A new type-class solver for Coq in Elpi

Davide Fissore and Enrico Tassi

July 31, 2023

Coq workshop - Białystok

Elaboration in Proof Assistant

`Coq` checks a “low level” (type theory) language where all information is explicit

This language is very verbose!

How to make it practical:

1. The user enters a “high level” text (leaving many details out)
2. This text is elaborated inferring:
   - Types
   - Instances of type classes
   - ...

Quantitative Example

```
1 Definition determinant n (A : 'M_n) : R :=
2   Σ (σ : perm_of n) sgn(σ) * Π i, A i (σ i).
```

After elaboration:

```
1  fun (n : nat) (A : matrix (GRing.SemiRing.sort
2  (GRing_Ring__to__GRing_SemiRing R)) n n) =>
3    @bigop.body
4      (GRing.Nmodule.sort
5        (GRing_SemiRing__to__GRing_Nmodule
6          (GRing_Ring__to__GRing_SemiRing R)))
7        (Finite.sort
8          (perm_perm_type__canonical__fintype_Finite
9            (fintype_ordinal__canonical__fintype_Finite n)))
10       (@GRing.zero (GRing_SemiRing__to__GRing_Nmodule
11         (GRing_Ring__to__GRing_SemiRing R)))
12       (index_enum
13         (perm_perm_type__canonical__fintype_Finite
14             (fintype_ordinal__canonical__fintype_Finite n)))
15       (fun σ : @perm_of (fintype_ordinal__canonical__fintype_Finite n)
16         (Phant (ordinal n)) =>
17           @BigBody
18             (GRing.Nmodule.sort
19               (GRing_SemiRing__to__GRing_Nmodule
20                 (GRing_Ring__to__GRing_SemiRing R)))
21             (@perm_of (fintype_ordinal__canonical__fintype_Finite n)
22               (Phant (ordinal n))) σ
23                 ...)
```

Why to rework the Coq's type-class solver

Coq's TC are very popular, however the solver:

- Has too many parameters
- Still not full control on it
- Performances are not stellar
- …

Type classes (TC) and instances

Definition (Type classes)

A type class is a form of *ad hoc* polymorphism allowing to overload operators and functions[1].

```
(* The type class for the equality *)
Class Eqb (T : Type) := {
  eqb : T -> T -> bool,
  eq_leibniz : forall A B, eqb A B = true -> A = B.
}.
```

[1] *How to make ad-hoc polymorphism less ad hoc* - P. Wadler & S. Blott

Instances

Implementation of the `Eqb` type class for `unit` and `pairs`.

```
Instance eqUnit : Eqb unit := {
  eqb x y := true, eq_leibniz := ...
}.

Instance eqProd `(Eqb A, Eqb B) : Eqb (A * B) :=
  { eqb x y := eqb x.1 y.1 && eqb x.2 y.2,
    eq_leibniz := ... }.
```

```
Check (eqb (tt, tt) (tt, tt)).
```

Instances

Implementation of the `Eqb` type class for `unit` and `pairs`.

```
Instance eqUnit : Eqb unit := {
  eqb x y := true, eq_leibniz := ...
}.

Instance eqProd `(Eqb A, Eqb B) : Eqb (A * B) :=
  { eqb x y := eqb x.1 y.1 && eqb x.2 y.2,
    eq_leibniz := ... }.
```

```
Check (@eqb ?Type ?Instance (tt, tt) (tt, tt)).
```

Instances

Implementation of the `Eqb` type class for `unit` and `pairs`.

```
Instance eqUnit : Eqb unit := {
  eqb x y := true, eq_leibniz := ...
}.

Instance eqProd `(Eqb A, Eqb B) : Eqb (A * B) :=
  { eqb x y := eqb x.1 y.1 && eqb x.2 y.2,
    eq_leibniz := ... }.
```

```
Check (@eqb (unit * unit) ?Instance (tt, tt) (tt, tt)).
```

Instances

Implementation of the **Eqb** type class for **unit** and **pairs**.

```
Instance eqUnit : Eqb unit := {
  eqb x y := true, eq_leibniz := ...
}.

Instance eqProd `(Eqb A, Eqb B) : Eqb (A * B) :=
  { eqb x y := eqb x.1 y.1 && eqb x.2 y.2,
    eq_leibniz := ... }.
```

```
Check (@eqb (unit * unit) (eqProd eqUnit eqUnit)
  (tt, tt) (tt, tt)).
```

Translation in Elpi

Instance search can be done via logic programming:

```
% [tc-Eqb T Solution]
% tc-Eqb     : the pred doing the recursive search for Eqb
% T          : Type applied to eqb
% Solution   : the instance to apply on Eqb T
tc-Eqb {{unit}} {{eqUnit}}.

tc-Eqb {{lp:A * lp:B}} {{eqProd lp:PA lp:PB}} :-
  tc-Eqb A PA, tc-Eqb B PB.
```

- **Goal**: `Eqb (unit * unit) Sol`

Translation in Elpi

Instance search can be done via logic programming:

```
% [tc-Eqb T Solution]
% tc-Eqb     : the pred doing the recursive search for Eqb
% T          : Type applied to eqb
% Solution   : the instance to apply on Eqb T
tc-Eqb {{unit}} {{eqUnit}}.

tc-Eqb {{lp:A * lp:B}} {{eqProd lp:PA lp:PB}} :-
  tc-Eqb A PA, tc-Eqb B PB.
```

