Extensible and Efficient Automation through Reflective Tactics

Gregory Malecha   Jesper Bengtson
gmalecha@cs.ucsd.edu   jebe@itu.dk

ESOP'16

April 6, 2016

Naïve Proof Objects do not Scale

Naïve Proof Objects do not Scale

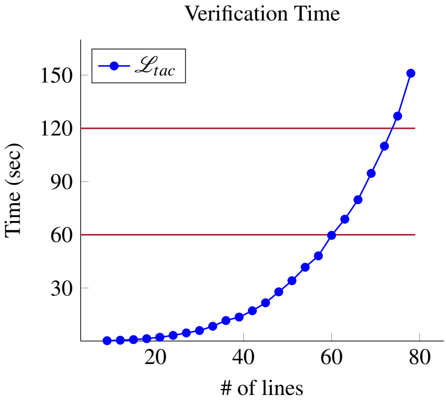
2+ hours

- Bedrock [Chl15]
- VST [App11, App14]
- Fiat [DPCGC15]
- CertiKOS [Sha15]

25 min

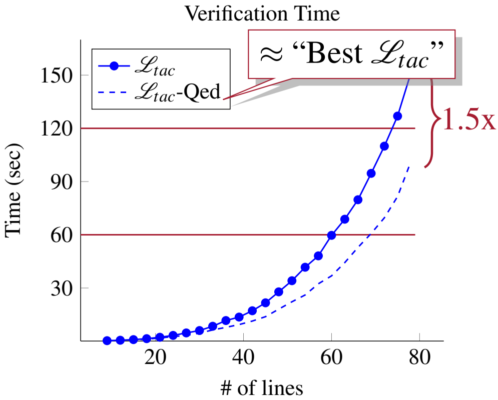
Naïve Proof Objects do not Scale

- Bedrock [Chl15]
- VST [App11, App14]
- Fiat [DPCGC15]
- CertiKOS [Sha15]



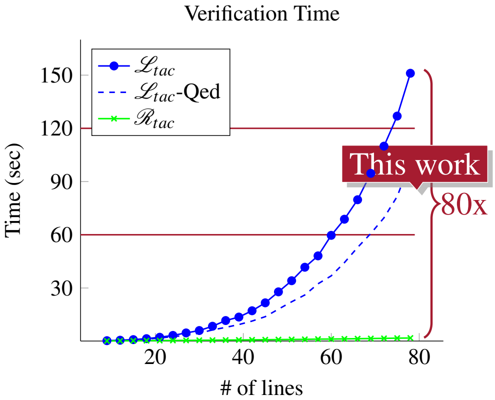
Naïve Proof Objects do not Scale

- Bedrock [Chl15]
- VST [App11, App14]
- Fiat [DPCGC15]
- CertiKOS [Sha15]



Naïve Proof Objects do not Scale

- Bedrock [Chl15]
- VST [App11, App14]
- Fiat [DPCGC15]
- CertiKOS [Sha15]



Background: Computational Reflection [Bou97]

Large proof $\sim O(n^2)$

$$\left\{ \begin{array}{c} A \oplus (B \oplus C) = A \oplus (B \oplus C) \\ \hline A \oplus (B \oplus C) = (A \oplus B) \oplus C \\ \hline A \oplus (B \oplus C) = C \oplus (A \oplus B) \\ \hline A \oplus (B \oplus C) = C \oplus (B \oplus A) \end{array} \right.$$

Background: Computational Reflection [Bou97]

Syntactic | Semantic

Denotation function

$$\frac{}{[\![A \oplus (B \oplus C) \equiv C \oplus (B \oplus A)]\!]_{Prop}}$$

Syntax

$$\cfrac{\cfrac{\cfrac{A \oplus (B \oplus C) = A \oplus (B \oplus C)}{A \oplus (B \oplus C) = (A \oplus B) \oplus C}}{A \oplus (B \oplus C) = C \oplus (A \oplus B)}}{A \oplus (B \oplus C) = C \oplus (B \oplus A)}$$

Background: Computational Reflection [Bou97]

Syntactic | Semantic

$$\mathrm{check}(A \oplus (B \oplus C) = C \oplus (B \oplus A)) = \mathrm{true}$$
$$\overline{[\![A \oplus (B \oplus C) = C \oplus (B \oplus A)]\!]_{Prop}}$$

```
Thm check_sound : ∀ g,
   check g = true → ⟦ g ⟧Prop.
Proof. ... Qed.
```

$$A \oplus (B \oplus C) = A \oplus (B \oplus C)$$
$$\overline{A \oplus (B \oplus C) = (A \oplus B) \oplus C}$$
$$\overline{A \oplus (B \oplus C) = C \oplus (A \oplus B)}$$
$$\overline{A \oplus (B \oplus C) = C \oplus (B \oplus A)}$$

Background: Computational Reflection [Bou97]

