Unification Utilities for Isabelle/ML

Kevin Kappelmann

October 4, 2023

Abstract

This article provides various unification utilities for Isabelle/ML, most prominently:

1. First-order and higher-order pattern E-unification and E-matching. While unifiers in Isabelle/ML only consider the $\alpha\beta\eta$-equational theory of the $\lambda$-calculus, unifiers in this article may take an extra background theory, in the form of an equational prover, into account. For example, the unification problem $n + 1 \equiv \exists m + Suc \ 0$ may be solved by providing a prover for the background theory $\forall n. \ n + 1 \equiv n + Suc \ 0$.

2. Tactics, methods, and attributes with adjustable unifiers (e.g. resolution, fact, assumption, OF).

3. A generalisation of unification hints [1]. Unification hints are a flexible extension for unifiers. Among other things, they can be used for reflective tactics, to provide canonical unification instances, or to simply strengthen the background theory of a unifier in a controlled manner.


5. Practical combinations of unification algorithms, e.g. a combination of first-order and higher-order pattern unification.

6. A hierarchical logger for Isabelle/ML, including per logger configurations with log levels, output channels, message filters.

While this entry works with every object logic, some extra setup for Isabelle/HOL and application examples are provided. All unifiers are tested with SpecCheck [2].

Contents

1 ML Code Utils 3
2 ML Attributes 3
3 ML Logger 3
  3.1 Setup Result Commands ....................... 4
  3.2 Examples .................................. 4
1 ML Code Utils

theory ML-Code-Utils
imports Pure
begin

Summary Utilities to generate and manipulate (parsed) ML code.
⟨ML⟩

end

2 ML Attributes

theory ML-Attributes
imports ML-Code-Utils
begin

Summary ML code as attributes.
⟨ML⟩

end

3 ML Logger

theory ML-Logger
imports ML-Attributes
begin

Summary Generic logging, at some places inspired by Apache’s Log4J
⟨ML⟩
3.1 Setup Result Commands

theory Setup-Result-Commands
imports Pure
keywords setup-result :: thy-decl and local-setup-result :: thy-decl
begin

Summary  Setup and local setup with result commands

end

3.2 Examples

theory ML-Logger-Examples
imports ML-Logger Setup-Result-Commands
begin

First some simple, barebone logging: print some information.

ML
To guarantee the existence of a "logger" in an ML structure, one should use the HAS-LOGGER signature.

ML
We can set up a hierarchy of loggers

ML
We can use different log levels to show/suppress messages. The log levels are based on Apache’s Log4J 2 https://logging.apache.org/log4j/2.x/manual/customloglevels.html.

ML declare [[ML-map-context (Logger.set-log-level parent1 Logger.DEBUG)]]

ML We can set options for all loggers below a given logger. Below, we set the log level for all loggers below (and including) parent1 to error, thus disabling warning messages.

ML declare [[ML-map-context (Logger.set-log-levels parent1 Logger.ERR)]]

ML declare [[ML-map-context (Logger.set-log-levels parent1 Logger.INFO)]]

We can set message filters.
One can also use different output channels (e.g. files) and hide/show some additional logging information. Ctrl+click on below values and explore.

To set up (local) loggers outside ML environments, 
\textit{ML-Unification.Setup-Result-Commands} contains two commands, \texttt{setup-result} and \texttt{local-setup-result}.

experiment
begin
\texttt{local-setup-result} \texttt{local-logger} = \texttt{Logger.new-logger Logger.root-logger Local}
\end
\textit{local-logger} is no longer available. The follow thus does not work:

Let us create another logger in the global context.
\texttt{setup-result} \texttt{some-logger} = \texttt{Logger.new-logger Logger.root-logger Some-Logger}
\textit{Let us delete it again.}
\texttt{declare} [[\textit{ML-map-context} \texttt{Logger.delete-logger some-logger}]]