- **Goal**: `Eqb (unit * unit) (eqProd PA PB)`
- apply `eqProd` → `A = unit, B = unit`
  - `Eqb A PA`
  - `Eqb B PB`

Translation in Elpi

Instance search can be done via logic programming:

```
% [tc-Eqb T Solution]
% tc-Eqb     : the pred doing the recursive search for Eqb
% T          : Type applied to eqb
% Solution   : the instance to apply on Eqb T
tc-Eqb {{unit}} {{eqUnit}}.

tc-Eqb {{lp:A * lp:B}} {{eqProd lp:PA lp:PB}} :-
  tc-Eqb A PA, tc-Eqb B PB.
```

- **Goal**: `Eqb (unit * unit) (eqProd PA PB)`
- apply `eqProd` → `A = unit`, `B = unit`
  - `Eqb A PA`
  - `Eqb B PB`

Translation in Elpi

Instance search can be done via logic programming:

```
% [tc-Eqb T Solution]
% tc-Eqb     : the pred doing the recursive search for Eqb
% T          : Type applied to eqb
% Solution   : the instance to apply on Eqb T
tc-Eqb {{unit}} {{eqUnit}}.

tc-Eqb {{lp:A * lp:B}} {{eqProd lp:PA lp:PB}} :-
  tc-Eqb A PA, tc-Eqb B PB.
```

- **Goal**: `Eqb (unit * unit) (eqProd PA PB)`
- apply `eqProd` → `A = unit, B = unit`
  - `Eqb unit PA`
  - `Eqb unit PB`

Translation in Elpi

Instance search can be done via logic programming:

```
% [tc-Eqb T Solution]
% tc-Eqb     : the pred doing the recursive search for Eqb
% T          : Type applied to eqb
% Solution   : the instance to apply on Eqb T
tc-Eqb {{unit}} {{eqUnit}}.

tc-Eqb {{lp:A * lp:B}} {{eqProd lp:PA lp:PB}} :-
  tc-Eqb A PA, tc-Eqb B PB.
```

- Goal: `Eqb (unit * unit) (eqProd PA PB)`
- apply `eqProd` → `A = unit, B = unit`
  - `Eqb unit PA` → apply `eqUnit` → `PA = eqUnit`
  - `Eqb unit PB`

Translation in Elpi

Instance search can be done via logic programming:

```
% [tc-Eqb T Solution]
% tc-Eqb     : the pred doing the recursive search for Eqb
% T          : Type applied to eqb
% Solution   : the instance to apply on Eqb T
tc-Eqb {{unit}} {{eqUnit}}.

tc-Eqb {{lp:A * lp:B}} {{eqProd lp:PA lp:PB}} :-
  tc-Eqb A PA, tc-Eqb B PB.
```

- Goal: `Eqb (unit * unit) (eqProd PA PB)`
- apply `eqProd` → `A = unit, B = unit`
  - `Eqb unit PA` → apply `eqUnit` → `PA = eqUnit`
  - `Eqb unit PB`

Translation in Elpi

Instance search can be done via logic programming:

```
% [tc-Eqb T Solution]
% tc-Eqb     : the pred doing the recursive search for Eqb
% T          : Type applied to eqb
% Solution   : the instance to apply on Eqb T
tc-Eqb {{unit}} {{eqUnit}}.

tc-Eqb {{lp:A * lp:B}} {{eqProd lp:PA lp:PB}} :-
  tc-Eqb A PA, tc-Eqb B PB.
```

- **Goal**: `Eqb (unit * unit) (eqProd eqUnit PB)`
- apply `eqProd` → `A = unit, B = unit`
  - `Eqb unit eqUnit` → apply `eqUnit` → `PA = eqUnit`
  - `Eqb unit PB`

Translation in Elpi

Instance search can be done via logic programming:

```
% [tc-Eqb T Solution]
% tc-Eqb     : the pred doing the recursive search for Eqb
% T          : Type applied to eqb
% Solution   : the instance to apply on Eqb T
tc-Eqb {{unit}} {{eqUnit}}.

tc-Eqb {{lp:A * lp:B}} {{eqProd lp:PA lp:PB}} :-
  tc-Eqb A PA, tc-Eqb B PB.
```

- **Goal**: `Eqb (unit * unit) (eqProd eqUnit PB)`
- apply `eqProd` → `A = unit, B = unit`
  - `Eqb unit eqUnit` → apply `eqUnit` → `PA = eqUnit`
  - `Eqb unit PB` → apply `eqUnit` → `PB = eqUnit`

Translation in Elpi

Instance search can be done via logic programming:

```
% [tc-Eqb T Solution]
% tc-Eqb     : the pred doing the recursive search for Eqb
% T          : Type applied to eqb
% Solution   : the instance to apply on Eqb T
tc-Eqb {{unit}} {{eqUnit}}.

tc-Eqb {{lp:A * lp:B}} {{eqProd lp:PA lp:PB}} :-
  tc-Eqb A PA, tc-Eqb B PB.
```

- `Goal`: `Eqb (unit * unit) (eqProd eqUnit PB)`
- apply `eqProd` → `A = unit, B = unit`
  - `Eqb unit eqUnit` → apply `eqUnit` → `PA = eqUnit`
  - `Eqb unit PB` → apply `eqUnit` → `PB = eqUnit`

