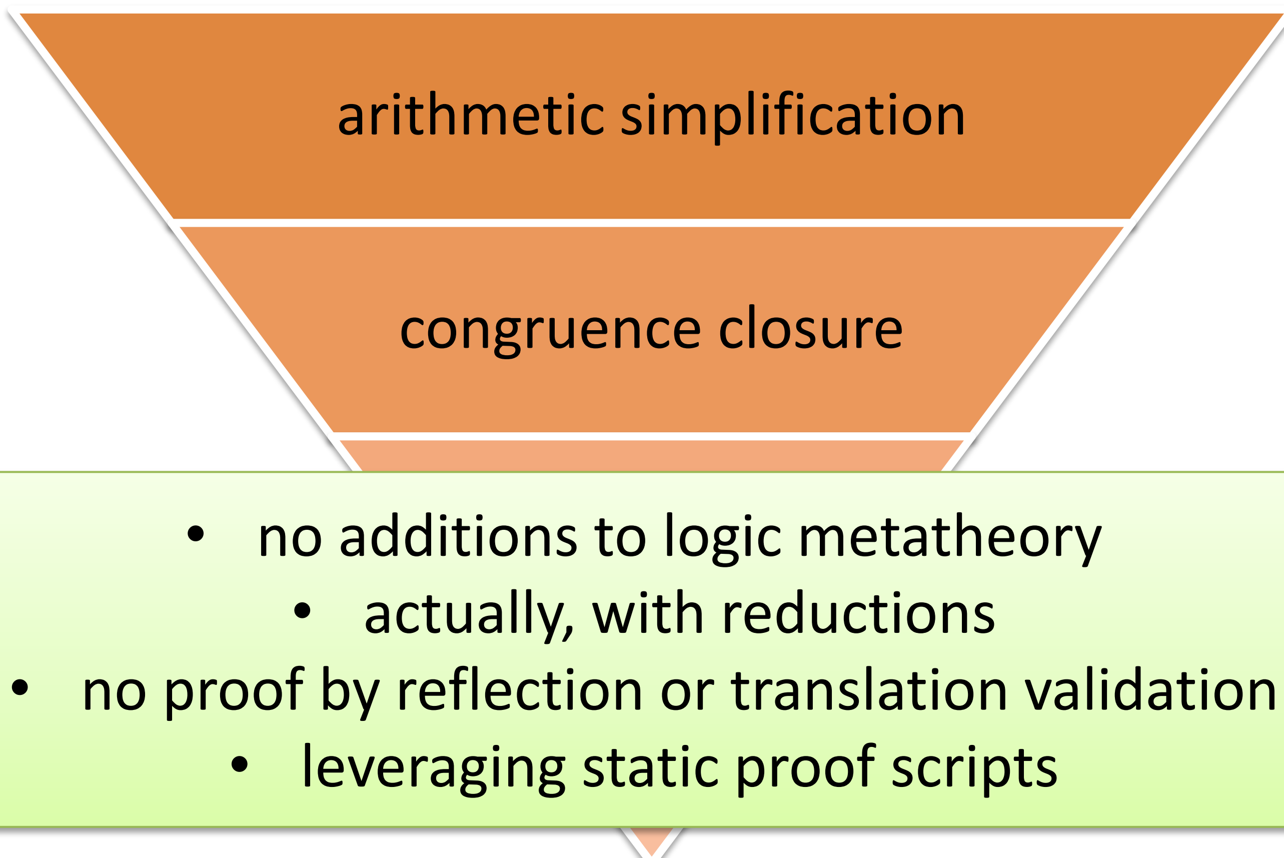
A stack of conversion rules



A stack of conversion rules

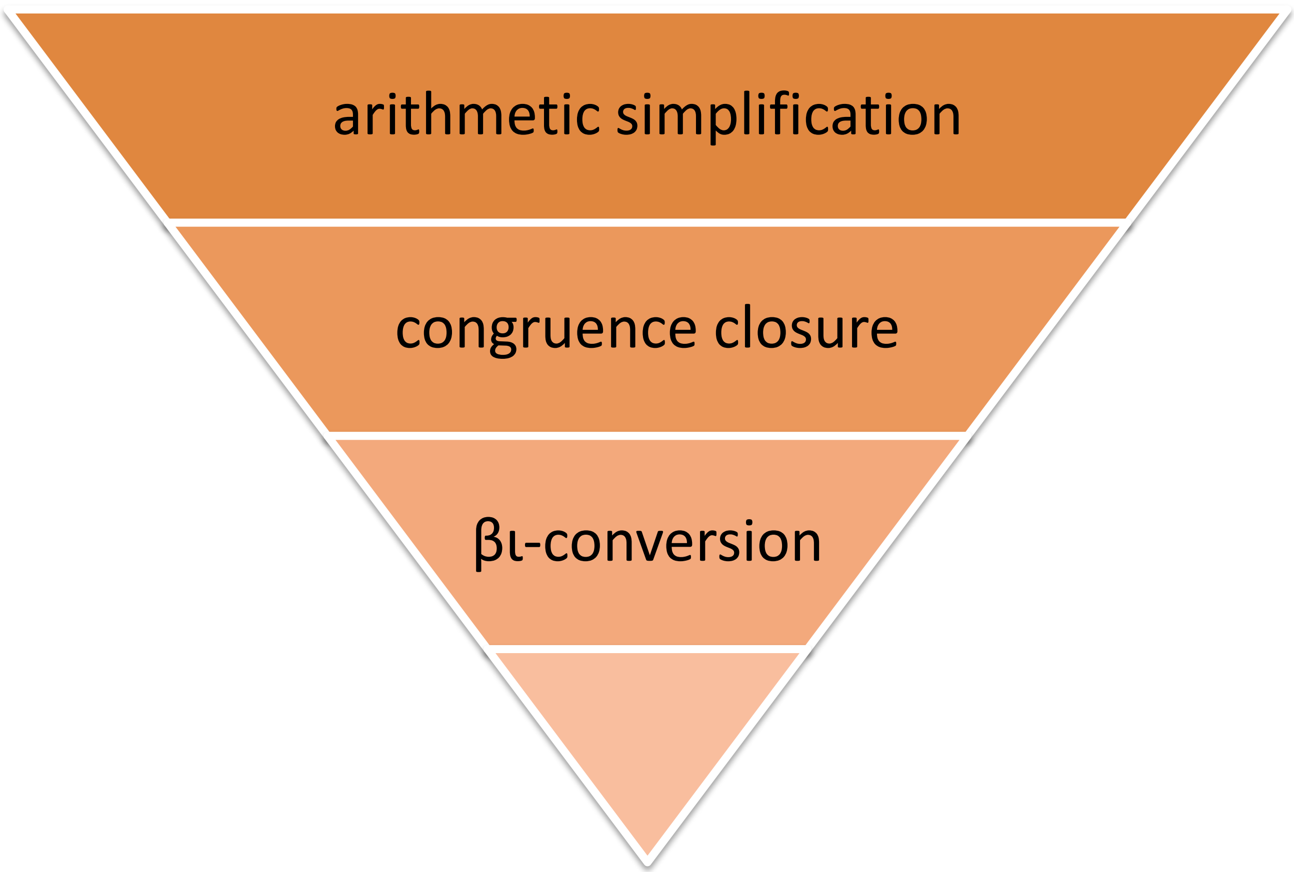


A stack of conversion rules