Syntactic | Semantic

$$\frac{\dfrac{\text{true} = \text{true} \ \checkmark}{\textsf{check}(A \underline{\oplus} (B \underline{\oplus} C) \underline{=} C \underline{\oplus} (B \underline{\oplus} A)) = \text{true}}}{[\![A \underline{\oplus} (B \underline{\oplus} C) \underline{=} C \underline{\oplus} (B \underline{\oplus} A)]\!]_{Prop}}$$

$$A \oplus (B \oplus C) = C \oplus (B \oplus A)$$

```
Thm check_sound : ∀ g,
   check g = true → ⟦ g ⟧Prop.
Proof. ... Qed.
```

Background: Computational Reflection [Bou97]

Syntactic | Semantic

Small proof, custom algorithm

$$\frac{\dfrac{\text{true} = \text{true} \;\checkmark}{\textsf{check}(A \oplus (B \oplus C) = C \oplus (B \oplus A)) = \text{true}}}{[\![A \oplus (B \oplus C) = C \oplus (B \oplus A)]\!]_{Prop}}$$

```
Thm check_sound : ∀ g,
   check g = true → ⟦ g ⟧Prop.
Proof. ... Qed.
```

Large proof

$$A \oplus (B \oplus C) = C \oplus (B \oplus A)$$

Reflection Recipe

1) Syntax

```
Ind 𝓔 :=
| e₁ ⊕ e₂
| 1
| ⌊ x ⌋
```

Reflection Recipe

1) Syntax

```
Ind ℰ :=
| e₁ ⊕ e₂
| 1
| ⌊ x ⌋
```

2) Reason

```
Fix check (e : ℰ) :=
  match e with
  | e_1 ⊕ e_2 ⇒
    check e_1 …
    check e_2
  | …
```

Reflection Recipe

1) Syntax

```
Ind 𝓔 :=
| e₁ ⊕ e₂
| 1
| ⌊ x ⌋
```

2) Reason

```
Fix check (e : 𝓔) :=
  match e with
  | e_1 ⊕ e_2 ⇒
    check e_1 ...
    check e_2
  | ...
```

3) Verify

```
Thm check_ok : ∀ e,
  check e = true →
  ⟦ e ⟧Prop.
Proof.
  induction e.
  (* proof *)
Qed.
```

Reflection Recipe

1) Syntax

```
Ind ℰ :=
| e₁ ⊕ e₂
| 1
| ⌊ x ⌋
```

2) Reason

```
Fix check (e : ℰ) :=
  match e with
  | e_1 ⊕ e_2 ⇒
    check e_1 ...
    check e_2
  | ...
```

3) Verify

```
Thm check_ok : ∀ e,
  check e = true →
  ⟦ e ⟧Prop.
Proof.
  induction e.
  (* proof *)
Qed.
```

- ✓ Highly customizable
- ✓ Very efficient

- ✗ Cumbersome to write
- ✗ Not extensible

Reflection with MirrorCore

1) Syntax

$$\lambda(\tau, \sigma)$$

Reflection with MIRRORCORE

1) Syntax

Generic language
w/ binders

$$\lambda(\tau, \sigma)$$

Domain-specific
types and symbols

Reflection with MirrorCore

1) Syntax

$$\lambda(\tau,\sigma)$$

2) Reason

```
Def check : rtac :=
 REPEAT10 FIRST
 [ APPLY lem1
 | REWRITE_STRAT ...
 | rtauto ].
```

Reflection with MIRRORCORE

1) Syntax

$\lambda(\tau, \sigma)$

2) Reason

$\mathscr{L}_{tac}$-inspired tactic language

```
Def check : rtac :=
  REPEAT10 FIRST
  [ APPLY lem1
  | REWRITE_STRAT ...
  | rtauto ].
```

Tactic combinators

Reasoning tactics

Reflection with MirrorCore

1) Syntax

$$\lambda(\tau,\sigma)$$

2) Reason

```
Def check : rtac :=
 REPEAT10 FIRST
 [ APPLY lem1
 | REWRITE_STRAT ...
 | rtauto ].
```

3) Verify

```
Thm check_ok :
 rtac_sound check.
Proof.
  rtac auto.
Qed.
```

Reflection with MIRRORCORE

1) Syntax

$\lambda(\tau, \sigma)$

2) P

Generic "soundness"

```
Def check : rtac :=
 REPEAT₁₀ FIRST
 [ APPLY lem1
 | REWRITE_STRAT ...
 | rtauto ].
```

3) Verify

```
Thm check_ok :
 rtac_sound check.
Proof.
  rtac auto.
Qed.
```

Automatic proofs†

Reflection with MirrorCore

1) Syntax

$$\lambda(\tau,\sigma)$$

2) Reason

```
Def check : rtac :=
 REPEAT10 FIRST
 [ APPLY lem1
 | REWRITE_STRAT ...
 | rtauto ].
```

3) Verify

```
Thm check_ok :
 rtac_sound check.
Proof.
  rtac auto.
Qed.
```

- ✓ Highly customizable
- ✓ Very efficient
- ✓ Easy to write
- ✓ Extensible

Reflection with MIRRORCORE

1) Syntax

$\lambda(\tau, \sigma)$

2) Reason

```
Def check : rtac :=
 REPEAT10 FIRST
 [ APPLY lem1
 | REWRITE_STRAT ...
 | rtauto ].
```

