Trakt: a generic preprocessing tactic for theory-based proof automation

Denis Cousineau [1]
Enzo Crance [1,2]
Assia Mahboubi [2]

1 – Mitsubishi Electric R&D Centre Europe (MERCE)

2 – Inria Gallinette

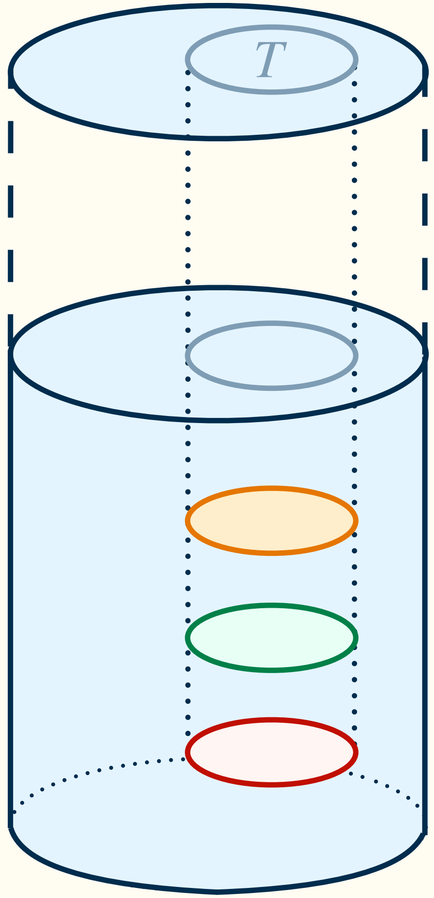
Coq Workshop 2022
Technion (Haifa, Israel)
Friday 12 August 2022

Proof automation in Coq

Decision procedures

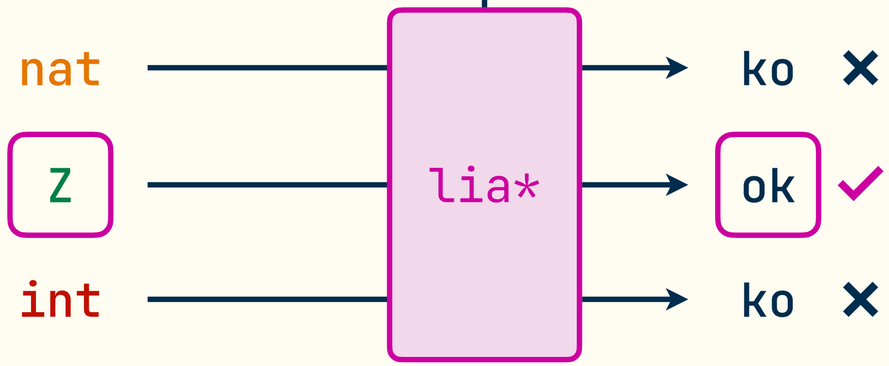
theory
(decidable)

possible encodings
in Coq



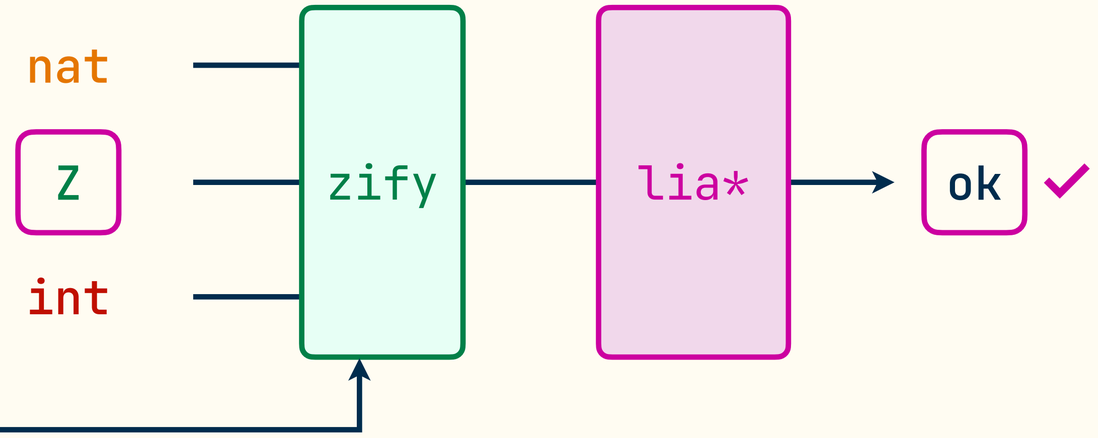
$\mathscr{D}_T$

decision procedure



implementation

Goal preprocessing



knowledge base

```
        nat ⇝ Z
Nat.add (+) ⇝ Z.add (+)
Nat.le (⩽) ⇝ Z.le (⩽)
        ... ⇝ ...
```