Translation in Elpi

Instance search can be done via logic programming:

```
% [tc-Eqb T Solution]
% tc-Eqb     : the pred doing the recursive search for Eqb
% T          : Type applied to eqb
% Solution   : the instance to apply on Eqb T
tc-Eqb {{unit}} {{eqUnit}}.

tc-Eqb {{lp:A * lp:B}} {{eqProd lp:PA lp:PB}} :-
  tc-Eqb A PA, tc-Eqb B PB.
```

- **Goal**: `Eqb (unit * unit) (eqProd eqUnit eqUnit)`
- apply `eqProd` → `A = unit, B = unit`
  - `Eqb unit eqUnit` → apply `eqUnit` → `PA = eqUnit`
  - `Eqb unit eqUnit` → apply `eqUnit` → `PB = eqUnit`

Translation in Elpi

Instance search can be done via logic programming:

```
% [tc-Eqb T Solution]
% tc-Eqb    : the pred doing the recursive search for Eqb
% T         : Type applied to eqb
% Solution  : the instance to apply on Eqb T
tc-Eqb {{unit}} {{eqUnit}}.

tc-Eqb {{lp:A * lp:B}} {{eqProd lp:PA lp:PB}} :-
  tc-Eqb A PA, tc-Eqb B PB.
```

- `Goal: Eqb (unit * unit) (eqProd eqUnit eqUnit)`
- apply `eqProd` → `A = unit, B = unit`
  - `Eqb unit eqUnit` → apply `eqUnit` → `PA = eqUnit`
  - `Eqb unit eqUnit` → apply `eqUnit` → `PB = eqUnit`
- `Sol = eqProd eqUnit eqUnit`

A step back: `Coq-elpi`

The `Coq-elpi` plugin allows to use `Elpi` inside `Coq`.

You can:

- Build new commands

A step back: `Coq-elpi`

The `Coq-elpi` plugin allows to use `Elpi` inside `Coq`.

You can:

- Build new commands
- Build new tactics

A step back: Coq-elpi

The Coq-elpi plugin allows to use Elpi inside Coq.

You can:

- Build new commands
- Build new tactics
- Override Coq's behavior

To override Coq's TC solver

```
Elpi Override TC My_solver [All|None|Only Class+].
```

Instance compilation in `Elpi`

`Coq`'s instance can be added in `Elpi`

- Automatically via the command `Elpi AddAllInstances`
  - Compiled as shown before, with some enhancements
  - See later...
- Manually
  - `Elpi`'s database of instances is extensible
  - The user is free to write its own rules to improve perfs

Highlights on the instance compiler

- *Deterministic* search
  - Use the logic `cut` operator
  - Forbid avoid backtracking on specified TC
- Goals reordering
  - Predicates with `Elpi` modes (input/output)
  - Compilation error of not valid instances

Change the search behavior: avoid backtracking

```
1 Class c (n : nat).
2 Local Instance c_2 : c 2 | 1.
3 Local Instance c_1 : c 1 | 10.
4 Class d (n : nat).
5 Local Instance d_1 : d 1.
6 Class e (n : nat).
7 Local Instance e_n {n} :
8   c n -> d n -> e n.
```

```
1
2 tc-c {{2}} {{c_2}}.
3 tc-c {{1}} {{c_1}}.
4
5 tc-d {{1}} {{d_1}}.
6
7 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
8   tc-c N P1, tc-d N P2.
```

```
Check (_ : e ?n)
```

- **Goal**: e N

Change the search behavior: avoid backtracking

```
1 Class c (n : nat).
2 Local Instance c_2 : c 2 | 1.
3 Local Instance c_1 : c 1 | 10.
4 Class d (n : nat).
5 Local Instance d_1 : d 1.
6 Class e (n : nat).
7 Local Instance e_n {n} :
8   c n -> d n -> e n.
```

```
1
2 tc-c {{2}} {{c_2}}.
3 tc-c {{1}} {{c_1}}.
4
5 tc-d {{1}} {{d_1}}.
6
7 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
8   tc-c N P1, tc-d N P2.
```

`Check (_ : e ?n)`

- **Goal**: `e N`
- `e N` → apply `e_n` → `Sol = e_n N P1 P2`
  - `c N P1`
  - `d N P2` .

Change the search behavior: avoid backtracking

```
1 Class c (n : nat).
2 Local Instance c_2 : c 2 | 1.
3 Local Instance c_1 : c 1 | 10.
4 Class d (n : nat).
5 Local Instance d_1 : d 1.
6 Class e (n : nat).
7 Local Instance e_n {n} :
8   c n -> d n -> e n.
```

```
1
2 tc-c {{2}} {{c_2}}.
3 tc-c {{1}} {{c_1}}.
4
5 tc-d {{1}} {{d_1}}.
6
7 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
8   tc-c N P1, tc-d N P2.
```

```
Check (_ : e ?n)
```

- **Goal**: `e N`
- `e N` → apply `e_n` → `Sol = e_n N P1 P2`
  - `c N P1` → apply `c_2` → `N = 2, P1 = c_2`
  - `d N P2` .