3) Verify

```
Thm check_ok :
 rtac_sound check.
Proof.
  rtac auto.
Qed.
```

$\mathscr{R}_{tac}$

- ✓ Highly customizable
- ✓ Very efficient
- ✓ Easy to write
- ✓ Extensible

The Soundness of Tactics

```
Definition rtac_spec ctx (s : CSUBST ctx) g r
: Prop :=
  match r with
  | Fail _ ⇒ True
  | Solved s' ⇒
    WellFormed_Goal (getUVars ctx) (getVars ctx) g →
    WellFormed_ctx_subst s →
    WellFormed_ctx_subst s' ∧
    match pctxD s
        , goalD (getUVars ctx) (getVars ctx) g
        , pctxD s'
    with
    | None , _ , _
    | Some _ , None , _ ⇒ True
    | Some _ , Some _ , None ⇒ False
    | Some cD , Some gD , Some cD' ⇒
      SubstMorphism s s' ∧
      ∀ us vs, cD' gD us vs
    end
  | More_ s' g' ⇒
    WellFormed_Goal (getUVars ctx) (getVars ctx) g →
    WellFormed_ctx_subst s →
    WellFormed_ctx_subst s' ∧
    WellFormed_Goal (getUVars ctx) (getVars ctx) g' ∧
    match pctxD s
        , goalD (getUVars ctx) (getVars ctx) g
        , pctxD s'
        , goalD (getUVars ctx) (getVars ctx) g'
    with
    | None , _ , _ , _
    | Some _ , None , _ , _ ⇒ True
    | Some _ , Some _ , None , _
    | Some _ , Some _ , Some _ , None ⇒ False
    | Some cD , Some gD , Some cD' , Some gD' ⇒
      SubstMorphism s s' ∧
      ∀ us vs,
        cD' (fun us vs ⇒ gD' us vs → gD us vs) us vs
    end
  end.
```

The Soundness of Tactics

```
Definition rtac_spec ctx (s : CSUBST ctx) g r
: Prop :=
  match r with
  | Fail _ ⇒ True
  | Solved s' ⇒
    WellFormed_Goal (getUVars ctx) (getVars ctx) g →
    WellFormed_ctx_subst s →
    WellFormed_ctx_subst s' ∧
    match pctxD s
        , goalD (getUVars ctx) (getVars ctx) g
        , pctxD s'
    with
    ...
                                  rs ctx) g →
    ...
                                  rs ctx) g' ∧
    ...
                                  ) g
        , pctxD s
        , goalD (getUVars ctx) (getVars ctx) g'
    with
    | None , _ , _ , _
    | Some _ , None , _ , _ ⇒ True
    | Some _ , Some _ , None , _
    | Some _ , Some _ , Some _ , None ⇒ False
    | Some cD , Some gD , Some cD' , Some gD' ⇒
      SubstMorphism s s' ∧
      ∀ us vs,
        cD' (fun us vs ⇒ gD' us vs → gD us vs) us vs
    end
  end.
```

$$\text{rtac\_sound}\ tac \triangleq \forall c\, g\, c'\, g',\ tac\ c\ g = \text{Some}(c', g') \rightarrow c \subseteq c' \wedge [\![c']\!]_{\text{ctx}} \left([\![g']\!]_{\text{goal}} \rightarrow_{c'} [\![g]\!]_{\text{goal}} \right)$$

The Soundness of Tactics

```
Definition rtac_spec ctx (s : CSUBST ctx) g r
: Prop :=
  match r with
  | Fail _ ⇒ True
  | Solved s' ⇒
    WellFormed_Goal (getUVars ctx) (getVars ctx) g →
    WellFormed_ctx_subst s →
    WellFo
    match
        , g
        , p
    with
                                                rs ctx) g →
                                                rs ctx) g' ∧
                                                ) g
        , pctxD s
        , goalD (getUVars ctx) (getVars ctx) g'
    with
    | None , _ , _ , _
    | Some _ , None , _ , _ ⇒ True
    | Some _ , Some _ , None , _
    | Some _ , Some _ , Some _ , None ⇒ False
    | Some cD , Some gD , Some cD' , Some gD' ⇒
      SubstMorphism s s' ∧
      ∀ us vs,
        cD' (fun us vs ⇒ gD' us vs → gD us vs) us vs
    end
  end.
```

Contexts and goals

$$\text{rtac\_sound}\ tac \triangleq \forall c\, g\, c'\, g',$$
$$tac\ c\ g = \text{Some}(c', g') \rightarrow$$
$$c \subseteq c'$$
$$\wedge [\![c']\!]_{\text{ctx}} \left([\![g']\!]_{\text{goal}} \rightarrow_{c'} [\![g]\!]_{\text{goal}}\right)$$