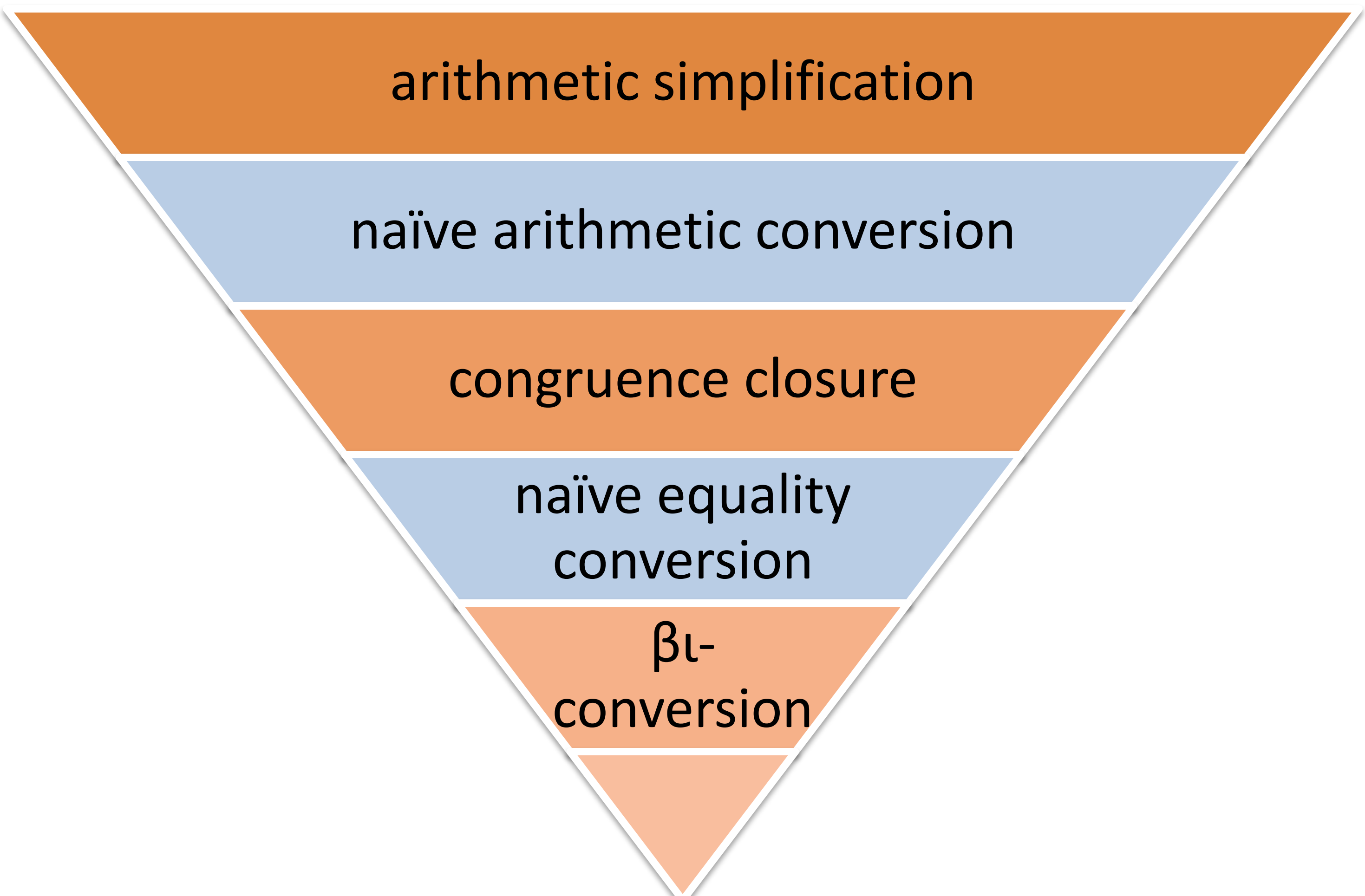


- no additions to logic metatheory
  - actually, with reductions
- no proof by reflection or translation validation
  - leveraging static proof scripts

