Higher-Order unification for free

Reusing the meta-language unification for the object language

Fissore Davide & Enrico Tassi

September 10, 2024

Supported by ANR-17-EURE-0004

UNIVERSITÉ CÔTE D'AZUR | ÉCOLE UNIVERSITAIRE DE RECHERCHE SYSTÈMES NUMÉRIQUES POUR L'HUMAIN

FRANCE 2030 Initiative d'Excellence

Metaprogramming for type-class resolution

- Our goal:
  - Type-class solver for Coq in Elpi
- Our problem:
  - The Elpi's unification algorithm differs from Coq's one
- Our contribution:
  - Reusing the meta-language unification for the object language

A type-class problem in Coq

```
Instance forall_dec: ∀A P, Finite A →                (* r3 *)
  (∀x:A, Decision (P x)) → Decision (∀x:A, P x).
```

```
Goal Decision (∀x: fin 7, nfact x 3).                (* g *)
```

A type-class problem in Coq

```
Instance forall_dec: ∀A P, Finite A →                 (* r3 *)
  (∀x:A, Decision (P x)) → Decision (∀x:A, P x) .
```

```
Goal Decision (∀x: fin 7, nfact x 3).                 (* g *)
```

- $\{A \mapsto \mathit{fin}\ 7; P \mapsto \lambda x.(\mathit{nfact}\ x\ 3)\}$

A type-class problem in Coq

```
Instance forall_dec: ∀A P, Finite A →                (* r3 *)
  (∀x:A, Decision (P x)) → Decision (∀x:A, P x).
```

```
Goal Decision (∀x: fin 7, nfact x 3).                 (* g *)
```

- $\{A \mapsto \mathit{fin}\ 7; P \mapsto \lambda x.(\mathit{nfact}\ x\ 3)\}$
- subgoals:
  `Finite (fin 7)` and `(∀x:A, Decision ((λ x.(nfact x 3)) x))`

Coq terms in elpi : HOAS

Coq	Elpi
f	`c"f"`
$f \cdot a$	`app[c"f", c"a"]`
$\lambda(x : T).F \cdot x$	`fun T (x\ app[F, x])`
$\forall(x : T), F \cdot x$	`all T (x\ app[F, x])`
$\dots$	...

The above type-class problem in elpi

```
Instance forall_dec: ∀A P, Finite A →              (* r3 *)
  (∀x:A, Decision (P x)) → Decision (∀x:A, P x).

Goal Decision (∀x: fin 7, nfact x 3).              (* g *)
```

$\downarrow$

The above type-class problem in elpi

```
Instance forall_dec: ∀A P, Finite A →                    (* r3 *)
  (∀x:A, Decision (P x)) → Decision (∀x:A, P x).

Goal Decision (∀x: fin 7, nfact x 3).                    (* g *)
```

$$\downarrow$$

```
decision (all A (x\ app [P, x])) :- finite A,            % r3
  pi w\ decision (app [P, w]).

?- decision (all (app [c"fin", c"7"])                    % g
                 (x\ app [c"nfact", x, c"3"])).
```

Solving the goal in elpi

```
decision (all A (x\ app [P, x] )) :- finite A,          % r3
  pi w\ decision (app [P, w]).

?- decision (all (app ["fin", "7"])                     % g
                 (x\ app [c"nfact", x, c"3"] )).
```

The idea

```
decision (all A (x\ P' x)) :-                      % r3
  link P' (fun (x\ app[P, x])),
  finite A,
  pi w\ decision (P' x).

?- decision (all (app ["fin", "7"])                % g
                 (x\ app [c"nfact", x, c"3"])).
```

What we propose

1. Compilation:
   - Recognize *problematic subterms* $p_1, \ldots, p_n$
   - Replace p_i with fresh unification variables X_i
   - *Link* p_i with X_i

     A link is a suspended unification problem
2. Runtime:
   - Unify p_i and X_i only when some conditions hold
   - Decompile remaining links

Some notations

- $\mathbb{P}$: the unification problems in the object language (ol)
- $\mathbb{Q}$: the unification problems in the meta-language (ml)
- $\mathbb{L}$, $\mathbb{M}$: the link store, the map store
- Three kinds of links: $\diamond\beta$, $\diamond\eta$, $\diamond\mathcal{L}_\lambda$[1]

- $\mathrm{run}_o(\mathbb{P}, n) \mapsto \rho$: the run of n unif pb in the ol
- $\mathrm{run}_m(\mathbb{P}, n) \mapsto \rho'$: the run of n unif pb in the ml
- $\mathrm{step}_o(\mathbb{P}, i, \rho_{i-1}) \mapsto \rho_i$: the execution of the i^{th} unif pb in ol
- $\mathrm{step}_m(\mathbb{Q}, i, \sigma_{i-1}, \mathbb{L}_{i-1}) \mapsto (\sigma_i, \mathbb{L}_i)$: the exec of the i^{th} unif pb in ml

[1] $\mathcal{L}_\lambda$ is a notation for the *pattern fragment*

Proven properties

Run Equivalence $\forall \mathbb{P}, \forall n$, if $\mathbb{P} \subseteq \mathcal{L}_\lambda$

$$\mathrm{run}_o(\mathbb{P}, n) \mapsto \rho \wedge \mathrm{run}_m(\mathbb{P}, n) \mapsto \rho' \Rightarrow \forall s \in \mathbb{P}, \rho s =_o \rho' s$$