The Soundness of Tactics

```
Definition rtac_spec ctx (s : CSUBST ctx) g r
: Prop :=
  match r with
  | Fail _ ⇒ True
  | Solved s' ⇒
    WellFormed_Goal (getUVars ctx) (getVars ctx) g →
    WellFormed_ctx_subst s →
    WellFormed_ctx_subst s' ∧
    match pctxD s
        , goalD (getUVars ctx) (getVars ctx) g
        , pctxD s'
    with
```

$$\text{rtac\_sound}\ tac \triangleq \forall c\, g\, c'\, g',$$

Tactic succeeds

$$tac\ c\ g = \text{Some}(c', g') \rightarrow$$

$$c \subseteq c'$$

$$\wedge [\![c']\!]_{\text{ctx}} \left([\![g']\!]_{\text{goal}} \rightarrow_{c'} [\![g]\!]_{\text{goal}} \right)$$

```
rs ctx) g →
rs ctx) g' ∧
) g
    , pctxD s
    , goalD (getUVars ctx) (getVars ctx) g'
  with
  | None , _ , _ , _
  | Some _ , None , _ , _ ⇒ True
  | Some _ , Some _ , None , _
  | Some _ , Some _ , Some _ , None ⇒ False
  | Some cD , Some gD , Some cD' , Some gD' ⇒
    SubstMorphism s s' ∧
    ∀ us vs,
      cD' (fun us vs ⇒ gD' us vs → gD us vs) us vs
  end
end
```

The Soundness of Tactics

```
Definition rtac_spec ctx (s : CSUBST ctx) g r
: Prop :=
  match r with
  | Fail _ ⇒ True
  | Solved s' ⇒
    WellFormed_Goal (getUVars ctx) (getVars ctx) g →
    WellFormed_ctx_subst s →
    WellFormed_ctx_subst s' ∧
    match pctxD s
        , goalD (getUVars ctx) (getVars ctx) g
        , pctxD s'
    with
...
rs ctx) g →
...
rs ctx) g' ∧
...
) g
        , pctxD s
        , goalD (getUVars ctx) (getVars ctx) g'
    with
    | None , _ , _ , _
    | Some _ , None , _ , _ ⇒ True
    | Some _ , Some _ , None , _
    | Some _ , Some _ , Some _ , None ⇒ False
    | Some cD , Some gD , Some cD' , Some gD' ⇒
      SubstMorphism s s' ∧
      ∀ us vs,
        cD' (fun us vs ⇒ gD' us vs → gD us vs) us vs
    end
end
```

$$\mathsf{rtac\_sound}\ tac \triangleq \forall c\, g\, c'\, g',$$
$$tac\ c\ g = \mathsf{Some}(c', g') \rightarrow$$
$$c \subseteq c'$$
$$\wedge [\![c']\!]_{\mathsf{ctx}} \left([\![g']\!]_{\mathsf{goal}} \rightarrow_{c'} [\![g]\!]_{\mathsf{goal}}\right)$$

Consistent context extension

The Soundness of Tactics

```
Definition rtac_spec ctx (s : CSUBST ctx) g r
: Prop :=
  match r with
  | Fail _ ⇒ True
  | Solved s' ⇒
    WellFormed_Goal (getUVars ctx) (getVars ctx) g →
    WellFormed_ctx_subst s →
    WellFormed_ctx_subst s' ∧
    match pctxD s
        , goalD (getUVars ctx) (getVars ctx) g
        , pctxD s'
    with
    ...
                                        rs ctx) g →
                                        rs ctx) g' ∧
                                              g
        , pctxD s
        , goalD (getUVars ctx) (getVars ctx) g'
    ...  , _ , _
       , None , _ , _ ⇒ True
       , Some _ , None , _
       , Some _ , Some _ , None ⇒ False
    | Some cD , Some gD , Some cD' , Some gD' ⇒
      SubstMorphism s s' ∧
      ∀ us vs,
        cD' (fun us vs ⇒ gD' us vs → gD us vs) us vs
    end
  end.
```

$$\text{rtac\_sound}\; tac \triangleq \forall c\, g\, c'\, g',$$
$$tac\; c\; g = \text{Some}(c', g') \rightarrow$$
$$c \subseteq c'$$
$$\wedge [\![c']\!]_{\text{ctx}} \left([\![g']\!]_{\text{goal}} \rightarrow_{c'} [\![g]\!]_{\text{goal}} \right)$$

In the final context, the result goal implies the input goal

The Soundness of Tactics

```
Definition rtac_spec ctx (s : CSUBST ctx) g r
: Prop :=
  match r with
  | Fail _ ⇒ True
  | Solved s' ⇒
    WellFormed_Goal (getUVars ctx) (getVars ctx) g →
    WellFormed_ctx_subst s →
    WellFormed_ctx_subst s' ∧
    match pctxD s
        , goalD (getUVars ctx) (getVars ctx) g
        , pctxD s'
    with
```

$$\text{rtac\_sound}\; tac \triangleq \forall c\, g\, c'\, g',\; tac\; c\; g = \text{Some}(c', g') \rightarrow c \subseteq c' \wedge [\![c']\!]_{\text{ctx}} \left([\![g']\!]_{\text{goal}} \rightarrow_{c'} [\![g]\!]_{\text{goal}} \right)$$

```
rs ctx) g →
rs ctx) g' ∧
) g
        , pctxD s
        , goalD (getUVars ctx) (getVars ctx) g'
    with
    | None , _ , _ , _
    | Some _ , None , _ , _ ⇒ True
    | Some _ , Some
    | Some _ , Some
    | Some cD , Som
      SubstMorph
      ∀ us vs,
        cD' (fun us vs ⇒ gD' us vs → gD us vs) us vs
    end
  end.
```

Paper contains full details of context reasoning.