(typeclasses)

```
Goal forall (x : nat), x + x ⩾ x.
Proof. Fail lia*. zify. lia*. Qed.
```

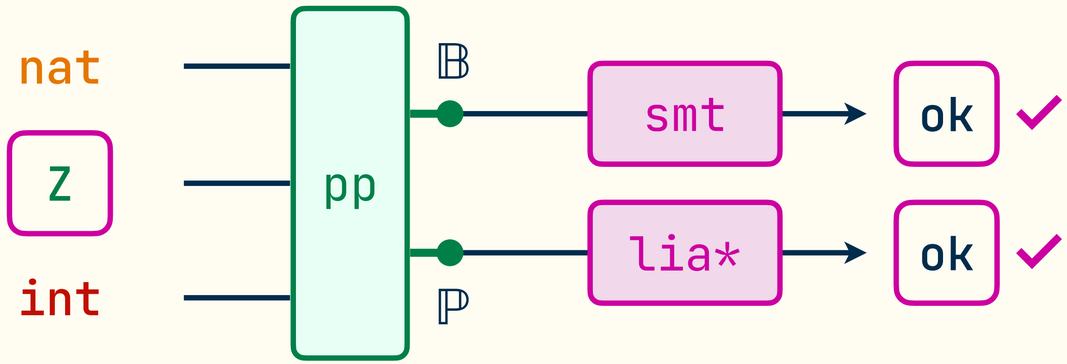
`zify` [Besson, 2017]
preprocessing tactic for `lia`

```
Goal forall (x : int), x ⩾ 0 → x + x ⩾ x.
Proof. zify. lia*. Qed.
```

`mczify` [Sakaguchi, 2021]
layer above `zify` for MathComp

Limits of existing tools

theories + logic +




```
Goal forall (f : Z → Z) (x : Z), f (2 * x) ⩽? f (x + x) = true.
Proof. smt. Qed.
```