The logger can no longer be found in the logger hierarchy.
\end

\section{ML Attribute Utils}

\texttt{theory} \textit{ML-Attribute-Utils}
\begin
\texttt{imports} \textit{Pure}
\end
\texttt{begin}
\texttt{Summary} \texttt{Utilities for attributes.}
\end

\section{ML Conversion Utils}

\texttt{theory} \textit{ML-Conversion-Utils}
imports
  Pure
begin

Summary  Utilities for conversions.

lemma  meta-eq-symmetric: (A ≡ B) ≡ (B ≡ A)
⟨proof⟩
⟨ML⟩
end

6 ML Parsing Utils

theory  ML-Parsing-Utils
  imports  ML-Attributes
           ML-Attribute-Utils
begin

Summary  Parsing utilities for ML. We provide an antiquotation that takes a list of keys and creates a corresponding record with getters and mappers and a parser for corresponding key-value pairs.
⟨ML⟩

Example  ⟨ML⟩
end

7 ML Functor Instances

theory  ML-Functor-Instances
  imports  ML-Parsing-Utils
begin

Summary  Utilities for ML functors that create context data.
⟨ML⟩

Example  ⟨ML⟩
end

8 General ML Utils

theory  ML-General-Utils
imports Pure
begin

Summary  General ML utilities.
⟨ML⟩
end

9  ML Generic Data Utils

theory ML-Generic-Data-Utils
  imports Pure
begin

Summary  Utilities for Generic_Data.
⟨ML⟩
end

10  ML Method Utils

theory ML-Method-Utils
  imports Pure
begin

Summary  Utilities for methods.
⟨ML⟩
end

theory ML-Priorities
  imports ML-Parsing-Utils
begin

Summary  Priorities for ML tactics.
⟨ML⟩
end

11  ML-Normalisations

theory ML-Normalisations
  imports Pure
begin
Summary Normalisation functions for terms, types, and theorems.

\begin{ML}
end
\end{ML}

12 ML-Binders

theory ML-Binders
  imports
    ML-General-Utils
    ML-Normalisations
begin

Summary Binders for ML.

\begin{ML}
end
\end{ML}

13 ML Term Utils

theory ML-Term-Utils
  imports ML-Binders
begin

Summary Utilities for terms.

\begin{ML}
end
\end{ML}

14 ML Tactic Utils

theory ML-Tactic-Utils
  imports
    ML-Logger
    ML-Term-Utils
    ML-Conversion-Utils
begin

Summary Utilities for tactics.

\begin{ML}
end
\end{ML}
15  ML Theorem Utils

 theory ML-Theorem-Utils
 imports Pure
 begin

 Summary Utilities for theorems.
 ⟨ML⟩

 end

16  ML Utils

 theory ML-Utils
 imports
 ML-Attribute-Utils
 ML-Conversion-Utils
 ML-Functor-Instances
 ML-General-Utils
 ML-Generic-Data-Utils
 ML-Method-Utils
 ML-Attributes
 ML-Code-Utils
 ML-Parsing-Utils
 ML-Priorities
 ML-Tactic-Utils
 ML-Term-Utils
 ML-Theorem-Utils
 begin

 end

17  ML Unification Basics

 theory ML-Unification-Base
 imports
 ML-Logger
 ML-Binders
 ML-Normalisations
 begin

 Summary Basic definitions and utilities for unification algorithms.
 ⟨ML⟩

 end
18 Simps To

theory Simps-To
imports
  ML-Tactic-Utils
  ML-Theorem-Utils
  ML-Unification-Base
  Setup-Result-Commands
begin

Summary Simple frameworks to ask for the simp-normal form of a term on the user-level.

setup-result simps-to-base-logger = (Logger.new-logger Logger.root-logger Simps-To-Base)

Using Simplification On Left Term definition SIMPS-TO s t ≡ (s ≡ t)

lemma SIMPS-TO-eq: SIMPS-TO s t ≡ (s ≡ t)
(proof)