Change the search behavior: avoid backtracking

```
1 Class c (n : nat).
2 Local Instance c_2 : c 2 | 1.
3 Local Instance c_1 : c 1 | 10.
4 Class d (n : nat).
5 Local Instance d_1 : d 1.
6 Class e (n : nat).
7 Local Instance e_n {n} :
8   c n -> d n -> e n.
```

```
1
2 tc-c {{2}} {{c_2}}.
3 tc-c {{1}} {{c_1}}.
4
5 tc-d {{1}} {{d_1}}.
6
7 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
8   tc-c N P1, tc-d N P2.
```

```
Check (_ : e ?n)
```

- **Goal**: e N
- e N → apply e_n → Sol = e_n N P1 P2
  - c N P1 → apply c_2 → N = 2, P1 = c_2
  - d N P2 .

Change the search behavior: avoid backtracking

```
1 Class c (n : nat).
2 Local Instance c_2 : c 2 | 1.
3 Local Instance c_1 : c 1 | 10.
4 Class d (n : nat).
5 Local Instance d_1 : d 1.
6 Class e (n : nat).
7 Local Instance e_n {n} :
8   c n -> d n -> e n.
```

```
1
2 tc-c {{2}} {{c_2}}.
3 tc-c {{1}} {{c_1}}.
4
5 tc-d {{1}} {{d_1}}.
6
7 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
8   tc-c N P1, tc-d N P2.
```

`Check (_ : e ?n)`

- **Goal**: e 2
- e N → apply e_n → Sol = e_n 2 c_2 P2
  - c 2 c_2 → apply c_2 → N = 2, P1 = c_2
  - d 2 P2 .

Change the search behavior: avoid backtracking

```
1 Class c (n : nat).
2 Local Instance c_2 : c 2 | 1.
3 Local Instance c_1 : c 1 | 10.
4 Class d (n : nat).
5 Local Instance d_1 : d 1.
6 Class e (n : nat).
7 Local Instance e_n {n} :
8   c n -> d n -> e n.
```

```
1
2 tc-c {{2}} {{c_2}}.
3 tc-c {{1}} {{c_1}}.
4
5 tc-d {{1}} {{d_1}}.
6
7 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
8   tc-c N P1, tc-d N P2.
```

```
Check (_ : e ?n)
```

- **Goal**: e 2
- e N → apply e_n → Sol = e_n 2 c_2 P2
  - c 2 c_2 → apply c_2 → N = 2, P1 = c_2
  - d 2 P2 → no match → apply c_2 fails.

Change the search behavior: avoid backtracking

```
1 Class c (n : nat).
2 Local Instance c_2 : c 2 | 1.
3 Local Instance c_1 : c 1 | 10.
4 Class d (n : nat).
5 Local Instance d_1 : d 1.
6 Class e (n : nat).
7 Local Instance e_n {n} :
8   c n -> d n -> e n.
```

```
1
2 tc-c {{2}} {{c_2}}.
3 tc-c {{1}} {{c_1}}.
4
5 tc-d {{1}} {{d_1}}.
6
7 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
8   tc-c N P1, tc-d N P2.
```

`Check (_ : e ?n)`

- **Goal**: `e N`
- `e N` → apply `e_n` → `Sol = e_n N P1 P2`
  - `c N P1`
  - `d N P2`

Change the search behavior: avoid backtracking

```
1 Class c (n : nat).
2 Local Instance c_2 : c 2 | 1.
3 Local Instance c_1 : c 1 | 10.
4 Class d (n : nat).
5 Local Instance d_1 : d 1.
6 Class e (n : nat).
7 Local Instance e_n {n} :
8   c n -> d n -> e n.
```

```
1
2 tc-c {{2}} {{c_2}}.
3 tc-c {{1}} {{c_1}}.
4
5 tc-d {{1}} {{d_1}}.
6
7 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
8   tc-c N P1, tc-d N P2.
```

```
Check (_ : e ?n)
```

- **Goal**: `e N`
- `e N` → apply `e_n` → `Sol = e_n N P1 P2`
  - `c N P1` → apply `c_1` → `N = 1, P1 = c_1`
  - `d N P2`

Change the search behavior: avoid backtracking

```
1 Class c (n : nat).
2 Local Instance c_2 : c 2 | 1.
3 Local Instance c_1 : c 1 | 10.
4 Class d (n : nat).
5 Local Instance d_1 : d 1.
6 Class e (n : nat).
7 Local Instance e_n {n} :
8   c n -> d n -> e n.
```

```
1
2 tc-c {{2}} {{c_2}}.
3 tc-c {{1}} {{c_1}}.
4
5 tc-d {{1}} {{d_1}}.
6
7 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
8   tc-c N P1, tc-d N P2.
```

```
Check (_ : e ?n)
```

- **Goal**: e N
- e N → apply e_n → Sol = e_n N P1 P2
  - c N P1 → apply c_1 → N = 1, P1 = c_1
  - d N P2

Change the search behavior: avoid backtracking

```
1 Class c (n : nat).
2 Local Instance c_2 : c 2 | 1.
3 Local Instance c_1 : c 1 | 10.
4 Class d (n : nat).
5 Local Instance d_1 : d 1.
6 Class e (n : nat).
7 Local Instance e_n {n} :
8   c n -> d n -> e n.
```

```
1
2 tc-c {{2}} {{c_2}}.
3 tc-c {{1}} {{c_1}}.
4
5 tc-d {{1}} {{d_1}}.
6
7 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
8   tc-c N P1, tc-d N P2.
```

```
Check (_ : e ?n)
```