SMTCoq [Armand et coll., 2011]
plugin to connect Coq to SMT solvers

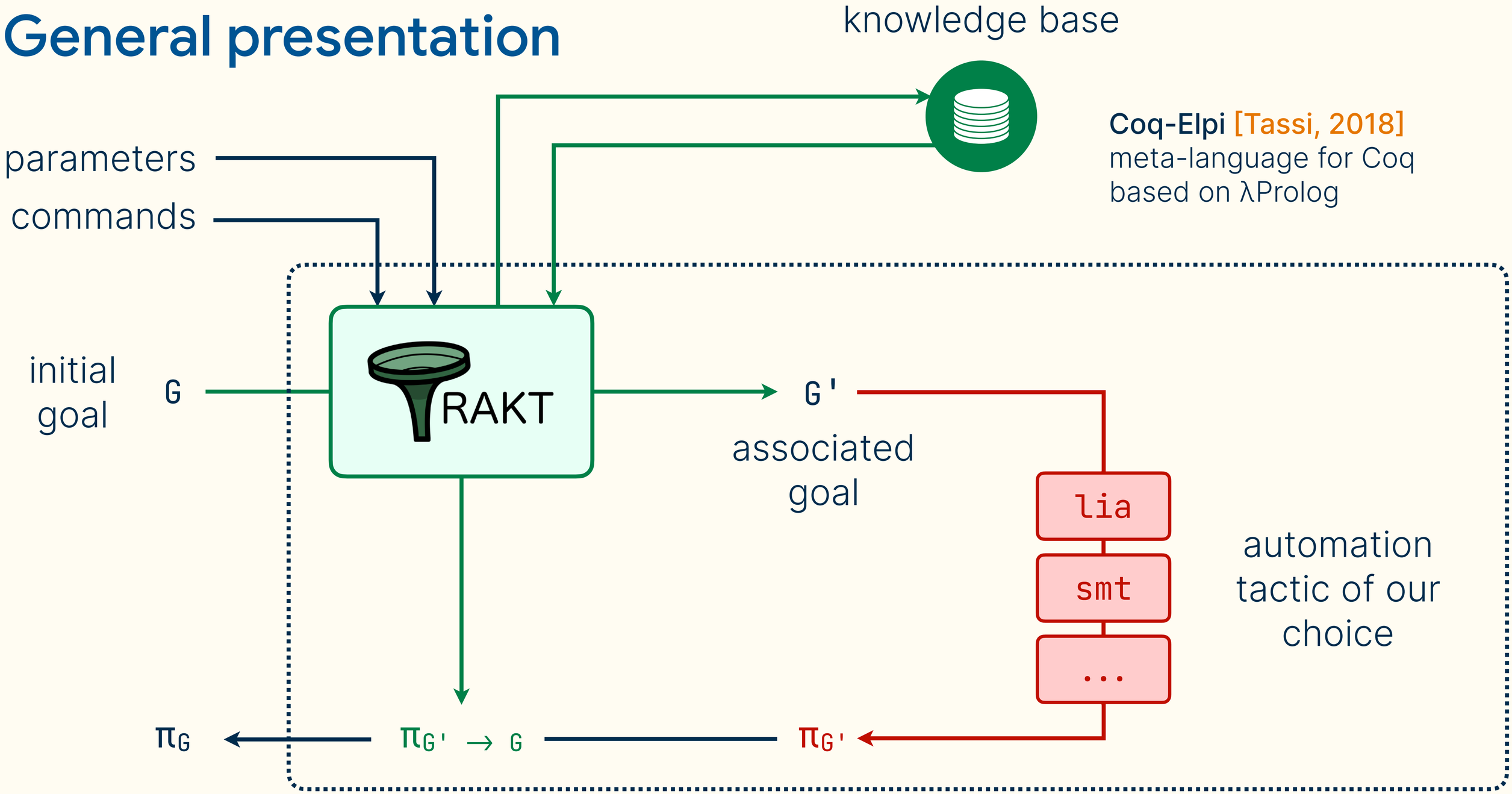
`itauto` [Besson, 2017]
SAT solver for Coq with arithmetic support

```
Goal forall (f : int → int) (x : int), f (2 * x) ⩽ f (x + x).
Proof. zify. Fail smt. Abort.
```

Trakt, a generic and extensible proxy

trakt = funnel 

General presentation



User API: filling the database

Coq

```
Trakt Add Embedding T T' f g π→ π←.
```

```
π→ : forall (x : T), g (f x) = x
π← : forall (x' : T'), f (g x') = x'
```

Elpi

```
embedding {{ T }} {{ T' }} ...
```

```
Trakt Add Embedding int Z Z_of_int Z_to_int π→ π←.
```

User API: filling the database

Coq

```
Trakt Add Embedding T T' f g π→ πc← πcond.
```

```
π→ : forall (x : T), g (f x) = x
πc← : forall (x' : T'), C x' → f (g x') = x'
Tcond := forall (x : T), C (f x)
πcond : Tcond
```

Elpi

```
embedding {{ T }} {{ T' }} ...
```

```
Trakt Add Embedding nat Z Z.of_nat Z.to_nat π→ πc← πcond.
```

User API: filling the database

Coq

```
Trakt Add Symbol S S' π.
```

```
S : T1
S' : T1'
π : ?f1 S = S'
```

Elpi

```
symbol {{ S }} {{ S' }} {{ π }} ...
```

```
Trakt Add Symbol 0i 0z π0.
```

User API: filling the database

Coq

```
Trakt Add Symbol S S' π.
```

```
S : T1 → T2 → T3
S' : T1 → T2 → T3'
π : forall x y, ?f3 (S x y) = S' (?f1 x) (?f2 y)
```

Elpi

```
symbol {{ S }} {{ S' }} {{ π }} ...
```

```
Trakt Add Symbol +i +Z π+.
```

