Elpi, the extension language for your ITP

Enrico Tassi
Deducteam seminars - 2020

This talk is about Elpi, that is...

- An extension language
  - its interpreter comes as a library
  - with an API/FFI to write glue code
- A very high level, domain specific, language
  - Data with binders
  - Data with unification variables
- LGPL, by C.Sacerdoti Coen and myself

Elpi = λProlog + CHR

Outline

- Elpi 101
  - λProlog 101: type checker for $\lambda_{\rightarrow}$
  - λProlog + CHR 101: even & odd
- POC: Deducti + Elpi
- Example of a Coq-Elpi based tool

λProlog 101

$$
\begin{aligned}
e &= x \\
&\mid e_1 e_2 \\
&\mid \lambda x.e
\end{aligned}
$$

$$
\begin{aligned}
\tau &= C \\
&\mid \tau \rightarrow \tau
\end{aligned}
$$

```
% HOAS of terms

type app term → term → term.

type lam (term → term) → term.

% HOAS of types

type arrow ty → ty → ty.
```

```
% Example: identity function
lam (x\ x)
% Example: fst
lam x\ lam y\ x
```

λProlog 101

```
pred of i:term, o:ty.
```

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau \rightarrow \tau' \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1\ e_2 : \tau'}$$

```
of (app H A) T :-
  of H (arrow S T), of A S.
```

$$\frac{\Gamma, x : \tau \vdash e : \tau'}{\Gamma \vdash \lambda x.e : \tau \rightarrow \tau'}$$

```
of (lam F) (arrow S T) :-
  pi x\ of x S => of (F x) T.
```

% Convention
X % universally quantified around the rule
X_i % not quantified (existentially quantified, globally)

λProlog 101

$$\vdash \lambda x.\lambda y.x\ y : Q$$

Goal

```
of (lam x\ lam y\ app x y) Q0.
```

Program

```
of (app H A) T :- of H (arrow S T), of A S.
of (lam F) (arrow S T) :-
  pi x\ of x S => of (F x) T.
```

Assignments

$Q_0 = \dots$

λProlog 101

$$\vdash \lambda x.\lambda y.x\ y : Q$$

Goal

of ((x\ lam y\ app x y) c_1) T_1.

Program

of (app H A) T :- of H (arrow S T), of A S.
of (lam F) (arrow S T) :-
  pi x\ of x S => of (F x) T.
of c_1 S_1.

Assignments

Q_0 = arrow S_1 T_1
F_1 = (x\ lam y\ app x y)

λProlog 101

$$\vdash \lambda x.\lambda y.x\ y : Q$$

Goal

```
of (lam y\ app c1 y) T1.
```

Program

```
of (app H A) T :- of H (arrow S T), of A S.
of (lam F) (arrow S T) :-
  pi x\ of x S => of (F x) T.
of c1 S1.
```

Assignments

Q_0 = arrow S_1 T_1

F_1 = (x\ lam y\ app x y)

λProlog 101

$$\vdash \lambda x.\lambda y.x\ y : Q$$

Goal

of ((y\ app c_1 y) c_2) T_2.

Program

of (app H A) T :- of H (arrow S T), of A S.
of (lam F) (arrow S T) :-
  pi x\ of x S => of (F x) T.
of c_1 S_1.
of c_2 S_2.

Assignments

Q_0 = arrow S_1 (arrow S_2 T_2)
F_1 = (x\ lam y\ app x y)
F_2 = (y\ app c_1 y)

λProlog 101

$$\vdash \lambda x.\lambda y.x\ y : Q$$

Goal

of (app c_1 c_2) T_2.

Program

of (app H A) T :- of H (arrow S T), of A S.
of (lam F) (arrow S T) :-
  pi x\ of x S => of (F x) T.
of c_1 S_1.
of c_2 S_2.

Assignments

Q_0 = arrow S_1 (arrow S_2 T_2)
F_1 = (x\ lam y\ app x y)
F_2 = (y\ app c_1 y)

λProlog 101

$$\vdash \lambda x.\lambda y.x\ y : Q$$

Goal

of c_1 (arrow S_3 T_2).
of c_2 S_3.

Program

of (app H A) T :- of H (arrow S T), of A S.
of (lam F) (arrow S T) :-
  pi x\ of x S => of (F x) T.
of c_1 S_1.
of c_2 S_2.