Simulation fidelity $\forall \mathbb{P}$, in the context of run_o and run_m, $\forall i \in 1 \dots n$,

$$\mathrm{step}_o(\mathbb{P}, i, \rho_{i-1}) \mapsto \rho_i \Leftrightarrow \mathrm{step}_m(\mathbb{Q}, i, \sigma_{i-1}, \mathbb{L}_{i-1}) \mapsto (\sigma_i, \mathbb{L}_i)$$

Compilation round trip If the compilation of s gives a term t and the stores $\mathbb{L}$ and $\mathbb{M}$ then $\forall \sigma$,

$$\langle \sigma, \mathbb{M}, \mathbb{L} \rangle^{-1} \mapsto \rho \wedge \rho t =_o \rho s$$

Problematic subterms recognition: $\diamond\beta$

- $X \cdot y$ becomes `A y` with mapping $X \mapsto A$
- For example, $\lambda y.X \cdot y = \lambda y.f \cdot y \cdot a$
- Is compiled into: `fun (w\ A w) = fun (w\ app[c"f", w, c"a"])`
- Unification gives: $\{A \mapsto$ `(w\ app[c"f", w, c"a"])`$\}$
- Decompilation of A gives $\{X \mapsto \lambda y.f \cdot y \cdot a\}$

Problematic subterms recognition: $\diamond\eta$

- $\lambda x.s \in \diamond\eta$, if $\exists\rho, \rho(\lambda x.s)$ is an η-redex
- Detection of $\diamond\eta$ terms is not trivial:
- $$\begin{array}{lll} \lambda x.f \cdot (A \cdot x) & \in \diamond\eta & \rho = \{\ A \mapsto \lambda x.x\ \} \\ \lambda x.f \cdot (A \cdot x) \cdot x & \in \diamond\eta & \rho = \{\ A \mapsto \lambda x.a\ \} \\ \lambda x.f \cdot x \cdot (A \cdot x) & \notin \diamond\eta & \\ \lambda x.\lambda y.f \cdot (A \cdot x) \cdot (B \cdot y \cdot x) & \in \diamond\eta & \rho = \{\ A \mapsto \lambda x.x\ ;\ B \mapsto \lambda y.\lambda x.y\ \} \end{array}$$

Problematic subterms recognition: $\diamond\eta$ link resumption

- Several conditions: like lhs is assigned to a rigid term, two η-link with same lhs, the rhs becomes outside $\diamond\eta$...
- These conditions guarantee the prefixed properties !
- An example:

$$
\begin{aligned}
\mathbb{P} &= \{ \quad f \simeq_o \lambda x.(f \cdot (X \cdot x)) \ \} \\
\mathbb{Q} &= \{ \ \texttt{"f"} \simeq_m \texttt{A} \ \} \\
\mathbb{M} &= \{ \ X \mapsto \texttt{B} \ \} \\
\mathbb{L} &= \{ \ \vdash \texttt{A} =_\eta \texttt{fun (x\textbackslash\ app[c"f", B x])} \ \}
\end{aligned}
$$

- After unification of `A` with `c"f"`, the lhs of the link becomes rigid and `fun (x\ app[c"f", B x])` is unified with `fun (x\ app[c"f", x])`
- That is $\{B \mapsto \texttt{x\textbackslash x}\}$
- Decompilation will assign $\lambda x.x$ to X

Problematic subterms recognition: $\diamond\mathcal{L}_\lambda$

- Example:

$$\begin{array}{ll} \mathbb{P} = \{\ X \simeq_o \lambda x.a & (X \cdot a) \simeq_o a \ \} \\ \mathbb{Q} = \{\ \texttt{A} \simeq_m \texttt{fun (x\textbackslash\ c"a")} & \texttt{B} \simeq_m \texttt{"a"} \ \} \\ \mathbb{M} = \{\ X \mapsto \texttt{A}\ \} & \\ \mathbb{L} = \{\ \vdash \texttt{B} =_{\mathcal{L}_\lambda} \texttt{A (c"a")}\ \} & \end{array}$$

- After unification of `A` with `fun (x\"a")`, the lhs of the $\mathcal{L}_\lambda$-link becomes `c"a"`, the link is triggered and B is unified to `c"a"`
- Decompilation will assign $\lambda x.a$ to A

Going further: the Constraint Handling Rules

- Elpi has a CHR for goal suspension and resumption
- This fits well our notion of link: a suspended unification problem

---

This can easily introduce new unification behaviors

- We can for example mimic the unification of the ol
- Add heuristic for HO unification outside the pattern fragment

---

```
% By def, R is not in the pattern fragment
link-llam L R :- not (var L), unif-heuristic L R.
```

Conclusion

- Takes advantage of the unification capabilities of the meta language at the price of handling problematic sub-terms on the side.
- It is worth mentioning that we replace terms with variables only when it is strictly needed, leaving the rest of the term structure intact and hence *indexable*.
- Our approach is flexible enough to accommodate different strategies and *heuristics* to handle terms outside the pattern fragment