User API: filling the database

Coq

```
Trakt Add Relation N R R' π.
```

$$\frac{\texttt{L = L' = \{\{ bool \}\}}}{\texttt{\sim = =}}$$

$R : T_1 \to T_2 \to L$
$R' : T_1' \to T_2' \to L'$
$\pi : R\ x_1\ x_2 \sim R'\ (?f_1\ x_1)\ (?f_2\ x_2)$

otherwise
$\sim = \longleftrightarrow$

Elpi

```
relation {{ R }} {{ R' }} {{ L }} {{ L' }} {{ π }} ...
```

```
Trakt Add Relation 2 =_i =_Z π_=.
```

User API: filling the database

Coq

```
Trakt Add Relation N R R' π.
```

$$\frac{\texttt{L = L' = \{\{ bool \}\}}}{\texttt{\~{} = =}}$$

$R : T_1 \to T_2 \to L$
$R' : T_1' \to T_2' \to L'$
$\pi : R\ x_1\ x_2 \sim R'\ (?f_1\ x_1)\ (?f_2\ x_2)$

otherwise
$\sim = \longleftrightarrow$

Elpi

```
relation {{ R }} {{ R' }} {{ L }} {{ L' }} {{ π }} ...
```

```
Trakt Add Relation 2 (Order.le rd int_ot) Z.leb π_⩽.
```

User API: calling the tactic

Coq
```
trakt E L.
```

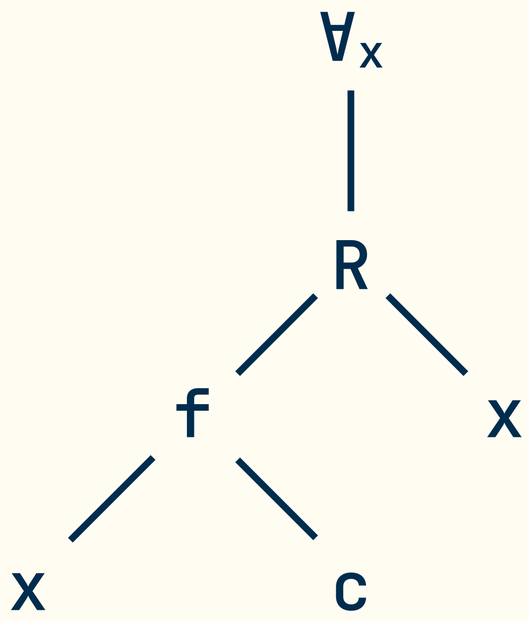
Coq
```
trakt E L with rel ...
```

E : target embedding type (optional argument)
L : target logical type (either Prop or bool)

```
Goal forall (x : int), x +i 0i =i x.
Proof.
  trakt Z Prop.          (*    forall (x' : Z),  x' +Z 0Z =Z x'    *)
  lia.
Qed.
```

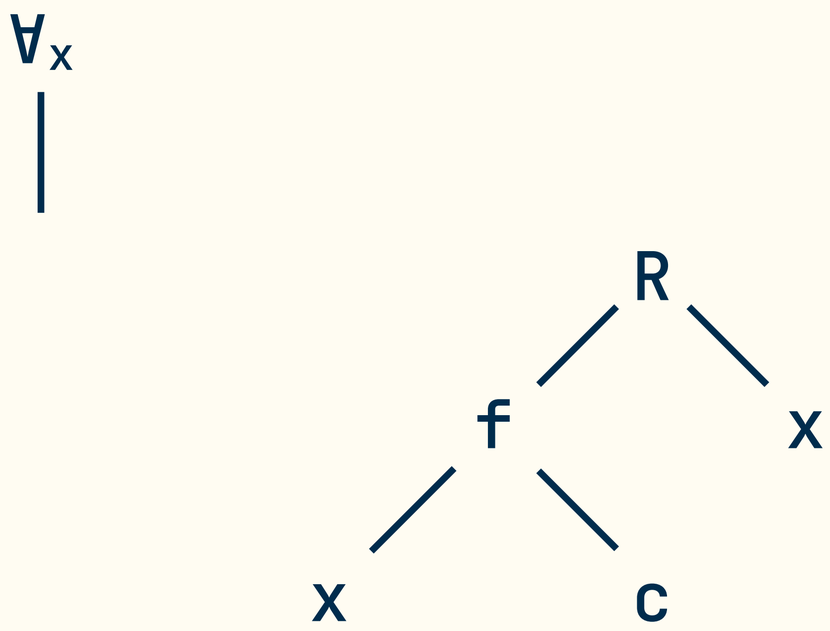
Algorithm overview

```
forall (x : int),   x +ᵢ 0ᵢ =ᵢ x
```