Assignments

Q_0 = arrow S_1 (arrow S_2 T_2)
F_1 = (x\ lam y\ app x y)
F_2 = (y\ app c_1 y)
H_3 = c_1
A_3 = c_2

λProlog 101

$$\vdash \lambda x.\lambda y.x\ y : Q$$

Goal

of c_2 S_3.

Program

of (app H A) T :- of H (arrow S T), of A S.
of (lam F) (arrow S T) :-
  pi x\ of x S => of (F x) T.
of c_1 (arrow S_3 T_2).
of c_2 S_2.

Assignments

Q_0 = arrow (arrow S_3 T_2) (arrow S_2 T_2)
F_1 = (x\ lam y\ app x y)
F_2 = (y\ app c_1 y)
H_3 = c_1 S_1 = (arrow S_3 T_2)
A_3 = c_2

λProlog 101

$$\vdash \lambda x.\lambda y.x\ y : (S \to T) \to S \to T$$

Goal

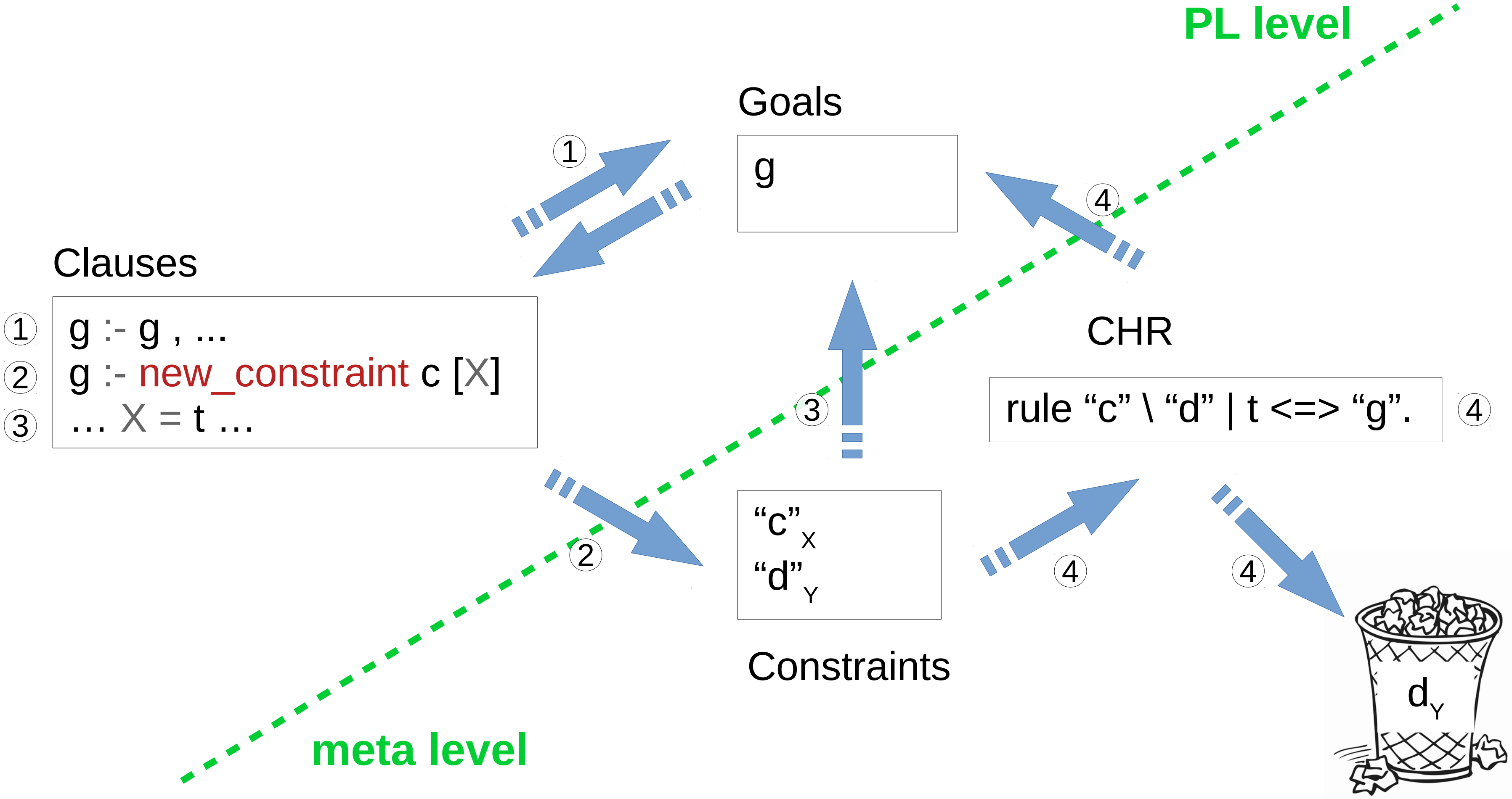
Program

of (app H A) T :- of H (arrow S T), of A S.
of (lam F) (arrow S T) :-
  pi x\ of x S => of (F x) T.
of c_1 (arrow S_2 T_2).
of c_2 S_2.

