Hints in unification

Enrico Tassi

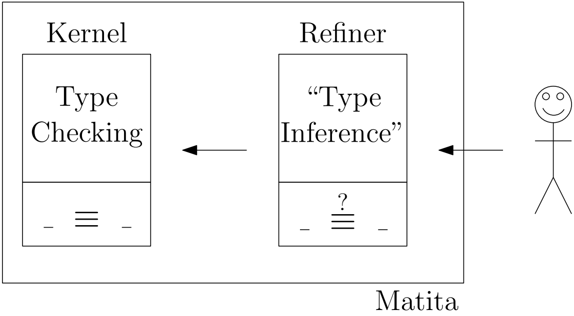
University of Bologna - Department of Computer Science

13 May 2009

"Hints in Unification", A.Asperti, W. Ricciotti, C. Sacerdoti Coen, E. Tassi. Accepted for publication in the proceedings of TPHOLs 2009

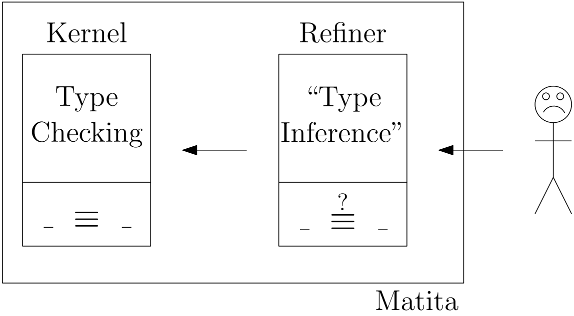
Context of the work

- Matita: ITP based on CIC
- Type-checking v.s. Type-inference
  - conversion v.s. unification



Context of the work

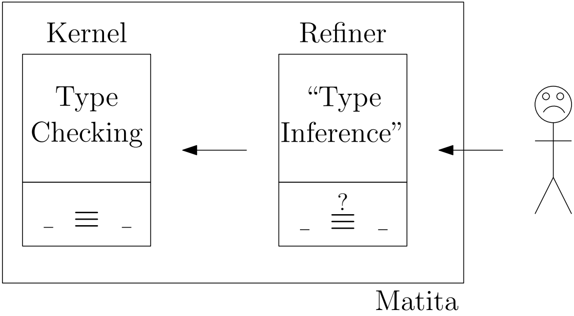
- Matita: ITP based on CIC
- Type-checking v.s. Type-inference
  - conversion v.s. unification



- The user works all the time with the refiner, that is based on an heuristic.

Context of the work

- Matita: ITP based on CIC
- Type-checking v.s. Type-inference
  - conversion v.s. unification



- The user works all the time with the refiner, that is based on an heuristic.
- Can the user customise/drive unification?!

(Ad hoc) mechanisms

Unification is an hard problem in CIC

$$t_1 \quad \stackrel{?}{\equiv} \quad t_2 \qquad \text{find } \sigma \text{ such that} \quad t_1\sigma \equiv t_2\sigma$$

Very common unification problems deserve ad-hoc mechanisms to let the user drive the unification algorithm

- canonical structures
- coercions pullback
- ...

Unification hints are a framework to let the user customise the unification algorithm that generalise these mechanisms (and little more...)

Outline

1. Unification hints by examples
2. (re) implementing ad-hoc mechanisms with hints
3. Unification hints and reification
4. Conclusions

Unification hints — an example

- Hints are pairs of convertible terms
- Unification algorithm $\mathcal{U}\ t_1\ t_2$
- $\mathcal{U}'\ t_1\ t_2 =$
  $\text{try}\ \mathcal{U}\ t_1\ t_2$
  $\text{with Fail} \Rightarrow \text{hints}\ t_1\ t_2$

Unification hints — an example

- Hints are pairs of convertible terms
- Unification algorithm $\mathcal{U}\ t_1\ t_2$
- $\mathcal{U}'\ t_1\ t_2 =$
  $\text{try}\ \mathcal{U}\ t_1\ t_2$
  $\text{with Fail} \Rightarrow \text{hints}\ t_1\ t_2$

Example: $?_1 + ?_2 \stackrel{?}{\equiv} S(x + y)$

Unification hints — an example

- Hints are pairs of convertible terms
- Unification algorithm $\mathcal{U}\ t_1\ t_2$
- $\mathcal{U}'\ t_1\ t_2 =$
  $\text{try}\ \mathcal{U}\ t_1\ t_2$
  $\text{with Fail} \Rightarrow \text{hints}\ t_1\ t_2$