Algorithm overview

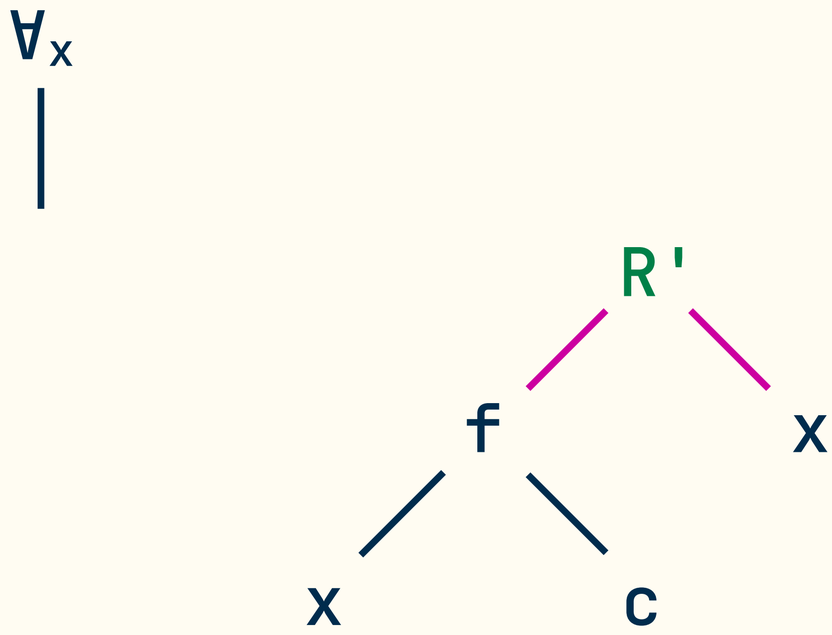
$$x : \mathbb{Z} \vdash x +_{\mathbb{Z}} 0_{\mathbb{Z}} =_{\mathbb{Z}} x$$



universal quantifier traversal

Algorithm overview

$$x : \mathbb{Z} \vdash \texttt{Z\_of\_int}\ (x +_{\mathbb{Z}} 0_{\mathbb{Z}}) =_{\mathbb{Z}} \texttt{Z\_of\_int}\ x$$

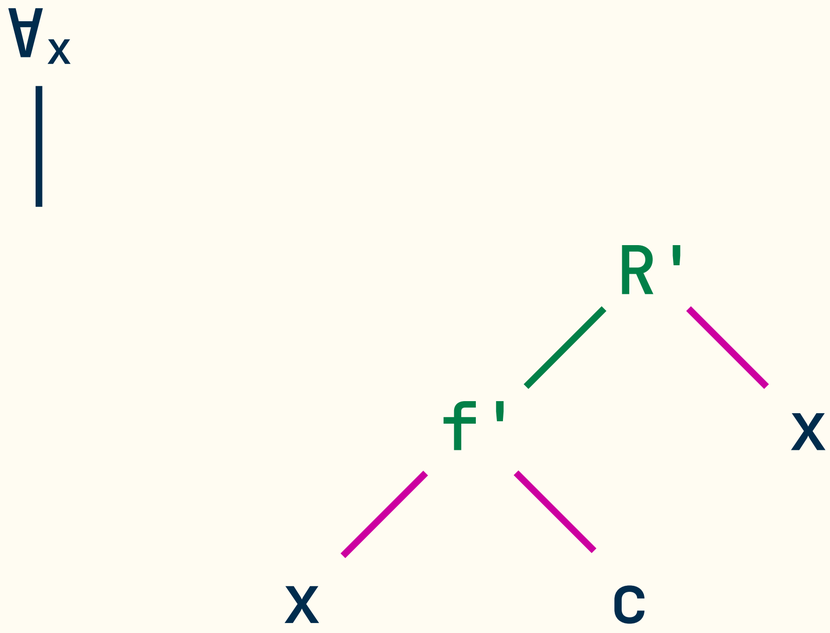


universal quantifier traversal

introduction of embeddings

Algorithm overview

$$x : \mathtt{int} \vdash \mathtt{Z\_of\_int}\ x\ \boxed{+_Z}\ \mathtt{Z\_of\_int}\ 0_i =_Z \mathtt{Z\_of\_int}\ x$$



