Higher-Order unification for free

Reusing the meta-language unification for the object language

Fissore Davide & Enrico Tassi

June 13, 2024

Context

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 fin_fin: ∀n, Finite (fin n).                  (* r1 *)
Instance nfact_dec: ∀n nf, Decision (nfact n nf). (* r2 *)
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 fin_fin: ∀n, Finite (fin n).                   (* r1 *)
Instance nfact_dec: ∀n nf, Decision (nfact n nf).        (* r2 *)
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 *)
```

- Back-chain to `forall_dec` with
- $\{A \mapsto \mathit{fin}\ 7; P \mapsto \lambda x.(\mathit{nfact}\ x\ 3)\}$

A type-class problem in Coq

```
Instance fin_fin: ∀n, Finite (fin n).                   (* r1 *)
Instance nfact_dec: ∀n nf, Decision (nfact n nf).       (* r2 *)
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 fin_fin: ∀n, Finite (fin n).                  (* r1 *)
Instance nfact_dec: ∀n nf, Decision (nfact n nf).      (* r2 *)
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

Coq	Elpi
$f \cdot a$	`app["f", "a"]`
$\lambda x.\lambda y.F_{xy}$	`lam x\ lam y\ app[uva F [], x, y]`
$\lambda x.F_x a$	`lam x\ app[uva F [], x, "a"]`

Note on unification:

- In coq: $\lambda x.F_x$ unifies with $\lambda x.f\ x\ 3$
- In elpi: "`lam x\app [F, x]`" can't unify with "`lam x\app [f, x, 3]`"
- But, "`lam x\F x`" unifies with "`lam x\app [f, x, 3]`"

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 *)
```

↓

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 ["fin", "7"]) x\               % g
                   app ["nfact", x, "3"]).
```

The above type-class problem in elpi

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

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

NOTE: Elpi can unify `(P x)` with `app["nfact", x, "3"]`

The idea

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

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

Compilation and simulation

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
2. Runtime:
   - Unify p_i and X_i only when some conditions hold
   - Decompile remaining links

NOTE: This unification strategy is generalizable to any meta-language when manipulating terms of the object language

Some notations

- $\mathbb{P}$: the unification problems in the object language (ol)
- $\mathbb{Q}$: the unification problems in the target language (ml)
- step_o: the execution of a unif pb in the ol
- step_m: the execution of a unif pb in the ml
- run_o: the run of n steps
- run_m: the run of n steps

- $\mathbb{M}$, $\mathbb{L}$: the map store, the link store
- A link in $\mathbb{L}$ is like $X =_\lambda t$
- A mapping in $\mathbb{M}$ is like $\{X \mapsto t\}$

Proven properties

Run Equivalence $\forall \mathbb{P}, \forall n$, if $\mathbb{P} \subseteq \mathcal{L}$
$\mathrm{run}_o(\mathbb{P}, n) \mapsto \rho \wedge \mathrm{run}_m(\mathbb{P}, n) \mapsto \rho' \Rightarrow \forall s \in \mathbb{P}, \rho s =_o \rho' s$

Simulation fidelity In the context of run_o and run_m,
if $\mathbb{P} \subseteq \mathcal{L}$ we have that $\forall p \in 1 \ldots n$,
$\mathrm{step}_o(\mathbb{P}, p, \rho_{p-1}) \mapsto \rho_p \Leftrightarrow \mathrm{step}_m(\mathbb{Q}, p, \sigma_{p-1}, \mathbb{L}_{p-1}) \mapsto (\sigma_p, \mathbb{L}_p)$

Fidelity ricovery In the context of run_o and run_m,
if $\rho_{p-1}\mathbb{P}_p \subseteq \mathcal{L}$ (even if $\mathbb{P}_p \not\subseteq \mathcal{L}$) then
$\mathrm{step}_o(\mathbb{P}, p, \rho_{p-1}) \mapsto \rho_p \Leftrightarrow \mathrm{step}_m(\mathbb{Q}, p, \sigma_{p-1}, \mathbb{L}_{p-1}) \mapsto (\sigma_p, \mathbb{L}_p)$

Problematic subterms recognition: $\diamond\beta$

A HO variable in the pattern fragment:

- X_x becomes `A x` with mapping $X \mapsto A^1$
- Decompilation: transform the lambda abstraction of the meta language to the lambda abstraction of the object one.
- For example, if $\{A \mapsto (x \backslash f \cdot x \cdot a)\}$, then decompilation produces the following substitution $\{X \mapsto \lambda x. f \cdot x \cdot a\}$

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:
- $$\begin{array}{lll} \lambda x.f \cdot (A \cdot x) & \in \diamond\eta & \rho = \{\ A \mapsto \lambda x.x\ \} \\ \lambda x.f \cdot (A \cdot x) \cdot x & \in \diamond\eta & \rho = \{\ A \mapsto \lambda x.a\ \} \\ \lambda x.f \cdot x \cdot (A \cdot x) & \notin \diamond\eta & \\ \lambda x.\lambda y.f \cdot (A \cdot x) \cdot (B \cdot y \cdot x) & \in \diamond\eta & \rho = \{\ A \mapsto \lambda x.x\ ;\ B \mapsto \lambda y.\lambda x.y\ \} \end{array}$$
- Need of some primitives like `may-contract-to` and `occurs-rigidly`

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

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

$$\begin{aligned}
\mathbb{P} &= \{\ \lambda x.X \cdot x \ \simeq_o\ f\ \} \\
\mathbb{Q} &= \{\ \qquad A \simeq_m f\ \} \\
\mathbb{M} &= \{\ X \mapsto B^1\ \} \\
\mathbb{L} &= \{\ \vdash A =_\eta \lambda x.B_x\ \}
\end{aligned}$$

- After unification of A with f, the lhs of the link is assigned, the link is triggered and $\lambda x.B_x$ is unified with $\lambda x.f \cdot x$
- That is $\{B_x \mapsto f\}$
- Decompilation will assign $\lambda x.f \cdot x$ to X

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

Use of heuristics

Use of CHR

Conclusion

- Takes advantage of the unification capabilities of the meta language at the price of handling problematic sub-terms on the side.
- As a result our encoding takes advantage of indexing data structures and mode analysis for clause filtering.
- 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