- **Goal**: e 1
- e N $\rightarrow$ apply e_n $\rightarrow$ Sol = e_n 1 c_1 P2
  - c 1 c_1 $\rightarrow$ apply c_1 $\rightarrow$ N = 1, P1 = c_1
  - d 1 P2

Change the search behavior: avoid backtracking

```
1 Class c (n : nat).
2 Local Instance c_2 : c 2 | 1.
3 Local Instance c_1 : c 1 | 10.
4 Class d (n : nat).
5 Local Instance d_1 : d 1.
6 Class e (n : nat).
7 Local Instance e_n {n} :
8   c n -> d n -> e n.
```

```
1
2 tc-c {{2}} {{c_2}}.
3 tc-c {{1}} {{c_1}}.
4
5 tc-d {{1}} {{d_1}}.
6
7 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
8   tc-c N P1, tc-d N P2.
```

```
Check (_ : e ?n)
```

- **Goal**: e 1
- e N → apply e_n → Sol = e_n 1 c_1 P2
  - c 1 c_1 → apply c_1 → N = 1, P1 = c_1
  - d 1 P2 → apply d_1 → P2 = d_1

Change the search behavior: avoid backtracking

```
1 Class c (n : nat).
2 Local Instance c_2 : c 2 | 1.
3 Local Instance c_1 : c 1 | 10.
4 Class d (n : nat).
5 Local Instance d_1 : d 1.
6 Class e (n : nat).
7 Local Instance e_n {n} :
8   c n -> d n -> e n.
```

```
1
2 tc-c {{2}} {{c_2}}.
3 tc-c {{1}} {{c_1}}.
4
5 tc-d {{1}} {{d_1}}.
6
7 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
8   tc-c N P1, tc-d N P2.
```

```
Check (_ : e ?n)
```

- **Goal**: e 1
- e N → apply e_n → Sol = e_n 1 c_1 P2
  - c 1 c_1 → apply c_1 → N = 1, P1 = c_1
  - d 1 P2 → apply d_1 → P2 = d_1

Change the search behavior: avoid backtracking

```
1 Class c (n : nat).
2 Local Instance c_2 : c 2 | 1.
3 Local Instance c_1 : c 1 | 10.
4 Class d (n : nat).
5 Local Instance d_1 : d 1.
6 Class e (n : nat).
7 Local Instance e_n {n} :
8   c n -> d n -> e n.
```

```
1
2 tc-c {{2}} {{c_2}}.
3 tc-c {{1}} {{c_1}}.
4
5 tc-d {{1}} {{d_1}}.
6
7 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
8   tc-c N P1, tc-d N P2.
```

`Check (_ : e ?n)`

- **Goal**: `e 1`
- `e N` → apply `e_n` → `Sol = e_n 1 c_1 d_1`
  - `c 1 c_1` → apply `c_1` → `N = 1, P1 = c_1`
  - `d 1 d_1` → apply `d_1` → `P2 = d_1`

Change the search behavior: avoid backtracking

```
1 Class c (n : nat).
2 Local Instance c_2 : c 2 | 1.
3 Local Instance c_1 : c 1 | 10.
4 Class d (n : nat).
5 Local Instance d_1 : d 1.
6 Class e (n : nat).
7 Local Instance e_n {n} :
8   c n -> d n -> e n.
```

```
1
2 tc-c {{2}} {{c_2}}.
3 tc-c {{1}} {{c_1}}.
4
5 tc-d {{1}} {{d_1}}.
6
7 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
8   tc-c N P1, tc-d N P2.
```

Check `(_ : e ?n)` → `e_n c_1 d_1 : e 1`

- **Goal**: `e 1`
- `e N` → apply `e_n` → `Sol = e_n 1 c_1 d_1`
  - `c 1 c_1` → apply `c_1` → `N = 1, P1 = c_1`
  - `d 1 d_1` → apply `d_1` → `P2 = d_1`

Change the search behavior: avoid backtracking

```
1
2
3 #[det] Class c (n : nat).
4 Local Instance c_2 : c 2 | 1.
5 Local Instance c_1 : c 1 | 10.
6 Class d (n : nat).
7 Local Instance d_1 : d 1.
8 Class e (n : nat).
9 Local Instance e_n {n} :
10   c n -> d n -> e n.
```

```
1 pred do-once i:prop.
2 do-once A :- A, !.
3
4 tc-c {{2}} {{c_2}}.
5 tc-c {{1}} {{c_1}}.
6
7 tc-d {{1}} {{d_1}}.
8
9 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
10   do-once (tc-c N P1), tc-d N P2.
```

```
Check (_ : e ?n)
```

- **Goal**: e N

Change the search behavior: avoid backtracking

```
1
2
3 #[det] Class c (n : nat).
4 Local Instance c_2 : c 2 | 1.
5 Local Instance c_1 : c 1 | 10.
6 Class d (n : nat).
7 Local Instance d_1 : d 1.
8 Class e (n : nat).
9 Local Instance e_n {n} :
10   c n -> d n -> e n.
```

```
1 pred do-once i:prop.
2 do-once A :- A, !.
3
4 tc-c {{2}} {{c_2}}.
5 tc-c {{1}} {{c_1}}.
6
7 tc-d {{1}} {{d_1}}.
8
9 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
10   do-once (tc-c N P1), tc-d N P2.
```

`Check (_ : e ?n)`

- **Goal**: `e N`
- `e N` → apply `e_n` → `Sol = e_n N P1 P2`
  - `c N P1`
  - `d N P2`