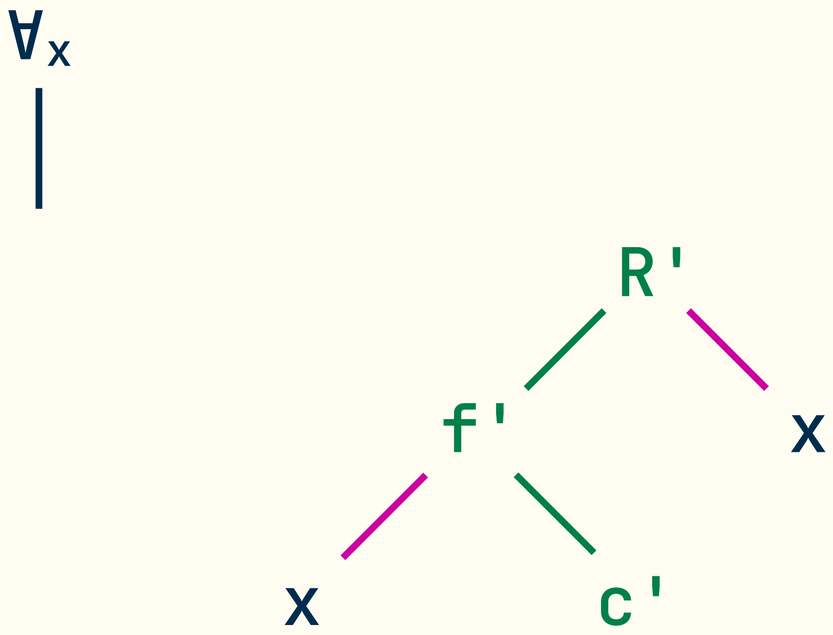
universal quantifier traversal

introduction of embeddings

embedding descent

Algorithm overview

```
x : int  ⊢  Z_of_int x +Z 0Z =Z Z_of_int x
```



universal quantifier traversal

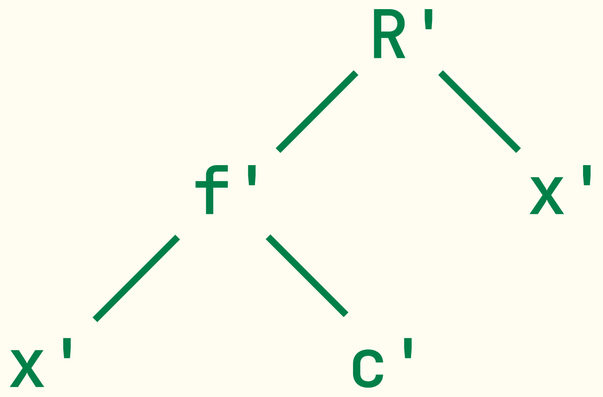
introduction of embeddings

embedding descent

constant replacement

Algorithm overview

$$x' : \mathbb{Z} \quad \vdash \quad x' +_{\mathbb{Z}} 0_{\mathbb{Z}} =_{\mathbb{Z}} x'$$