Example: $?_1 + ?_2 \stackrel{?}{\equiv} S(x + y)$

User-declared hint: $x, y : \mathbb{N} \vdash (S\ x) + y \equiv S(x + y)$

Unification hints — an example

- Hints are pairs of convertible terms
- Unification algorithm $\mathcal{U}\ t_1\ t_2$
- $\mathcal{U}'\ t_1\ t_2 =$
  $\text{try } \mathcal{U}\ t_1\ t_2$
  $\text{with Fail} \Rightarrow \text{hints } t_1\ t_2$

Example: $?_1 + ?_2 \stackrel{?}{\equiv} S(x+y)$

User-declared hint:

$$\frac{}{(S\ ?_x) + ?_y \equiv S\ (?_x + ?_y)}\ \text{hint}$$

Unification hints — an example

- Hints are pairs of convertible terms
- Unification algorithm $\mathcal{U}\ t_1\ t_2$
- $\mathcal{U}'\ t_1\ t_2 =$
  $\text{try}\ \mathcal{U}\ t_1\ t_2$
  $\text{with Fail} \Rightarrow \text{hints}\ t_1\ t_2$

Example: $?_1 + ?_2 \stackrel{?}{\equiv} S(x+y)$

User-declared hint:

$$\frac{}{(S\ ?_x) + ?_y \equiv S\ (?_x + ?_y)}\ \text{hint}$$

$$\frac{(S\ ?_x) + ?_y \stackrel{?}{=} ?_1 + ?_2 \qquad S(x+y) \stackrel{?}{=} S\ (?_x + ?_y)}{?_1 + ?_2 \stackrel{?}{\equiv} S\ (x+y)}\ \text{hint}$$

Unification hint — recursion

What is used to solve $\_ \stackrel{?}{=} \_$? $\mathcal{U}$ or $\mathcal{U}'$?

Another example: $?_1 + ?_2 \stackrel{?}{\equiv} S\ (S\ (x+y))$

$$\frac{\ldots \qquad S\ (S\ (x+y)) \stackrel{?}{=} S\ (?_x + ?_y)}{?_1 + ?_2 \stackrel{?}{\equiv} S\ (S\ (x+y))}\ \text{hint}$$

Unification hint — recursion

What is used to solve $\_ \stackrel{?}{=} \_$? $\mathcal{U}$ or $\mathcal{U}'$?

Another example: $?_1 + ?_2 \stackrel{?}{\equiv} S\ (S\ (x + y))$

$$\frac{\dots \qquad S\ (x + y) \stackrel{?}{=} ?_x + ?_y}{?_1 + ?_2 \stackrel{?}{\equiv} S\ (S\ (x + y))}\ \text{hint}$$

Unification hint — recursion

What is used to solve $\_ \stackrel{?}{=} \_$? $\mathcal{U}$ or $\mathcal{U}'$?

Another example: $?_1 + ?_2 \stackrel{?}{\equiv} S\ (S\ (x+y))$

$$\frac{\dots \qquad S\ (x+y) \stackrel{?}{=} ?_x + ?_y}{?_1 + ?_2 \stackrel{?}{\equiv} S\ (S\ (x+y))}\ \text{hint}$$

Problems generated by hints need $\mathcal{U}'$

Unification hint — recursion

What is used to solve $\_ \stackrel{?}{=} \_$? $\mathcal{U}$ or $\mathcal{U}'$?

Another example: $?_1 + ?_2 \stackrel{?}{\equiv} S\ (S\ (x + y))$

$$\frac{\ldots \qquad S\ (x+y) \stackrel{?}{=} ?_x + ?_y}{?_1 + ?_2 \stackrel{?}{\equiv} S\ (S\ (x+y))}\ \text{hint}$$

Problems generated by hints need $\mathcal{U}'$

Possible (new) source of divergence...

Unification hint — indexing

How to efficiently index (many) hints?