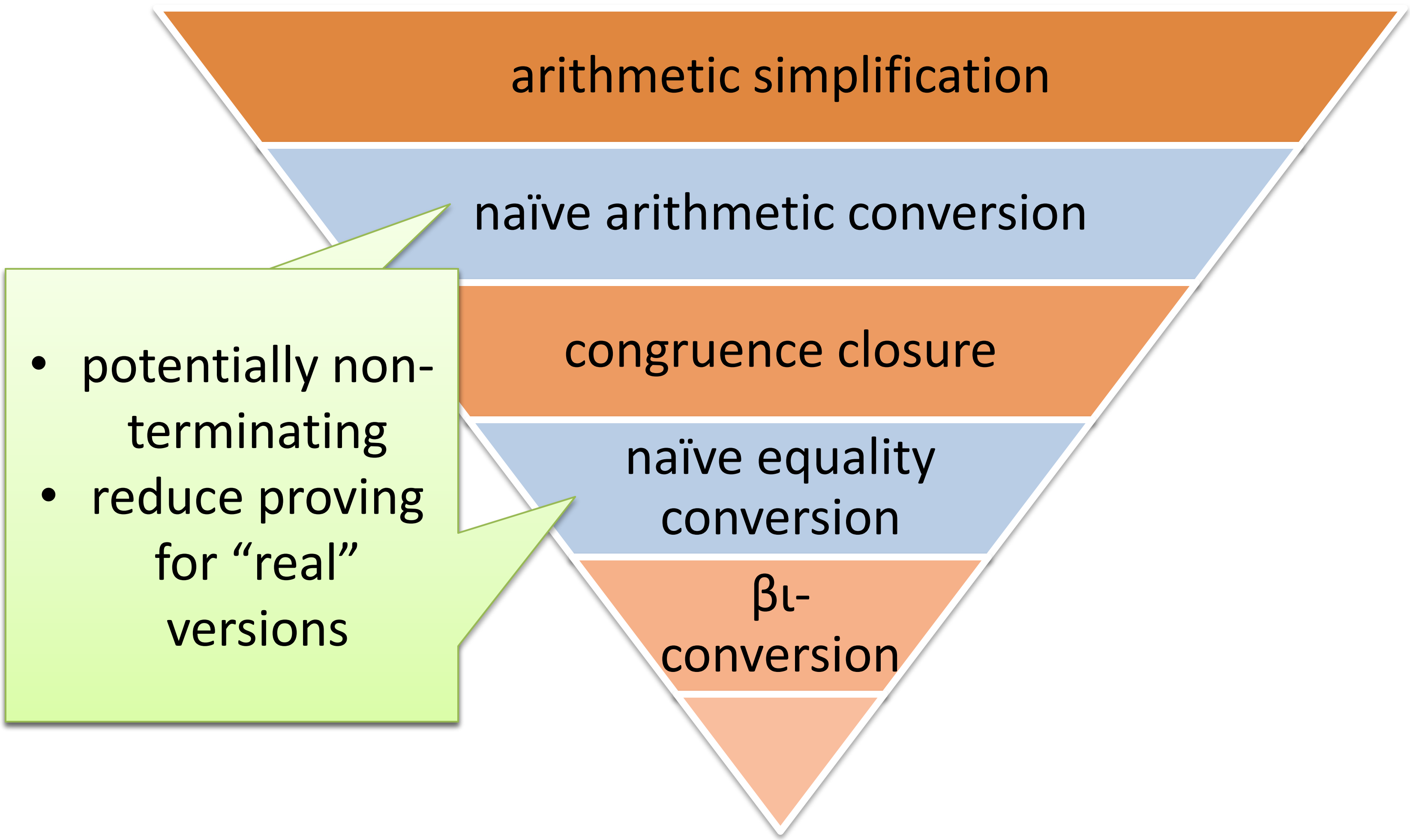
A stack of conversion rules



A stack of conversion rules



A stack of conversion rules



Static proof scripts in tactics

Motivating Coq Example

```
Require Import Arith.
Variable x : Nat.

Theorem test1 : 0 + x = x.
trivial.
Qed.

Theorem test2 : x + 0 = x.
trivial.
Qed.
```

Motivating Coq Example

```
Require Import Arith.
Variable x : Nat.

Theorem test1 : 0 + x = x.
trivial.
Qed.

Theorem test2 : x + 0 = x.
trivial.
Qed.
```

Proof completed

Motivating Coq Example

```
Require Import Arith.
Variable x : Nat.

Theorem test1 : 0 + x = x.
trivial.
Qed.

Theorem test2 : x + 0 = x.
trivial.
Qed.
```

Attempt to save an incomplete proof

Motivating Coq Example

```
Require Import Arith.
Variable x : Nat.

Theorem test1 : 0 + x = x.
trivial.
Qed.

Theorem test2 : x + 0 = x.
trivial.
Qed.
```

Conversion rule can prove this

but can't prove this

Let's add this to our conversion rule!

$$\begin{aligned} \mathsf{lemma1} &: \forall x. x + 0 = x \\ \mathsf{lemma2} &: \forall x y. x + (\mathsf{succ}\ y) = \mathsf{succ}(x + y) \end{aligned}$$

- write a rewriter based on these lemmas
- register it with conversion
- now it's trivial; lemmas used implicitly

$$\begin{aligned} \textsf{lemma1} &: \forall x. x + 0 = x \\ \textsf{lemma2} &: \forall x y. x + (\textsf{succ } y) = \textsf{succ}(x + y) \end{aligned}$$

$$\mathsf{lemma1} \quad : \quad \forall x. x + 0 = x$$