universal quantifier traversal

introduction of embeddings

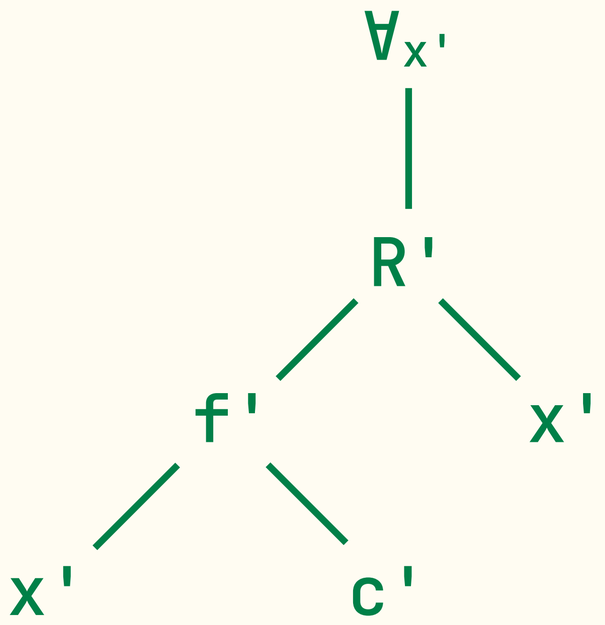
embedding descent

constant replacement

abstraction of embedded variables

Algorithm overview

```
forall (x' : Z),    x' +Z 0Z =Z x'
```



Demo

Recap and future work

Comparative table

	`zify`	`mczify`	Trakt
supported theories	arithmetic		**any theory**
symbols handled	theory-specific		theory-specific **+ uninterpreted symbols**
extensibility	typeclasses		**user-friendly commands** + parameterised tactic
flexibility	designed for `lia`	designed for `lia` + MathComp	**target-agnostic**
symbol detection	keyed matching		table-driven conversion

Meta-programming with Coq-Elpi

1. HOAS encoding of Coq terms

```
type prod name → term → (term → term) → term.
type fun name → term → (term → term) → term.
```

No De Bruijn indices!

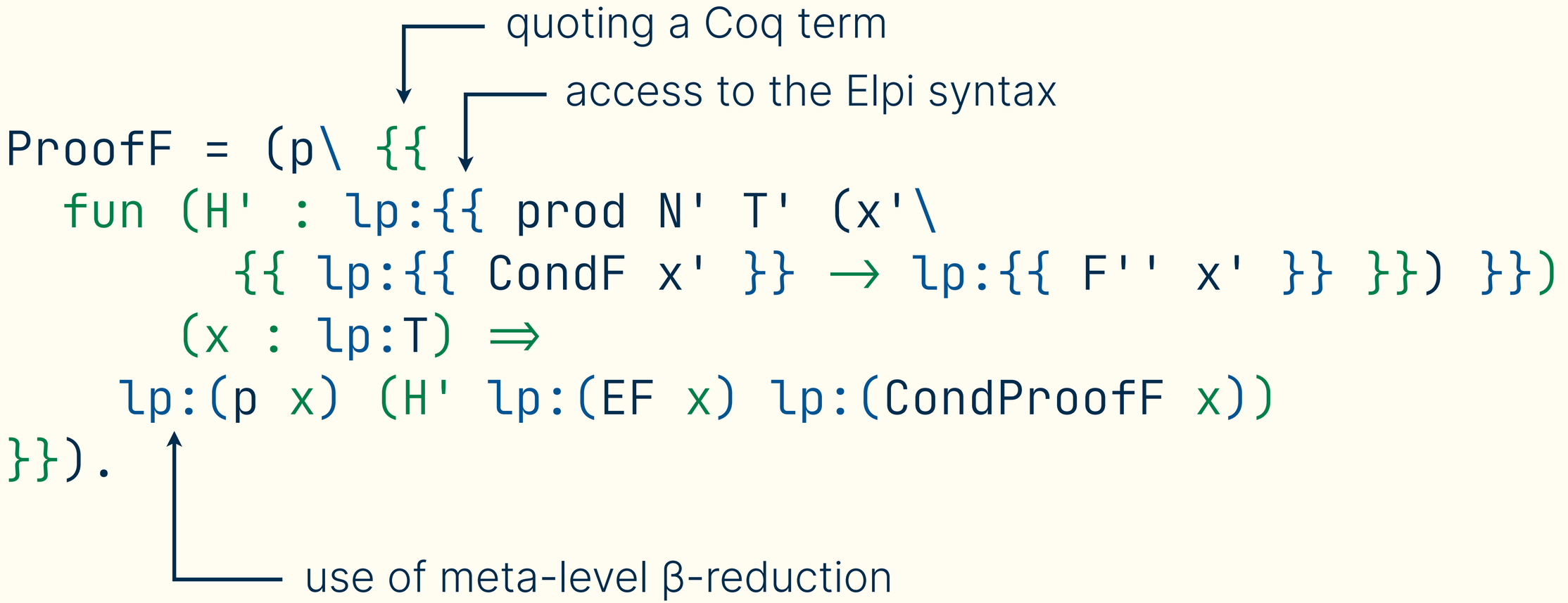
```
type-of (fun Nx Tx F) (prod Nx Tx TF) :-
  pi x\ type-of x Tx ⇒ type-of (F x) (TF x).
```

"universal constant"

```
abstract X Term (fun _ _ F) :-
  pi x\ copy X x ⇒ copy Term (F x).
```