Assignments

Q_0 = arrow (arrow S_2 T_2) (arrow S_2 T_2)
F_1 = (x\ lam y\ app x y)
F_2 = (y\ app c_1 y)
H_3 = c_1 S_1 = (arrow S_3 T_2)
A_3 = c_2 S_3 = S_2

λProlog + CHR 101



PL level

Goals

g

Clauses

① g :- g , ...
② g :- new_constraint c [X]
③ … X = t …

CHR

rule “c” \ “d” | t <=> “g”. ④

“c”$_X$
“d”$_Y$

Constraints

d$_Y$

① ② ③ ④ ④ ④ ④

meta level

λProlog + CHR 101

```
type zero nat.  type succ nat -> nat.

pred odd i:nat.  pred even i:nat.  pred double i:nat, o:nat.

even zero.
odd (succ X) :- even X.
even (succ X) :- odd X.

even X :- var X, new_constraint (even X) [X].
odd X :- var X, new_constraint (odd X) [X].

double zero zero.
double (succ X) (succ (succ Y)) :- double X Y.

double X Y :- var X, new_constraint (double X Y) [X].

constraint even odd double {
  rule (even X) (odd X) <=> fail.
  rule (double _ X) <=> (even X).
}
```

λProlog + CHR 101

even X, X = succ Y, not (double Z Y)

Goals

```
even X
X = succ Y
not (double Z Y)
```

Program

```
even zero.
odd (succ X) :- even X.
even (succ X) :- odd X.
even X :- var X, new_constraint (even X) [X].
odd X :- var X, new_constraint (odd X) [X].
double zero zero.
double (succ X) (succ (succ Y)) :- double X Y.
double X Y :- var X, new_constraint (double X Y) [X].
```

Constraint store

Rules

```
(even X) (odd X) <=> fail.
(double _ X) <=> (even X).
```

λProlog + CHR 101

even X, X = succ Y, not (double Z Y)

Goals

```
X = succ Y
not (double Z Y)
```

Constraint store

even $\mathbf{F}_X$

Program

```
even zero.
odd (succ X) :- even X.
even (succ X) :- odd X.
even X :- var X, new_constraint (even X) [X].
odd X :- var X, new_constraint (odd X) [X].
double zero zero.
double (succ X) (succ (succ Y)) :- double X Y.
double X Y :- var X, new_constraint (double X Y) [X].
```

Rules

```
(even X) (odd X) <=> fail.
(double _ X) <=> (even X).
```

λProlog + CHR 101

even X, X = succ Y, not (double Z Y)

Goals

```
even (succ Y)
not (double Z Y)
```

Constraint store

Program

```
even zero.
odd (succ X) :- even X.
even (succ X) :- odd X.
even X :- var X, new_constraint (even X) [X].
odd X :- var X, new_constraint (odd X) [X].
double zero zero.
double (succ X) (succ (succ Y)) :- double X Y.
double X Y :- var X, new_constraint (double X Y) [X].
```

Rules

```
(even X) (odd X) <=> fail.
(double _ X) <=> (even X).
```

λProlog + CHR 101

```
even X, X = succ Y, not (double Z Y)
```

Goals

```
odd Y
not (double Z Y)
```

Constraint store

Program

```
even zero.
odd (succ X) :- even X.
even (succ X) :- odd X.
even X :- var X, new_constraint (even X) [X].
odd X :- var X, new_constraint (odd X) [X].
double zero zero.
double (succ X) (succ (succ Y)) :- double X Y.
double X Y :- var X, new_constraint (double X Y) [X].
```

Rules

```
(even X) (odd X) <=> fail.
(double _ X) <=> (even X).
```

λProlog + CHR 101

`even X, X = succ Y, not (double Z Y)`

Goals

```
not (double Z Y)
```

Constraint store

odd $\mathbf{F}_Y$

Program

```
even zero.
odd (succ X) :- even X.
even (succ X) :- odd X.
even X :- var X, new_constraint (even X) [X].
odd X :- var X, new_constraint (odd X) [X].
double zero zero.
double (succ X) (succ (succ Y)) :- double X Y.
double X Y :- var X, new_constraint (double X Y) [X].
```