$$\mathsf{lemma2} \quad : \quad \forall x y. x + (\mathsf{succ}\ y) = \mathsf{succ}(x + y)$$

rewriter : $(t : T) \rightarrow (t' : T \times proof : t = t')$

rewriter t =

match t with

$x + y \mapsto$

let y', H = rewriter y in

match y' with

$0 \mapsto x, ?$

$|\ succ\ y'' \mapsto succ\ (x + y''), ?$

$|\ \_ \mapsto t, ?$

$|\ \ldots$

$$\text{lemma1} \quad : \quad \forall x. x + 0 = x$$

$$\text{lemma2} \quad : \quad \forall x y. x + (\text{succ } y) = \text{succ}(x + y)$$

rewriter : $(t : T) \rightarrow (t' : T \times proof : t = t')$

rewriter t =

  match t with

    $x + y \mapsto$

      let y', H = rewriter y

      match y' with

        $0 \mapsto x, ?$

        $\mid succ\ y'' \mapsto succ\ (x + y''), ?$

        $\mid \_ \mapsto t, ?$

  $\mid \ldots$

$$\frac{\begin{array}{l} x,\ y,\ y'' : T \\ H : y = succ\ y'' \end{array}}{proof : x + y = succ\ (x + y'')}$$

$$\text{lemma1} \quad : \quad \forall x. x + 0 = x$$

$$\text{lemma2} \quad : \quad \forall x y. x + (\text{succ } y) = \text{succ}(x + y)$$

rewriter : $(t : T) \rightarrow (t' : T \times proof : t = t')$

rewriter t =

match t with

$x + y \mapsto$

let y', H = rewriter y

match y' with

$0 \mapsto x, ?$

$\mid succ\ y'' \mapsto succ\ (x + y''),$

$\mid \_ \mapsto t, ?$

$\mid \ldots$

$$\frac{\begin{array}{l} x,\ y,\ y'' : T \\ H : y = succ\ y'' \end{array}}{proof : x + y = succ\ (x + y'')}$$