Meta-programming with Coq-Elpi

2 Quote & unquote system



Interesting design questions

- detection of declared symbols
- API design
- feature amount
- meta-language
- preventing errors

conversion ?

kinds of symbols ?

remain general

alternatives ?

user guidance

Future work

Possible improvements for Trakt v1

- bug fixes
- more flexibility in declarations
- better error messages
- compatibility with relevant user needs

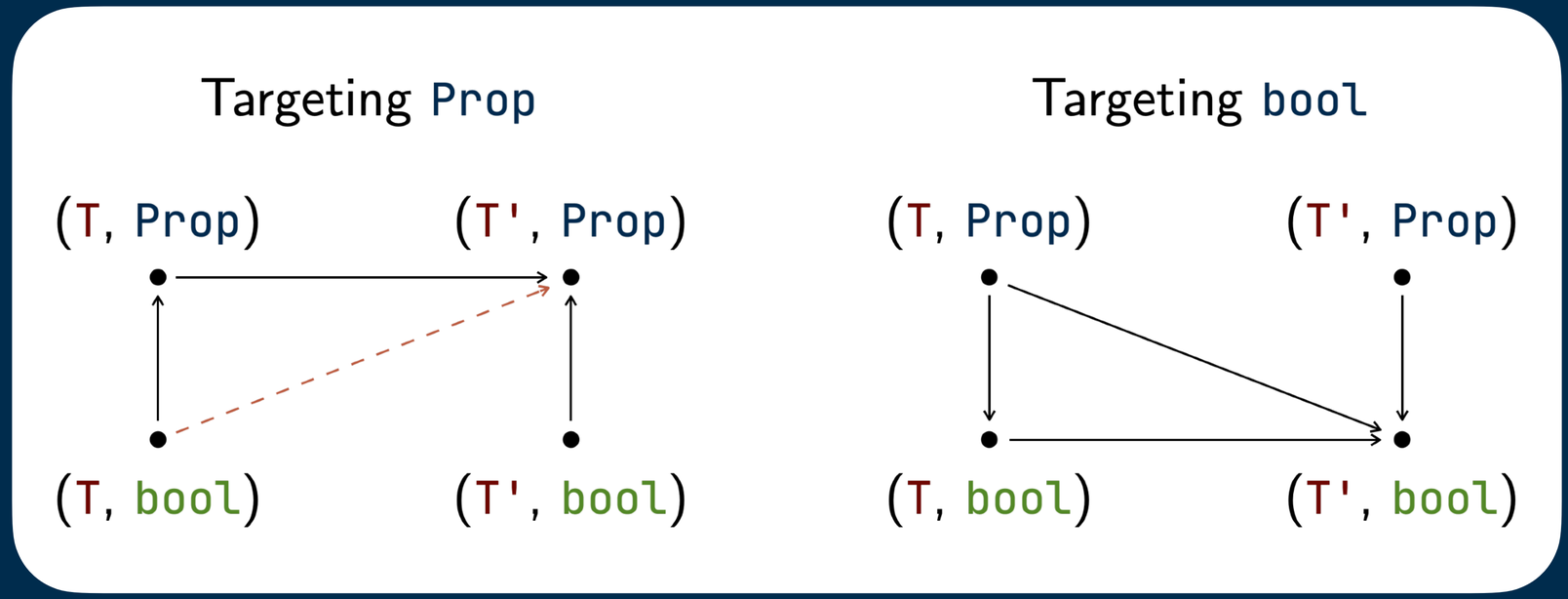
Development of Trakt v2

- second prototype of preprocessing tactic based on parametricity

Questions and answers

Does the tool preserve readability of the proof state?

Why does the translation have to be done in 2 phases when targeting `Prop`?




```
Nat.eqb ⇝ @eq Z

Definition Nat_eqb_Prop n m := Nat.eqb n m = true.

Nat.eqb ⇝ Nat_eqb_Prop ⇝ @eq Z
```

How does the plugin behave if a user declaration is missing?

Has the success of Trakt been evaluated empirically?

Do you plan to distribute Trakt along with libraries of pre-filled databases?

Try Trakt !

https://ecrancemerce.github.io/trakt/

```
opam repo add coq-released https://coq.inria.fr/opam/released
opam install coq-trakt
```