Change the search behavior: avoid backtracking

```
1
2
3 #[det] Class c (n : nat).
4 Local Instance c_2 : c 2 | 1.
5 Local Instance c_1 : c 1 | 10.
6 Class d (n : nat).
7 Local Instance d_1 : d 1.
8 Class e (n : nat).
9 Local Instance e_n {n} :
10   c n -> d n -> e n.
```

```
1 pred do-once i:prop.
2 do-once A :- A, !.
3
4 tc-c {{2}} {{c_2}}.
5 tc-c {{1}} {{c_1}}.
6
7 tc-d {{1}} {{d_1}}.
8
9 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
10   do-once (tc-c N P1), tc-d N P2.
```

`Check (_ : e ?n)`

- **Goal**: e N
- e N → apply e_n → Sol = e_n N P1 P2
  - c N P1 → apply c_2 → N = 2, P1 = c_2 /* choice point */
  - d N P2 .

Change the search behavior: avoid backtracking

```
1
2
3 #[det] Class c (n : nat).
4 Local Instance c_2 : c 2 | 1.
5 Local Instance c_1 : c 1 | 10.
6 Class d (n : nat).
7 Local Instance d_1 : d 1.
8 Class e (n : nat).
9 Local Instance e_n {n} :
10   c n -> d n -> e n.
```

```
1 pred do-once i:prop.
2 do-once A :- A, !.
3
4 tc-c {{2}} {{c_2}}.
5 tc-c {{1}} {{c_1}}.
6
7 tc-d {{1}} {{d_1}}.
8
9 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
10   do-once (tc-c N P1), tc-d N P2.
```

```
Check (_ : e ?n)
```

- `Goal`: `e N`
- `e N` → apply `e_n` → `Sol = e_n N P1 P2`
  - `c N P1` → apply `c_2` → `N = 2, P1 = c_2` */* choice point */*
  - `d N P2` .

Change the search behavior: avoid backtracking

```
1
2
3 #[det] Class c (n : nat).
4 Local Instance c_2 : c 2 | 1.
5 Local Instance c_1 : c 1 | 10.
6 Class d (n : nat).
7 Local Instance d_1 : d 1.
8 Class e (n : nat).
9 Local Instance e_n {n} :
10   c n -> d n -> e n.
```

```
1 pred do-once i:prop.
2 do-once A :- A, !.
3
4 tc-c {{2}} {{c_2}}.
5 tc-c {{1}} {{c_1}}.
6
7 tc-d {{1}} {{d_1}}.
8
9 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
10   do-once (tc-c N P1), tc-d N P2.
```

```
Check (_ : e ?n)
```

- **Goal**: `e 2`
- `e N` → apply `e_n` → `Sol = e_n 2 c_1 P2`
  - `c 2 c_2` → apply `c_2` → `N = 2, P1 = c_2` */* choice point */*
  - `d 2 P2` .

Change the search behavior: avoid backtracking

```
1
2
3 #[det] Class c (n : nat).
4 Local Instance c_2 : c 2 | 1.
5 Local Instance c_1 : c 1 | 10.
6 Class d (n : nat).
7 Local Instance d_1 : d 1.
8 Class e (n : nat).
9 Local Instance e_n {n} :
10   c n -> d n -> e n.
```

```
1 pred do-once i:prop.
2 do-once A :- A, !.
3
4 tc-c {{2}} {{c_2}}.
5 tc-c {{1}} {{c_1}}.
6
7 tc-d {{1}} {{d_1}}.
8
9 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
10   do-once (tc-c N P1), tc-d N P2.
```

```
Check (_ : e ?n)
```

- **Goal**: `e 2`
- `e N` → apply `e_n` → `Sol = e_n 2 c_1 P2`
    - `c 2 c_2` → apply `c_2` → `N = 2, P1 = c_2` */* choice point */*
    - `d 2 P2` → no match → apply `c_2` fails.

Change the search behavior: avoid backtracking

```
1
2
3 #[det] Class c (n : nat).
4 Local Instance c_2 : c 2 | 1.
5 Local Instance c_1 : c 1 | 10.
6 Class d (n : nat).
7 Local Instance d_1 : d 1.
8 Class e (n : nat).
9 Local Instance e_n {n} :
10   c n -> d n -> e n.
```

```
1 pred do-once i:prop.
2 do-once A :- A, !.
3
4 tc-c {{2}} {{c_2}}.
5 tc-c {{1}} {{c_1}}.
6
7 tc-d {{1}} {{d_1}}.
8
9 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
10   do-once (tc-c N P1), tc-d N P2.
```

`Check (_ : e ?n)`

- **Goal**: `e N`
- `e N` → apply `e_n` → `Sol = e_n N P1 P2`
  - `c N P1`
  - `d N P2`

Change the search behavior: avoid backtracking

```
1
2
3 #[det] Class c (n : nat).
4 Local Instance c_2 : c 2 | 1.
5 Local Instance c_1 : c 1 | 10.
6 Class d (n : nat).
7 Local Instance d_1 : d 1.
8 Class e (n : nat).
9 Local Instance e_n {n} :
10   c n -> d n -> e n.
```

```
1 pred do-once i:prop.
2 do-once A :- A, !.
3
4 tc-c {{2}} {{c_2}}.
5 tc-c {{1}} {{c_1}}.
6
7 tc-d {{1}} {{d_1}}.
8
9 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
10   do-once (tc-c N P1), tc-d N P2.
```

```
Check (_ : e ?n)
```