Reflective Reasoning

Fully reflective

$$\underbrace{y{:}\mathbb{N}, y > 0}_{\text{Context}} \vdash \underbrace{\mathsf{False} \vee \exists x, x = y \wedge x > 0}_{\text{Goal}}$$

Reflective Reasoning

Fully reflective

```
Thm vI : ∀ P Q,
  Q → P ∨ Q
```

$$\frac{}{y{:}\mathbb{N}, y > 0 \vdash \mathsf{False} \vee \exists x, x = y \wedge x > 0}$$

Reflective Reasoning

Fully reflective

- **Local reasoning**

```
Thm vI : ∀ P Q,
  Q → P ∨ Q
```

$$\dfrac{\dfrac{}{y{:}\mathbb{N}, y > 0 \vdash \exists x, x = y \land x > 0}}{y{:}\mathbb{N}, y > 0 \vdash \mathsf{False} \lor \exists x, x = y \land x > 0}\ \lor\text{-I}$$

Reflective Reasoning

Fully reflective

- Local reasoning

Existential quantifiers

$$\dfrac{\overline{y{:}\mathbb{N}, y > 0 \vdash \exists x, x = y \wedge x > 0}}{y{:}\mathbb{N}, y > 0 \vdash \mathsf{False} \vee \exists x, x = y \wedge x > 0} \vee\text{-I}$$

Reflective Reasoning

Fully reflective

- Local reasoning
- **Unification variables**

Fresh unification variables

$$\dfrac{\dfrac{\dfrac{}{y{:}\mathbb{N}, y > 0, ?x{:}\mathbb{N} \quad \vdash ?x = y \wedge ?x > 0}}{y{:}\mathbb{N}, y > 0 \vdash \exists x, x = y \wedge x > 0}}{y{:}\mathbb{N}, y > 0 \vdash \mathsf{False} \vee \exists x, x = y \wedge x > 0} \; \vee\text{-I}$$

Reflective Reasoning

Fully reflective

- Local reasoning
- Unification variables

$$\dfrac{\dfrac{\overline{\quad y{:}\mathbb{N}, y > 0, ?x{:}\mathbb{N} \qquad \vdash ?x = y \wedge ?x > 0 \quad}}{y{:}\mathbb{N}, y > 0 \vdash \exists x, x = y \wedge x > 0}}{y{:}\mathbb{N}, y > 0 \vdash \mathsf{False} \vee \exists x, x = y \wedge x > 0}\ \vee\text{-I}$$

Reflective Reasoning

Fully reflective

- Local reasoning
- Unification variables
- **Multiple goals**

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\ldots, ?x{:}\mathbb{N} \quad \vdash ?x = y}\textsc{Refl} \qquad \dfrac{}{\ldots, y > 0, ?x{:}\mathbb{N} \quad \vdash ?x > 0}}{y{:}\mathbb{N}, y > 0, ?x{:}\mathbb{N} \quad \vdash ?x = y \wedge ?x > 0}}{y{:}\mathbb{N}, y > 0 \vdash \exists x, x = y \wedge x > 0}}{y{:}\mathbb{N}, y > 0 \vdash \mathsf{False} \vee \exists x, x = y \wedge x > 0}\vee\text{-I}$$

Local reasoning

Reflective Reasoning

Fully reflective

- Local reasoning
- Unification variables & **instantiation**
- Multiple goals

Instatiate unification variables

Affects parallel goals

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\ldots, ?x{:}\mathbb{N} = \mathbf{y} \vdash ?x = y}\;\textsc{Refl} \qquad \dfrac{}{\ldots, y > 0, ?x{:}\mathbb{N} = \mathbf{y} \vdash ?x > 0}}{y{:}\mathbb{N}, y > 0, ?x{:}\mathbb{N} = \mathbf{y} \vdash ?x = y \wedge ?x > 0}}{y{:}\mathbb{N}, y > 0 \vdash \exists x, x = y \wedge x > 0}}{y{:}\mathbb{N}, y > 0 \vdash \mathsf{False} \vee \exists x, x = y \wedge x > 0}\;\vee\text{-I}$$