lemma2 x y''

$$\text{lemma1} \quad : \quad \forall x. x + 0 = x$$

$$\text{lemma2} \quad : \quad \forall x y. x + (\text{succ } y) = \text{succ}(x + y)$$

rewriter : $(t : T) \rightarrow (t' : T \times proof : t = t')$

rewriter t =

  match t with

    $x + y \mapsto$

      let y', H = rewriter y

      match y' with

        $0 \mapsto x, ?$

        $\mid succ\ y'' \mapsto succ\ (x + y''),$

        $\mid \_ \mapsto t, ?$

  $\mid \ldots$

$$\frac{\begin{array}{l} x,\ y,\ y'' : T \\ H : y = succ\ y'' \end{array}}{proof : x + y = succ\ (x + y'')}$$

lem… y''

$$\text{lemma1} \quad : \quad \forall x. x + 0 = x$$

$$\text{lemma2} \quad : \quad \forall x y. x + (\text{succ } y) = \text{succ}(x + y)$$

rewriter : $(t : T) \rightarrow (t' : T \times proof : t = t')$

rewriter t =

  match t with

   $x + y \mapsto$

     let $y', \; H \;=\;$ rewriter y

     match y' with

       $0 \qquad\quad \mapsto x, ?$

      $\mid succ\; y'' \mapsto succ\; (x + y''), ?$

      $\mid \_ \qquad\quad \mapsto t, ?$

  $\mid \ldots$

$$\frac{\begin{array}{l} x,\; y,\; y'' : T \\ H : y = succ\; y'' \end{array}}{proof : x + y = succ\; (x + y'')}$$

lemma1 : $\forall x . x + 0 = x$

lemma2 : $\forall x y . x + (\mathsf{succ}\ y) = \mathsf{succ}(x + y)$

lemma2' : $\forall x y y' . y = succ\ y' \rightarrow$
$x + y = succ(x + y')$

$= t')$

$x,\ y,\ y'' : T$
$H : y = succ\ y''$

$proof : x + y = succ\ (x + y'')$

rewriter t =
match t with
$x + y \mapsto$
let y', H = rewriter y
match y' with
$0 \mapsto x, ?$
| $succ\ y'' \mapsto succ\ (x + y''),$ lemma2' $x\ y\ y''$
| $\_ \mapsto t, ?$
| ...

lemma1 : $\forall x.x + 0 = x$

lemma2 : $\forall \; y.x + (\mathsf{succ}\ y) = \mathsf{succ}(x + y)$

lemma2' : $\forall \; y'.y = succ\ y' \rightarrow$

$x + y = succ(x + y')$

$= t')$

$x,\ y,\ y'' : T$

$H : y = succ\ y''$

$proof : x + y = succ\ (x + y'')$

rewriter t =

match t with

$x + y \mapsto$

let $y',\ H$ = rewriter y i

match y' with

$0 \mapsto x, ?$

| $succ\ y'' \mapsto succ\ (x + y''),$ lemma2' $x\ y\ y''$

| $\_ \mapsto t, ?$

| …

$$\text{lemma1} \quad : \quad \forall x. x + 0 = x$$

$$\text{lemma2} \quad : \quad \forall x y. x + (\text{succ } y) = \text{succ}(x + y)$$

$$\text{Exact} \; : \; (proof : P) \to (proof : P')$$

$$\text{rewriter} : (t : T) \to (t' : T \times proof : t = t')$$

$$\frac{\begin{array}{l} x,\; y,\; y'' : T \\ H : y = succ\, y'' \end{array}}{proof : x + y = succ\,(x + y'')}$$

rewriter t =

  match t with

  $x + y \mapsto$

    let y', H = rewriter y

    match y' with

$$\begin{array}{lll} & 0 & \mapsto x, ? \\ | & succ\, y'' & \mapsto succ\,(x + y''), ? \\ | & \_ & \mapsto t, ? \end{array}$$

  | ...