- Goal: `e N`
- `e N` → apply `e_n` → `Sol = e_n N P1 P2`
  - `c N P1` → `c_1` not applicable due to cut!
  - `d N P2`

Change the search behavior: avoid backtracking

```
1
2
3 #[det] Class c (n : nat).
4 Local Instance c_2 : c 2 | 1.
5 Local Instance c_1 : c 1 | 10.
6 Class d (n : nat).
7 Local Instance d_1 : d 1.
8 Class e (n : nat).
9 Local Instance e_n {n} :
10   c n -> d n -> e n.
```

```
1 pred do-once i:prop.
2 do-once A :- A, !.
3
4 tc-c {{2}} {{c_2}}.
5 tc-c {{1}} {{c_1}}.
6
7 tc-d {{1}} {{d_1}}.
8
9 tc-e N {{e_n lp:N lp:P1 lp:P2}} :-
10   do-once (tc-c N P1), tc-d N P2.
```

`Check (_ : e ?n)` → `?e : e ?n`

- **Goal**: `e N`
- `e N` → apply `e_n` → `Sol = e_n N P1 P2`
  - `c N P1` → `c_1` not applicable due to cut!
  - `d N P2`
- Fail

TC examples for mode menagement

- For every new TC, we create a new elpi predicate:

```
Class c1 (T1 : Type) (T2 : Type).
Class c2 (T1 : Type).
Class c3 (T1 : Type).
Global Hint Mode c1 + - : typeclass_instances.
Global Hint Mode c2 + : typeclass_instances.
Global Hint Mode c3 + : typeclass_instances.
```

Goals reordering (1)

- `Instance Inst3 T1 T2 `(c2 T2, c1 T1 T2) : c3 T1.`
- Recall: `(Mode c1 + -)` and `(Mode c2 +)`

`Check (_ : c3 nat)`

```
1 tc-c3 T1 Sol :-
2   tc-c2 T2 PC2,
3   tc-c1 T1 T2 PC1,
4   Sol = {{Inst3 lp:T1 lp:T2 lp:PC2 lp:PC1}}.
```

- `Goal c3 nat`

Goals reordering (1)

- `Instance Inst3 T1 T2 `(c2 T2, c1 T1 T2) : c3 T1.`
- Recall: `(Mode c1 + -)` and `(Mode c2 +)`

`Check (_ : c3 nat)`

```
1 tc-c3 T1 Sol :-
2   tc-c2 T2 PC2,
3   tc-c1 T1 T2 PC1,
4   Sol = {{Inst3 lp:T1 lp:T2 lp:PC2 lp:PC1}}.
```

- `Goal c3 nat`
- `c3 nat` → apply `Inst3` → `T1 = nat`
  - `c2 T2 PC2`
  - `c1 T1 T2 PC1`

Goals reordering (1)

- `Instance Inst3 T1 T2 `(c2 T2, c1 T1 T2) : c3 T1.`
- Recall: `(Mode c1 + -)` and `(Mode c2 +)`

  `Check (_ : c3 nat)`

```
1 tc-c3 T1 Sol :-
2   tc-c2 T2 PC2,
3   tc-c1 T1 T2 PC1,
4   Sol = {{Inst3 lp:T1 lp:T2 lp:PC2 lp:PC1}}.
```

- `Goal c3 nat`
- `c3 nat` → apply `Inst3` → `T1 = nat`
  - `c2 T2 PC2`
  - `c1 nat T2 PC1`

Goals reordering (1)

- `Instance Inst3 T1 T2 `(c2 T2, c1 T1 T2) : c3 T1.`
- Recall: `(Mode c1 + -)` and `(Mode c2 +)`

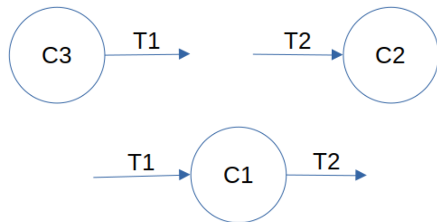
`Check (_ : c3 nat)`

```
tc-c3 T1 Sol :-
  tc-c2 T2 PC2,
  tc-c1 T1 T2 PC1,
  Sol = {{Inst3 lp:T1 lp:T2 lp:PC2 lp:PC1}}.
```

- `Goal c3 nat`
- `c3 nat` → apply `Inst3` → `T1 = nat`
  - `c2 T2 PC2` → No solution, `T2` is flexible
  - `c1 nat T2 PC1`

Goals reordering (1)

- `Instance Inst3 T1 T2 `(c2 T2, c1 T1 T2) : c3 T1.`
- Recall: `(Mode c1 + -)` and `(Mode c2 +)`
- We present the goals in a graphical way

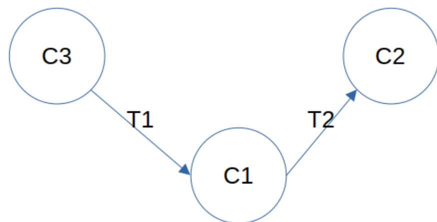


(a) Inst3

```
1 tc-c3 T1 Sol :-
2   tc-c2 T2 PC2,
3   tc-c1 T1 T2 PC1,
4   Sol = {{Inst3 lp:T1 lp:T2
  ↪   lp:PC2 lp:PC1}}.
```