Unification hint — indexing

How to efficiently index (many) hints?
We could use discrimination trees (nets):

$$(S\ \_) + \_\ ,\ S\ (\_ + \_)\ \mapsto\ \dfrac{}{(S\ ?_x) + ?_y \equiv S\ (?_x + ?_y)}\ \text{hint}$$

Unification hint — indexing

How to efficiently index (many) hints?
We could use discrimination trees (nets):

$$(S\ \_) + \_\ ,\ S\ (\_ + \_) \ \mapsto\ \frac{}{(S\ ?_x) + ?_y \equiv S\ (?_x + ?_y)}\ \text{hint}$$

The first example is OK: $?_1 + ?_2 \stackrel{?}{\equiv} S\ (x + y)$

Unification hint — indexing

How to efficiently index (many) hints?
We could use discrimination trees (nets):

$$(S\ \_) + \_\ ,\ S\ (\_ + \_) \ \mapsto\ \frac{}{(S\ ?_x) + ?_y \equiv S\ (?_x + ?_y)}\ \text{hint}$$

The first example is OK: $\_ + \_\ ,\ S\ (\_ + \_)$

Unification hint — indexing

How to efficiently index (many) hints?
We could use discrimination trees (nets):

$$(S\ \_) + \_\ ,\ S\ (\_ + \_)\ \mapsto\ \frac{}{(S\ ?_x) + ?_y \equiv S\ (?_x + ?_y)}\ \text{hint}$$

The first example is OK: $\_ + \_\ ,\ S\ (\_ + \_)$

The second example is KO: $?_1 + ?_2 \stackrel{?}{\equiv} S\ (S\ (x + y))$

Unification hint — indexing

How to efficiently index (many) hints?
We could use discrimination trees (nets):

$$(S\ \_) + \_\ ,\ S\ (\_ + \_) \ \mapsto\ \frac{}{(S\ ?_x){+}?_y \equiv S\ (?_x{+}?_y)}\,\text{hint}$$

The first example is OK: $\_ + \_\ ,\ S\ (\_ + \_)$

The second example is KO: $\_ + \_\ ,\ S\ (S\ (\_ + \_))$

Unification hint — indexing

How to efficiently index (many) hints?
We could use discrimination trees (nets):

$$(S\ \_) + \_\ ,\ S\ (\_ + \_)\ \mapsto\ \frac{}{(S\ ?_x){+}?_y \equiv S\ (?_x{+}?_y)}\ \text{hint}$$

The first example is OK: $\_ + \_\ ,\ S\ (\_ + \_)$

The second example is KO: $\_ + \_\ ,\ S\ (S\ (\_ + \_))$

Some subterms (where $\mathcal{U}'$ may be needed) can confuse indexing

Unification hints

The correct version of the hint:

$$x, y : \mathbb{N}, z := x + y \vdash (S\ x) + y \equiv S\ z$$

Unification hints

The correct version of the hint:

$$\frac{?_z \equiv ?_x + ?_y}{(S\ ?_x) + ?_y \equiv S\ ?_z}\ \text{myhint}$$

Application

$$\frac{(S\ ?_x) + ?_y \stackrel{?}{=} A \qquad B \stackrel{?}{=} S\ ?_z \qquad ?_z \stackrel{?}{\equiv} ?_x + ?_y}{A \stackrel{?}{\equiv} B}\ \text{myhint}$$

Hints for canonical structures

In presence of records like

$$\mathcal{Z} : \mathrm{Group} := \{\mathrm{gcarr} := \mathbb{Z}; \ldots\}$$

The canonical structure mechanism allows the user to specify a canonical solution for $?_g$:

$$\mathrm{gcarr}\ ?_g \stackrel{?}{\equiv} \mathbb{Z} \qquad \Rightarrow \qquad ?_g := \mathcal{Z}$$

By declaring the following hint, the user obtains the same result

$$\frac{}{\mathrm{gcarr}\ \mathcal{Z} \equiv \mathbb{Z}}\ \text{hint-for-group-}\mathcal{Z}$$

Coercions pullback

The user inputs:

$$x * (y + z)$$

Arguments of $*$ must have the same type:

$$\mathrm{gcarr}\ ?_g \stackrel{?}{\equiv} \mathrm{mcarr}\ ?_m$$

$$\begin{tikzcd}
 & \boxed{\text{ring}} \arrow[dl, "\text{r_group}"'] \arrow[dr, "\text{r_monoid}"] & \\
\boxed{\text{group}} \arrow[dr, "\text{gcarr}"'] & & \boxed{\text{monoid}} \arrow[dl, "\text{mcarr}"] \\
 & \text{Type} &
\end{tikzcd}$$

The solution is the pullback of gcarr and mcarr.

$$?_g := \mathrm{r\_group}\ ?_r$$

$$?_m := \mathrm{r\_monoid}\ ?_r$$

The graph must be coherent!