lemma1 : $\forall x. x + 0 = x$

lemma2 : $\forall x y. x + (\text{succ } y) = \text{succ}(x + y)$

Exact : $(proof : P) \to (proof : P')$

rewriter : $(t : T) \to (t' : T \times proof : t = t')$

rewriter t =

  match t with

    $x + y \mapsto$

      let y', H = rewriter y

      match y' with

        $0 \mapsto x, {?}$

        $\mid succ\ y'' \mapsto succ\ (x + y''), {?}$

        $\mid \_ \mapsto t, {?}$

  $\mid \ldots$

$$\frac{\begin{array}{l} x,\ y,\ y'' : T \\ H : y = succ\ y'' \end{array}}{proof : x + y = succ\ (x + y'')}$$

$$\text{lemma1} \;:\; \forall x. x + 0 = x$$

$$\text{lemma2} \;:\; \forall x y. x + (\text{succ } y) = \text{succ}(x + y)$$

$$\text{Exact} \;:\; (proof : P) \to (proof : P')$$

- similar to trivial
- uses conversion
- P and P' inferred

rewriter $: T) \to (t' : T \times proof : t = t')$

$$\frac{\begin{array}{l} x,\ y,\ y'' : T \\ H : y = succ\ y'' \end{array}}{proof : x + y = succ\ (x + y'')}$$

re

let y', H = rewriter y

match y' with

$$\begin{array}{lll} & 0 & \mapsto x, ? \\ | & succ\ y'' & \mapsto succ\ (x + y''), ? \\ | & \_ & \mapsto t, ? \end{array}$$

$| \ldots$

$\mathsf{lemma1} \quad : \quad \forall x. x + 0 = x$

$\mathsf{lemma2} \quad : \quad \forall x y. x + (\mathsf{succ}\ y) = \mathsf{succ}(x + y)$

$\mathsf{Exact} \ : \ (proof : P) \rightarrow (proof : P')$

$\mathsf{rewriter} : (t : T) \rightarrow (t' : T \times proof : t = t')$

$$\frac{\begin{array}{l} x,\ y,\ y'' : T \\ H : y = succ\ y'' \end{array}}{proof : x + y = succ\ (x + y'')}$$

$\mathsf{rewriter}\ t =$

$\quad \mathsf{match}\ t\ \mathsf{with}$

$\quad\quad x + y \mapsto$

$\quad\quad\quad \mathsf{let}\ y',\ H\ =\ \mathsf{rewriter}\ y$

$\quad\quad\quad \mathsf{match}\ y'\ \mathsf{with}$

$\quad\quad\quad\quad 0 \mapsto x, ?$

$\quad\quad\quad\quad |\ succ\ y'' \mapsto succ\ (x + y''),$

$\quad\quad\quad\quad |\ \_ \mapsto t, ?$

$\quad |\ \ldots$

Exact $(lemma2\ x\ y'')$

$\text{lemma1} \quad : \quad \forall x. x + 0 = x$

$\text{lemma2} \quad : \quad \forall x y. x + (\text{succ } y) = \text{succ}(x + y)$

$\text{Exact} \ : \ (proof : P) \rightarrow (proof : P')$

$\text{rewriter} : (t : T) \rightarrow (t' : T \times proof : t = t')$

$\text{rewriter } t =$

$\quad \text{match } t \text{ with}$

$\quad\quad x + y \mapsto$

$\quad\quad\quad \text{let } y', \ H \ = \ \text{rewriter } y \text{ i}$

$x, \ y, \ y'' : T$

$H : y = succ\ y''$

$$\frac{\begin{array}{l} x, \ y, \ y'' : T \\ H : y = succ\ y'' \end{array}}{proof : x + y = succ\ (x + y'')}$$

- not checked statically
- recomputed many times

$\quad\quad\quad | \ succ\ y'' \mapsto succ\ (x + y''),$ $\text{Exact } (lemma2\ x\ y'')$

$\quad\quad\quad | \ \_ \quad\quad \mapsto t, ?$

$\quad | \ \ldots$

$$\text{lemma1} \quad : \quad \forall x. x + 0 = x$$

$$\text{lemma2} \quad : \quad \forall x y. x + (\text{succ } y) = \text{succ}(x + y)$$

$$\text{Exact} \; : \; (proof : P) \rightarrow (proof : P')$$

$$\text{rewriter} : (t : T) \rightarrow (t' : T \times proof : t = t')$$

rewriter t =

  match t with

    $x + y \mapsto$

      let $y', \; H \; = \;$ rewriter y

      match y' with

        $0 \qquad\qquad \mapsto x, ?$

        $\mid succ\ y'' \mapsto succ\ (x + y''), ?$

        $\mid \_ \qquad\qquad \mapsto t, ?$

  $\mid \ldots$

$$\frac{x,\ y,\ y'' : T \qquad H : y = succ\ y''}{proof : x + y = succ\ (x + y'')}$$

$$\mathsf{lemma1} \quad : \quad \forall x. x + 0 = x$$

$$\mathsf{lemma2} \quad : \quad \forall x y. x + (\mathsf{succ}\ y) = \mathsf{succ}(x + y)$$