Reflective Reasoning

Fully reflective

- Local reasoning
- Unification variables & instantiation
- Multiple goals

$$\textsc{Refl}\ \dfrac{\checkmark}{\ldots, ?x{:}\mathbb{N} = \mathbf{y} \vdash ?x = y} \qquad \dfrac{}{\ldots, y > 0, ?x{:}\mathbb{N} = \mathbf{y} \vdash ?x > 0}$$

$$\dfrac{\dfrac{\dfrac{\ldots, ?x{:}\mathbb{N} = \mathbf{y} \vdash ?x = y \qquad \ldots, y > 0, ?x{:}\mathbb{N} = \mathbf{y} \vdash ?x > 0}{y{:}\mathbb{N}, y > 0, ?x{:}\mathbb{N} = \mathbf{y} \vdash ?x = y \land ?x > 0}}{y{:}\mathbb{N}, y > 0 \vdash \exists x, x = y \land x > 0}}{y{:}\mathbb{N}, y > 0 \vdash \mathsf{False} \lor \exists x, x = y \land x > 0}\ \lor\text{-I}$$

Reflective Reasoning

Fully reflective

- Local reasoning
- Unification variables & instantiation
- Multiple goals
- **Assumptions**

$$
\dfrac{\dfrac{\dfrac{\text{REFL}\;\dfrac{\checkmark}{\ldots, ?x{:}\mathbb{N} = \mathbf{y} \vdash ?x = y} \qquad \dfrac{\checkmark}{\ldots, y > 0, ?x{:}\mathbb{N} = \mathbf{y} \vdash ?x > 0}}{y{:}\mathbb{N}, y > 0, ?x{:}\mathbb{N} = \mathbf{y} \vdash ?x = y \wedge ?x > 0}}{y{:}\mathbb{N}, y > 0 \vdash \exists x, x = y \wedge x > 0}}{y{:}\mathbb{N}, y > 0 \vdash \mathsf{False} \vee \exists x, x = y \wedge x > 0}\;\vee\text{-I}
$$

Reflective Reasoning

Fully reflective

- Local reasoning
- Unification variables & instantiation
- Multiple goals
- Assumptions

Proof expressed by computation
(Small proof term)

✓ ✓

REFL

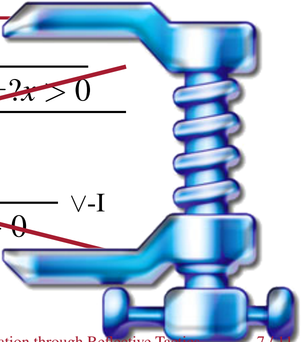
$$\ldots, ?x{:}\mathbb{N} = \mathbf{y} \vdash ?x = y \qquad \ldots, y > 0, ?x{:}\mathbb{N} = \mathbf{y} \vdash ?x > 0$$

$$y{:}\mathbb{N}, y > 0, ?x{:}\mathbb{N} = \mathbf{y} \vdash ?x = y \wedge ?x > 0$$

$$y{:}\mathbb{N}, y > 0 \vdash \exists x, x = y \wedge x > 0$$

$\vee$-I

$$y{:}\mathbb{N}, y > 0 \vdash \mathsf{False} \vee \exists x, x = y \wedge x > 0$$



Using Tactics

```
Thm my_lem : ∀ P,
   { P } Skip { P }.
Proof.
   (* Ltac proof *)
Qed.
```

Using Tactics

```
Thm my_lem : ∀ P,
  { P } Skip { P }.
Proof.
  (* ℒtac proof *)
Qed.
```

```
Def APPLY : lemma → rtac.

Thm APPLY_sound : ∀ l,
  ⟦ l ⟧lemma →
  rtac_sound (APPLY l).
```

Using Tactics

```
Thm my_lem : ∀ P,
  { P } Skip { P }.
Proof.
  (* ℒtac proof *)
Qed.

Reify Build Lemma
  < ...  >
  syn_my_lem : my_lem.
```

{ vars : list τ
; prems : list $\mathscr{E}$
; concl : $\mathscr{E}$ }

```
Def APPLY : lemma → rtac.

Thm APPLY_sound : ∀ l,
  mma →
  rtac_sound (APPLY l).
```

Construct automatically

Using Tactics

```
Thm my_lem : ∀ P,
  { P } Skip { P }.
Proof.
  (* ℒtac proof *)
Qed.

Reify Build Lemma
  < ... >
  syn_my_lem : my_lem.

Def use_it : rtac :=
  APPLY syn_my_lem.
```

```
Def APPLY : lemma → rtac.

Thm APPLY_sound : ∀ l,
  ⟦ l ⟧lemma →
  rtac_sound (APPLY l).
```

Using Tactics

```
Thm my_lem : ∀ P,
  { P } Skip { P }.
Proof.
  (* ℒtac proof *)
Qed.

Reify Build Lemma
  < ...  >
  syn_my_lem : my_lem.

Def use_it : rtac :=
  APPLY syn_my_lem.
```

```
Def APPLY : lemma → rtac.

Thm APPLY_sound : ∀ l,
  ⟦ l ⟧lemma →
  rtac_sound (APPLY l).
```