Goals reordering (1)

- `Instance Inst3 T1 T2 `(c2 T2, c1 T1 T2) : c3 T1.`
- Recall: `(Mode c1 + -)` and `(Mode c2 +)`
- In `Elpi`, we do a topological sort on premises according to modes



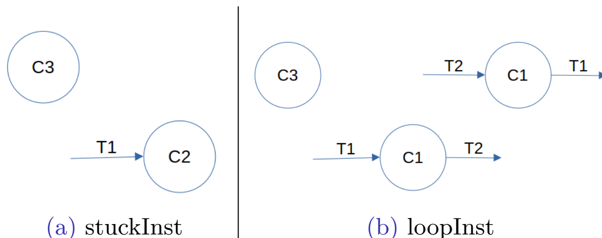
(a) Inst3

```
1 tc-c3 T1 Sol :-
2   tc-c1 T1 T2 PC1,
3   tc-c2 T2 PC2,
4   Sol = {{Inst3 lp:T1 lp:T2
 ↪    lp:PC2 lp:PC1}}.
```

Goals reordering (2)

- `Instance stuckInst T1 ` + "`" + `(c2 T1): c3 bool.`
- `Instance loopInst T1 T2 ` + "`" + `(c1 T1 T2, c1 T2 T1): c3 bool.`
- Recall: `(Mode c1 + -)` and `(Mode c2 +)`

- In `Elpi`, **compilation error** of both instances:
  - in `stuckInst`, `T1` is unknown
  - in `loopInst`, we have cyclic dependencies

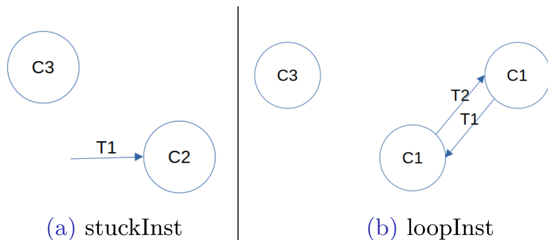


(a) stuckInst (b) loopInst

Goals reordering (2)

- `Instance stuckInst T1 `(c2 T1): c3 bool.`
- `Instance loopInst T1 T2 `(c1 T1 T2, c1 T2 T1): c3 bool.`
- Recall: `(Mode c1 + -)` and `(Mode c2 +)`

- In `Elpi`, **compilation error** of both instances:
  - in `stuckInst`, `T1` is unknown
  - in `loopInst`, we have cyclic dependencies

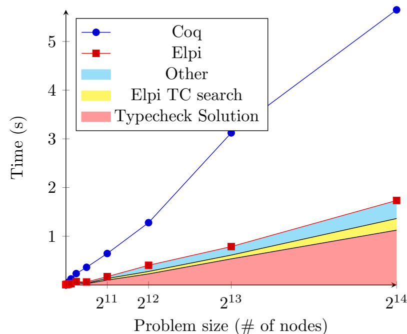


(a) stuckInst (b) loopInst

Handwritten rules

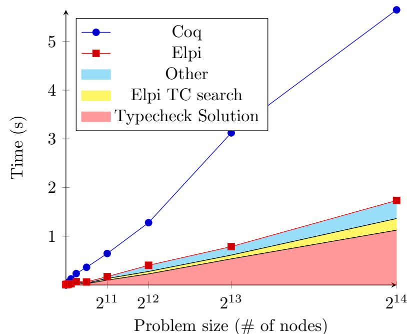
Small demo!

Benchmark

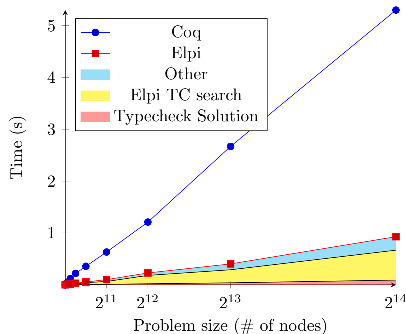


(a) Instance search classic rules

Benchmark



(a) Instance search classic rules



(b) Instance search ad-hoc rules

Ongoing and future work

- Check overlapping clauses if a TC is deterministic
  - `eg.:` If `c` is deterministic and we have
    ```
    tc-c {{1}} {{c_1}}.
    tc-c {{1}} {{c_1'}}.
    ```
    Then the instance `c_1'` can safely removed
- Add new search strategies *memoisation (tabled search)*
- Improve indexing algorithm
- ...

The `Coq-elpi` TC solver

- Deterministic instance search
- New goal sorting algorithm
- Customizable rules for your library

Thanks!

Problems

`Elpi` does a pattern fragment unification between the *goal* to solve and the *clauses* (head) in the DB.

Examples of problems:

- Different names for same function:

  `PeanoNat.Add.succ` $\leftrightarrow$ `S`

- Unification of terms $\rightarrow$ up to δ:

  $$(g \circ f) \leftrightarrow (\lambda x \Rightarrow g\ (f\ x))$$

- Pattern fragment: `Elpi` vs `Coq` term representation

  `(X a) vs app [X, a]`

Possible solutions

- Replace the syntactic match of terms with `Coq`'s unification and then solve the problem
  Drawback: expensive and maybe too smart
- Add rules by hand to solve the unification problem
  Drawback: we need to treat each case separately