$$\mathsf{Exact}\ :\ (proof : P) \to (proof : P')$$

$$\mathsf{rewriter}\ :\ (t : T) \to (t' : T \times proof : t = t')$$

$$
\begin{array}{l}
\mathsf{rewriter}\ t = \\
\quad \mathsf{match}\ t\ \mathsf{with} \\
\quad\quad x + y \mapsto \\
\quad\quad\quad \mathsf{let}\ y',\ H \ = \ \mathsf{rewriter}\ y\ \mathsf{i} \\
\quad\quad\quad \mathsf{match}\ y'\ \mathsf{with} \\
\quad\quad\quad\quad 0 \qquad\quad \mapsto x, {\color{red}?} \\
\quad\quad\quad\ \mid succ\ y'' \mapsto succ\ (x + y''), \\
\quad\quad\quad\ \mid \_ \qquad\quad \mapsto t, {\color{red}?} \\
\quad \mid \ldots
\end{array}
$$

$$
\begin{array}{l}
x,\ y,\ y'' : T \\
H : y = succ\ y'' \\
\hline\hline
proof : x + y = succ\ (x + y'')
\end{array}
$$

$$
\left\{ \begin{array}{l} \mathsf{Exact} \\ (lemma2\ x\ y'') \end{array} \right\}_{static}
$$

$$\text{lemma1} \quad : \quad \forall x. x + 0 = x$$

$$\text{lemma2} \quad : \quad \forall x y. x + (\text{succ } y) = \text{succ}(x + y)$$

$$\text{Exact} \ : \ (proof : P) \to (proof : P')$$

$$\text{rewriter} : (t : T) \to (t' : T \times proof : t = t')$$

rewriter t =

match t with

$x + y \mapsto$

$$\begin{array}{l} x,\ y,\ y'' : T \\ H : y = succ\ y'' \\ \hline\hline proof : x + y = succ\ (x + y'') \end{array}$$

- checked at definition time
- computed once
- transformation of runtime arguments to constant arguments

$+ y''),$

$$\left\{\begin{array}{l}\text{Exact} \\ (lemma2\ x\ y'')\end{array}\right\}_{static}$$

$| \ldots$

How does it work?

rewriter : $\begin{array}{l}(\Phi : \mathsf{ctx}) \longrightarrow (T : [\Phi]\,\mathsf{Type}) \longrightarrow (t : [\Phi]\,T) \\ \longrightarrow (t' : [\Phi]\,T \times proof : [\Phi]\,t = t')\end{array}$

rewriter Φ t $=$

match t with

$[\Phi]\,x + y \mapsto$

let $[\Phi]\,y',\ [\Phi]\,H\ =$ rewriter $[\Phi]\,y$ in

match y' with

$$\begin{array}{lll} & [\Phi]\,0 & \mapsto [\Phi]\,x, ? \\ | & [\Phi]\,succ\ y'' & \mapsto [\Phi]\,succ\ (x + y''), \left\{\begin{array}{l}\text{Exact} \\ (lemma2\ x\ y'')\end{array}\right\}_{static} \\ | & \_ & \mapsto [\Phi]\,t, ? \end{array}$$

$| \ \ldots$

How does it work?

$$\text{Exact} \;:\; (\Phi : \mathsf{ctx}) \to (proof : [\Phi]\, P) \to (proof : [\Phi]\, P')$$

$$\text{rewriter} : \boxed{\begin{array}{l}(\Phi : \mathsf{ctx}) \to (T : [\Phi]\, \mathsf{Type}) \to (t : [\Phi]\, T) \\ \to (t' : [\Phi]\, T \times proof : [\Phi]\, t = t')\end{array}}$$

$$\begin{array}{l}
\text{rewriter } \Phi\ t = \\
\quad \text{match } t \text{ with} \\
\quad\quad [\Phi]\, x + y \mapsto \\
\quad\quad\quad\quad \text{let } [\Phi]\, y',\ [\Phi]\, H \ = \ \text{rewriter } [\Phi]\, y \text{ in} \\
\quad\quad\quad\quad \text{match } y' \text{ with} \\
\quad\quad\quad\quad\quad \begin{array}{lll} & [\Phi]\, 0 & \mapsto [\Phi]\, x, {\color{red}?} \\ | & [\Phi]\, succ\ y'' & \mapsto [\Phi]\, succ\ (x + y''), \left\{ \begin{array}{l} \text{Exact} \\ (lemma2\ x\ y'') \end{array} \right\}_{static} \\ | & _ & \mapsto [\Phi]\, t, {\color{red}?} \end{array} \\
\quad | \ \ldots
\end{array}$$

$$
\begin{array}{l}
\textsf{letstatic}\ pf\ =\\
\qquad \textsf{let}\ \Phi'\ =\ [x, y, y'' : \textsf{Nat}, H : y = succ\ y'']\ \textsf{in}\\
\qquad \textsf{Exact}\ \Phi'\ ([\Phi']\, lemma2\ x\ y'')\\
\textsf{in}\\
[\Phi]\, pf\,/[x/\textsf{id}_\Phi, y/\textsf{id}_\Phi, y'/\textsf{id}_\Phi, H/\textsf{id}_\Phi]
\end{array}
$$