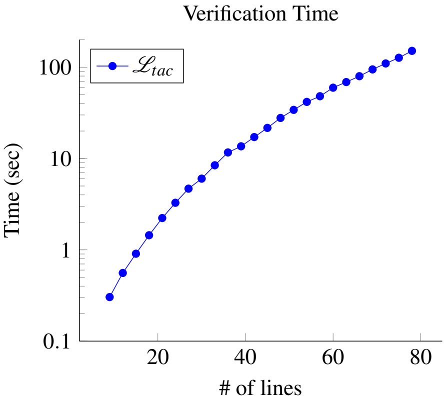
Case Study: Program Verification

Post-condition calc

```
Ltac sp :=
  first
   [ apply skip_rule ; sp
   | apply assign_rule ; sp
   | ..  ].
```

Entailment check

```
Ltac chk :=
  repeat eapply ex_i ;
  repeat conj_split ;
  ...
```



$\mathcal{L}_{tac} \rightarrow$ naïve $\mathcal{R}_{tac}$ ($\approx$ 1 day)

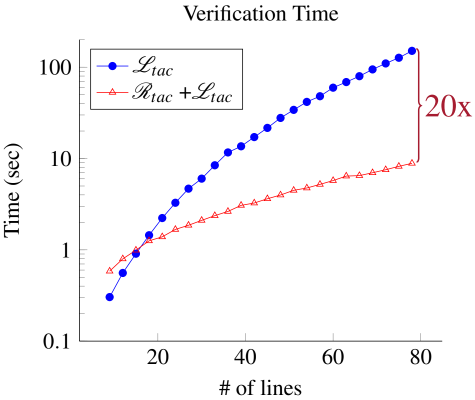
Case Study: Program Verification

Post-condition calc[†]

```
Def sp : rtac :=
  REC 100 (fun sp ⇒ FIRST
    [ APPLY lem_skip ;; sp
    | APPLY lem_assign ;; sp
    | ..  ].
```

Entailment check

```
Ltac chk :=
  repeat eapply ex_i ;
  repeat conj_split ;
  ...
```



$\mathscr{L}_{tac} \to$ naïve $\mathscr{R}_{tac}$ ($\approx$ 1 day)

† "Representative" code

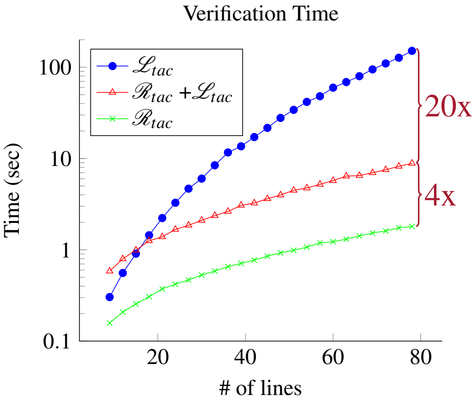
Case Study: Program Verification

Post-condition calc†

```
Def sp : rtac :=
  REC 100 (fun sp ⇒ FIRST
    [ APPLY lem_skip ;; sp
    | APPLY lem_assign ;; sp
    | ..  ].
```

Entailment check†

```
Def chk : rtac :=
  REPEAT 10
   (APPLY lem_ex_i ;; INTRO)
  ;;  APPLY lem_conj ;; ...
```



$\mathcal{L}_{tac} \to$ naïve $\mathcal{R}_{tac}$ ($\approx$ 1 day)

† "Representative" code

Case Study: Lifting Quantifiers w/ Rewriting

```
  P
∧ (∃ x : nat, Q x)
∧ (∃ y : nat, R y)
```

⇓

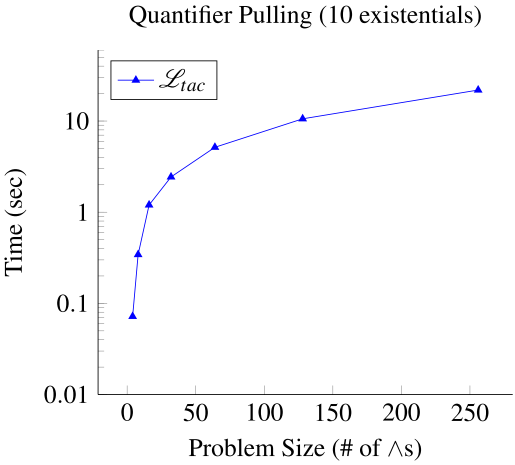
```
∃ x y : nat,
   P ∧ Q x ∧
   R y
```

Case Study: Lifting Quantifiers w/ Rewriting

```
  P
∧ (∃ x : nat, Q x)
∧ (∃ y : nat, R y)
```