Rules

```
(even X) (odd X) <=> fail.
(double _ X) <=> (even X).
```

λProlog + CHR 101

```
even X, X = succ Y, not (double Z Y)
```

Goals

```
not ( )
```

Constraint store

odd $\mathbf{F}_Y$

double $\mathbf{F}_Z$ $\mathbf{F}_Y$

Program

```
even zero.
odd (succ X) :- even X.
even (succ X) :- odd X.
even X :- var X, new_constraint (even X) [X].
odd X :- var X, new_constraint (odd X) [X].
double zero zero.
double (succ X) (succ (succ Y)) :- double X Y.
double X Y :- var X, new_constraint (double X Y) [X].
```

Rules

```
(even X) (odd X) <=> fail.
(double _ X) <=> (even X).
```

λProlog + CHR 101

`even X, X = succ Y, not (double Z Y)`

Goals

```
not (even Y)
```

Constraint store

odd F_Y

double F_Z F_Y

Program

```
even zero.
odd (succ X) :- even X.
even (succ X) :- odd X.
even X :- var X, new_constraint (even X) [X].
odd X :- var X, new_constraint (odd X) [X].
double zero zero.
double (succ X) (succ (succ Y)) :- double X Y.
double X Y :- var X, new_constraint (double X Y) [X].
```

Rules

```
(even X) (odd X) <=> fail.
(double _ X) <=> (even X).
```

λProlog + CHR 101

`even X, X = succ Y, not (double Z Y)`

Goals

```
not ( )
```

Constraint store

odd $\mathbf{F}_{Y}$

double $\mathbf{F}_{Z}$ $\mathbf{F}_{Y}$

even $\mathbf{F}_{Y}$

Program

```
even zero.
odd (succ X) :- even X.
even (succ X) :- odd X.
even X :- var X, new_constraint (even X) [X].
odd X :- var X, new_constraint (odd X) [X].
double zero zero.
double (succ X) (succ (succ Y)) :- double X Y.
double X Y :- var X, new_constraint (double X Y) [X].
```

Rules

```
(even X) (odd X) <=> fail.
(double _ X) <=> (even X).
```

λProlog + CHR 101

```
even X, X = succ Y, not (double Z Y)
```

Goals

```
not ( fail )
```

Constraint store

odd $\mathbf{F}_Y$

double $\mathbf{F}_Z$ $\mathbf{F}_Y$

even $\mathbf{F}_Y$

Program

```
even zero.
odd (succ X) :- even X.
even (succ X) :- odd X.
even X :- var X, new_constraint (even X) [X].
odd X :- var X, new_constraint (odd X) [X].
double zero zero.
double (succ X) (succ (succ Y)) :- double X Y.
double X Y :- var X, new_constraint (double X Y) [X].
```

Rules

```
(even X) (odd X) <=> fail.
(double _ X) <=> (even X).
```

λProlog + CHR 101

`even X, X = succ Y, not (double Z Y)`

Goals

Constraint store

odd $\mathbf{F}_Y$

Program

```
even zero.
odd (succ X) :- even X.
even (succ X) :- odd X.
even X :- var X, new_constraint (even X) [X].
odd X :- var X, new_constraint (odd X) [X].
double zero zero.
double (succ X) (succ (succ Y)) :- double X Y.
double X Y :- var X, new_constraint (double X Y) [X].
```

Rules

```
(even X) (odd X) <=> fail.
(double _ X) <=> (even X).
```

Elpi = λProlog + CHR

- λProlog for …
  - ■ backward reasoning, search
  - ✔ programming with binders recursively
- CHR for …
  - ■ forward reasoning
  - ✔ manipulate (frozen) unification variables
  - ✔ handle metadata on unification variables

What about Deducti

https://github.com/Deducteam/lambdapi/pull/418

- Demo
- Code overview

What about Coq-Elpi

https://github.com/LPCIC/coq-elpi/

- Coq’s syntax
  - `predicate {{ nat → lp:X }} :- use X, print {{ bool → lp:X }}.`
- Coq’s API
  - `$ grep pred coq-builtin.elpi  | wc -l`
  - `102`
  - `$ grep pred coq-lib.elpi  | wc -l`
  - `37`
- Coq’s vernacular commands:
  - Elpi Command foo
  - Elpi Tactic bar
- Hierarchy Builder (example)

Thanks!

- Questions?