Hints for pullbacks

Problems have many similar forms:

- $f_1 \; ?_1 \overset{?}{\equiv} g_1 \; ?_2$
- $f_1 \; (f_3 \; ?_1) \overset{?}{\equiv} g_1 \; ?_2$
- ...

All diamonds must be declared as hints, but smaller ones should be preferred.

$$\frac{}{f_1 \; (f_2 \;\; ?_c) \equiv g_1 \; (g_2 \;\; ?_c)}\;\text{hint-1}$$

$$\frac{}{f_1 \; (f_3 \;\; ?_d) \equiv g_1 \; (g_3 \;\; ?_d)}\;\text{hint-2}$$

$$\begin{tikzcd}
 & D \arrow[d, "h"] \arrow[ddl, "f_3"'] \arrow[ddr, "g_3"] & \\
 & C \arrow[dl, "f_2"] \arrow[dr, "g_2"'] & \\
A \arrow[dr, "f_1"'] & & B \arrow[dl, "g_1"] \\
 & T &
\end{tikzcd}$$

Reflexive tactics — reification with hints

When we build a reflexive tactic, we need a syntactic representation of the input that cannot be obtained inside the calculus (usually built in $\mathcal{L}$-tac, OCaml, ...)

```
inductive Expr : Type :=
  | Eunit  : Expr
  | Emult  : Expr → Expr → Expr
  | Einv   : Expr → Expr
  | Evar   : ℕ → Expr.
```

let rec $[\![e : \mathrm{Expr};\ \Gamma : \mathrm{list}\ (\mathrm{gcarr}\ \mathrm{g})]\!]_{(\mathrm{g:group})}$ on e : gcarr g :=

  match e **with**

  [Eunit $\Rightarrow 1_g$

  | Emult x y $\Rightarrow [\![x;\ \Gamma]\!]_g * [\![y;\ \Gamma]\!]_g$

  | Einv x $\Rightarrow [\![x;\ \Gamma]\!]_g^{-1}$

  | Evar n $\Rightarrow \Gamma(n)$].

Reflexive tactics — reification with hints

The unification problem for reification with sharing is:

$$[\![?_1;\ ?_2]\!]_{?_3} \stackrel{?}{\equiv} x * (x^{-1} * y)$$

Unification hints follows:

$$\frac{?_m \equiv [\![?_x;\ ?_\Gamma]\!] \qquad ?_n \equiv [\![?_y;\ ?_\Gamma]\!]}{[\![\mathrm{Emult}\ ?_x\ ?_y;\ ?_\Gamma]\!]_{?_g} \equiv ?_m * ?_n}\ \text{h-times}$$

$$\frac{}{[\![\mathrm{Eunit};\ ?_\Gamma]\!]_{?_g} \equiv 1}\ \text{h-unit}$$

$$\frac{?_o \equiv [\![?_z;\ ?_\Gamma]\!]_{?_g}}{[\![\mathrm{Einv}\ ?_z;\ ?_\Gamma]\!]_{?_g} \equiv ?_o^{-1}}\ \text{h-inv}$$

Reflexive tactics — reification with hints

The tricky part is the set of hints to reify variables

$$\frac{}{[\![\mathrm{Evar}\ 0;\ ?_r :: ?_\ominus]\!]_{?_g} \equiv ?_r}\ \text{h-var-base}$$

$$\frac{?_q \equiv [\![\mathrm{Evar}\ ?_p;\ ?_\Delta]\!]_{?_g}}{[\![\mathrm{Evar}\ (S\ ?_p);\ ?_s :: ?_\Delta]\!]_{?_g} \equiv ?_q}\ \text{h-var-rec}$$

Conclusion

- Unification hints are a general framework to let the user drive the unification algorithm
- Unification hints are general enough to express canonical structures and coercions pullback
- (not so) unexpectedly they can drive the unification algorithm in performing reification
- Future works: pragmatic study of divergence.

Conclusion

- Unification hints are a general framework to let the user drive the unification algorithm
- Unification hints are general enough to express canonical structures and coercions pullback
- (not so) unexpectedly they can drive the unification algorithm in performing reification
- Future works: pragmatic study of divergence.

Thanks for your attention!