Exact ...)

rewriter : ...

$$
\begin{array}{l}
\textsf{rewriter}\ \Phi\ t =\\
\quad \textsf{match}\ t\ \textsf{with}\\
\quad\quad [\Phi]\, x + y \mapsto\\
\quad\quad\quad \textsf{let}\ [\Phi]\, y',\ [\Phi]\, H\ =\ \textsf{rewriter}\ [\Phi]\, y\ \textsf{in}\\
\quad\quad\quad \textsf{match}\ y'\ \textsf{with}\\
\quad\quad\quad\quad [\Phi]\, 0 \mapsto [\Phi]\, x, ?\\
\quad\quad\quad\ \ \mid [\Phi]\, succ\ y'' \mapsto [\Phi]\, succ\ (x + y''), \left\{\begin{array}{l}\textsf{Exact}\\ (lemma2\ x\ y'')\end{array}\right\}_{static}\\
\quad\quad\quad\ \ \mid \_ \mapsto [\Phi]\, t, ?\\
\quad \mid \ldots
\end{array}
$$

Exact

rewriter :

rewriter Φ t =

match t w

$[\Phi]\; x +$

let $H = $ rewriter $[\Phi]\; y$ in

$2\; x\; y'')\}_{static}$

$|\; \_ \quad \mapsto [\Phi]\; t, ?$

$|\; \ldots$

letstatic pf =

let $\Phi' = [x, y, y'' : \mathsf{Nat}, H : y = succ\; y'']$ in

Exact Φ' $([\Phi']\; lemma2\; x\; y'')$

in

$[\Phi \ldots][x/\mathsf{id}_\Phi, y/\mathsf{id}_\Phi, y'/\mathsf{id}_\Phi, H/\mathsf{id}_\Phi]$

$$\frac{\bullet; \Sigma; \Gamma|_{\text{static}} \vdash e : \tau \qquad \Psi; \Sigma; \Gamma, x :_s \tau \vdash e' : \tau}{\Psi; \Sigma; \Gamma \vdash \text{letstatic}\; x = e \text{ in } e' : \tau}$$

Implementation

http://www.cs.yale.edu/homes/stampoulis/

- type inferencing and implicit arguments
- compiler to OCaml
- rewriter code generation
- inductive types

Talk to me for a demo!

What's in the paper and TR

- Static and dynamic semantics
- Metatheory:
  Type-safety theorem
  Proof erasure theorem
  Static proof script transformation
- Implementation details and examples implemented
- Typed proof scripts as flexible proof certificates

Related work

- proof-by-reflection
  - *restricted programming model (total functions)*
  - *tedious to set up*
  - *here: no need for termination proofs*
- automation through canonical structures / unification hints
  - *restricted programming model (logic programming)*
  - *very hard to debug*

Summary

- a new architecture for proof assistants
- user-extensible checking of proofs and tactics
- minimal trusted core
- reduce required effort for formal proofs

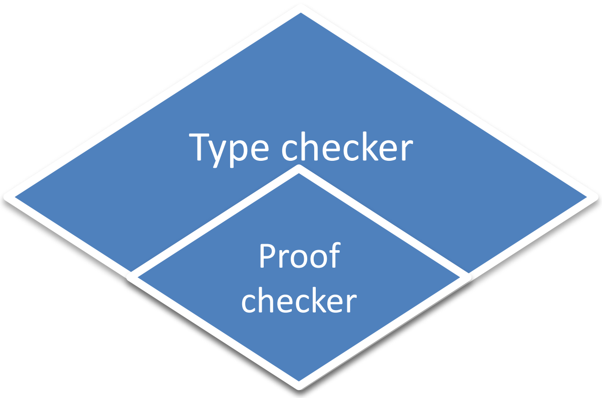
Thanks!

http://www.cs.yale.edu/homes/stampoulis/

Backup slides

Type checking proofs and tactics

-manipulate proofs and propositions in a type-safe manner
-dependent pattern matching on logical terms
-logic and computation are kept separate
-Beluga *[Pientka & Dunfield ‘08]*
-Delphin *[Poswolsky & Schürmann ‘08]*
-VeriML *[Stampoulis & Shao ‘10]*



Related work

- LCF family of proof assistants
  - *no information while writing proof scripts/tactics*
- Coq / CoqMT
  - *conversion rule is fixed*
  - *changing it requires re-engineering*
- NuPRL
  - *extensional type theory and sophisticated conversion*
  - *here: user decides conversion (level of undecidability)*
- Beluga / Delphin
  - *use as metalogic for LF*
  - *here: the logic is fixed; the language is the proof assistant*