⇓

```
 ∃ x y : nat,
   P ∧ Q x ∧
   R y
```

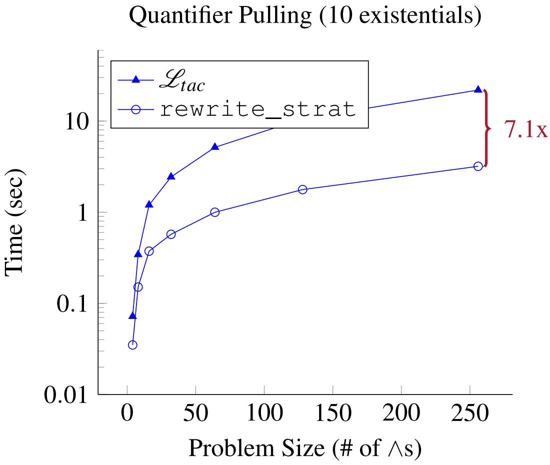


Case Study: Lifting Quantifiers w/ Rewriting

```
  P
∧ (∃ x : nat, Q x)
∧ (∃ y : nat, R y)
```

⇓

```
∃ x y : nat,
  P ∧ Q x ∧
  R y
```

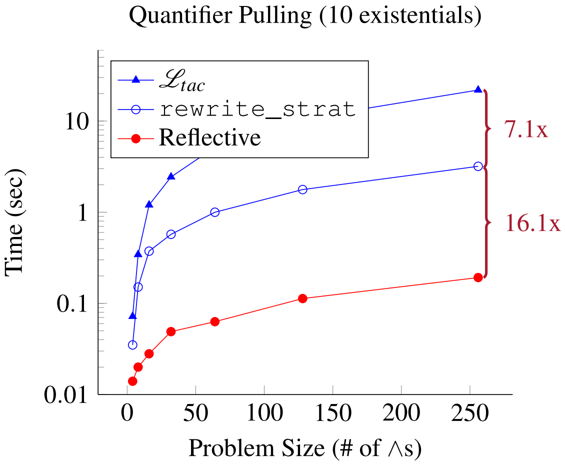


Case Study: Lifting Quantifiers w/ Rewriting

```
  P
∧ (∃ x : nat, Q x)
∧ (∃ y : nat, R y)
```

⇓

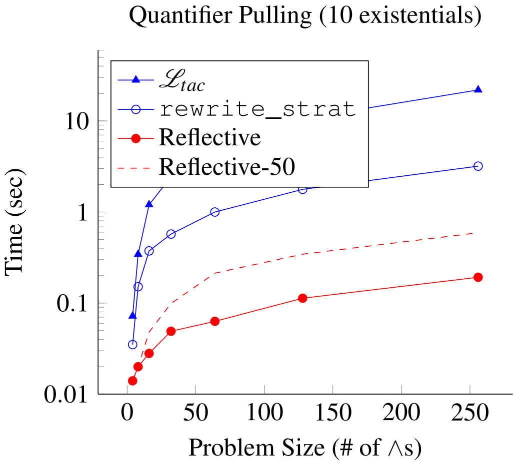
```
∃ x y : nat,
   P ∧ Q x ∧
   R y
```



Case Study: Lifting Quantifiers w/ Rewriting

```
  P
∧ (∃ x : nat, Q x)
∧ (∃ y : nat, R y)
```

```
 ∃ x y : nat,
   P ∧ Q x ∧
   R y
```



MirrorCore $= \lambda(\tau, \sigma) + \mathscr{R}_{tac}$

- Computational reflection enables scalable proofs
- MirrorCore provides generic, customizable syntax
- $\mathscr{R}_{tac}$ is a reflective tactic language
  - Backtracking proof search
  - Automatic proofs
  - Integration with custom tactics

```
github.com/gmalecha/mirror-core
$ opam install coq-mirror-core
```

“Side-by-Side” Comparison

```
Definition iter_right (n : nat) : rtac :=
  REC n (fun rec ⇒
          FIRST [ APPLY lem_plus_cancel ;;
                    ON_EACH [ APPLY lem_refl | IDTAC ]
                | APPLY lem_plus_assoc_c1 ;; ON_ALL rec
                | APPLY lem_plus_assoc_c2 ;; ON_ALL rec
                ])
    IDTAC.

Ltac iter_right :=
  first [ apply plus_cancel; [ apply refl | idtac ]
        | apply plus_assoc_c1 ; iter_right
        | apply plus_assoc_c2 ; iter_right ].
```

References I

Andrew W. Appel.
Verified software toolchain.
In *Proc. ESOP*, volume 6602 of *LNCS*, pages 1–17. Springer-Verlag, 2011.

Andrew W. Appel.
Verification of a cryptographic primitive: Sha-256, May 2014.

Samuel Boutin.
Using reflection to build efficient and certified decision procedures.
In *Proc. TACS*, 1997.

Adam Chlipala.
From network interface to multithreaded web applications: A case study in modular program verification.
POPL '15, pages 609–622, 2015.

Benjamin Delaware, Clément Pit-Claudel, Jason Gross, and Adam Chlipala.
Fiat: Deductive synthesis of abstract data types in a proof assistant.
POPL '15, pages 689–700, New York, NY, USA, 2015. ACM.

Zhong Shao.
Clean-slate development of certified os kernels.
CPP '15, pages 95–96, New York, NY, USA, 2015. ACM.