  Prevent simplification of second/right argument

lemma SIMPS-TO-cong [cong]: s ≡ s' ===> SIMPS-TO s t ≡ SIMPS-TO s' t
(proof)

lemma SIMPS-TOI: PROP SIMPS-TO s s (proof)
lemma SIMPS-TOD: PROP SIMPS-TO s t ===> s ≡ t (proof)
(ML)

Using Simplification On Left Term Followed By Unification definition SIMPS-TO-UNIF s t ≡ (s ≡ t)

  Prevent simplification

lemma SIMPS-TO-UNIF-cong [cong]: SIMPS-TO-UNIF s t ≡ SIMPS-TO-UNIF s t (proof)

lemma SIMPS-TO-UNIF-eq: SIMPS-TO-UNIF s t ≡ (s ≡ t) (proof)

lemma SIMPS-TO-UNIFI: PROP SIMPS-TO s s' ===> s' ≡ t ===> PROP SIMPS-TO-UNIF s t
(proof)
lemma SIMPS-TO-UNIFD: PROP SIMPS-TO-UNIF s t ===> s ≡ t
(proof)
(ML)

Examples experiment
begin
lemma
assumes [simp]: $P \equiv Q$
and [simp]: $Q \equiv R$
shows $PROP\ SIMPS-TO \ P\ Q$
⟨proof⟩
⟨ML⟩
⟨proof⟩

schematic-goal
assumes [simp]: $P \equiv Q$
and [simp]: $Q \equiv R$
shows $PROP\ SIMPS-TO \ P\ ?Q$
⟨proof⟩
end

19 ML Unifiers

theory $ML\text{-Unifiers}$
imports
$ML\text{-Unification-Base}$
$ML\text{-Functor-Instances}$
$ML\text{-Priorities}$
$Simps\text{-To}$
begin

Summary Unification modulo equations and combinators for unifiers.

Combinators ⟨ML⟩

Standard Unifiers ⟨ML⟩

Unification via Tactics ⟨ML⟩

Unification via Simplification ⟨ML⟩

Mixture of Unifiers ⟨ML⟩

declare [ucombine add = (Simplifier-Unification.Combine.eunif-data
(Simplifier-Unification.simp-unify
|> Unification-Combinator.norm-closed-unifier
(#norm-term Standard-Mixed-Unification.norms-first-higherp-first-comb-higher-unify)
|> Unification-Combinator.unifier-from-closed-unifier
|> K)
(Standard-Unification-Combine.default-metadata binding (simp-unif)))]

end
20 Unification Parsers

theory ML-Unification-Parsers
    imports
    ML-Parsing-Utils
begin

Summary  Common parsers needed for unification attributes, tactics, methods.
⟨ML⟩
end

20.1 Assumption Tactic

theory Unify-Assumption-Tactic
    imports
    ML-Functor-Instances
    ML-Unifiers
    ML-Unification-Parsers
begin

Summary  Assumption tactic and method with adjustable unifier.
⟨ML⟩

Examples  experiment
begin

lemma PROP P ⇒ PROP P
⟨proof⟩

lemma
    assumes h: ∀P. PROP P
    shows PROP P x
⟨proof⟩

schematic-goal \(∀x. \text{PROP P} (c :: 'a) ⇒ \text{PROP ?Y} (x :: 'a)\)
⟨proof⟩

schematic-goal a: \(\text{PROP ?P} (y :: 'a) ⇒ \text{PROP ?P} (?x :: 'a)\)
⟨proof⟩

schematic-goal
    \(\text{PROP ?P} (x :: 'a) ⇒ \text{PROP P (?x :: 'a)}\)
⟨proof⟩
Unlike assumption, `uassm` will not close the goal if the order of premises of the assumption and the goal are different. Compare the following two examples:

**lemma** \( \forall x. PROP D \implies (\forall y. PROP A y \implies PROP B x) \implies PROP C \implies PROP P x \)  
\hspace{1cm} \langle \text{proof} \rangle

**lemma** \( \forall x. PROP D \implies (\forall y. PROP A y \implies PROP B x) \implies PROP A x \implies PROP B x \)  
\hspace{1cm} \langle \text{proof} \rangle

### 20.2 Resolution Tactics

**theory** `Unify-Resolve-Tactics`

**imports**

- `Unify-Assumption-Tactic`
- `ML-Method-Utils`

**begin**

**Summary**

Resolution tactics and methods with adjustable unifier.

\( \langle ML \rangle \)

**Examples**

**experiment**

**begin**

**lemma**

assumes \( h: \forall x. PROP D x \implies PROP C x \)

shows \( \forall x. PROP A x \implies PROP B x \implies PROP C x \)  
\hspace{1cm} \langle \text{proof} \rangle

**lemma**

assumes \( h: PROP C x \)

shows \( PROP C x \)  
\hspace{1cm} \langle \text{proof} \rangle

**lemma**

assumes \( h: \forall x. PROP A x \implies PROP D x \)

shows \( \forall x. PROP A x \implies PROP B x \implies PROP C x \)  
\hspace{1cm} \langle \text{proof} \rangle

— use (r,e,d,f) to specify the resolution mode (resolution, elim, dest, forward)
You can specify how chained facts should be used. By default, \textit{urule} works like \textit{rule}: it uses chained facts to resolve against the premises of the passed rules.

\textbf{lemma}
\begin{itemize}
\item \textbf{assumes} \( h_1: \bigwedge x. (\text{PROP } F \ x \implies \text{PROP } E \ x) \implies \text{PROP } C \ x \)
\item \textbf{and} \( h_2: \bigwedge x. \text{PROP } F \ x \implies \text{PROP } E \ x \)
\item \textbf{shows} \( \bigwedge x. \text{PROP } A \ x \implies \text{PROP } B \ x \implies \text{PROP } C \ x \)
\end{itemize}
— Compare all of the following calls:

\begin{itemize}
\item \textbf{assumes} \( h_1: \bigwedge x \ y. \text{PROP } C \ y \implies \text{PROP } A \ x \implies \text{PROP } C \ x \)
\item \textbf{and} \( h_2: \bigwedge x \ y. \text{PROP } C \ x \implies \text{PROP } B \ x \implies \text{PROP } D \ x \)
\item \textbf{and} \( h_3: \bigwedge x \ y. \text{PROP } C \ x \)
\item \textbf{shows} \( \bigwedge x. \text{PROP } A \ x \implies \text{PROP } B \ x \implies \text{PROP } C \ x \)
\end{itemize}

\begin{itemize}
\item \textbf{assumes} \( h_1: \bigwedge x \ y. \text{PROP } C \ y \implies \text{PROP } A \ x \implies \text{PROP } C \ x \)
\item \textbf{and} \( h_2: \bigwedge x \ y. \text{PROP } C \ x \implies \text{PROP } B \ x \implies \text{PROP } D \ x \)
\item \textbf{and} \( h_3: \bigwedge x \ y. \text{PROP } C \ x \)
\item \textbf{shows} \( \bigwedge x. \text{PROP } A \ x \implies \text{PROP } B \ x \implies \text{PROP } C \ x \)
\end{itemize}

You can specify whether any or every rule must resolve against the goal:

\textbf{lemma}
\begin{itemize}
\item \textbf{assumes} \( h_1: \bigwedge x \ y. \text{PROP } C \ y \implies \text{PROP } D \ x \implies \text{PROP } C \ x \)
\item \textbf{and} \( h_2: \bigwedge x \ y. \text{PROP } C \ x \implies \text{PROP } D \ x \)
\item \textbf{and} \( h_3: \bigwedge x \ y. \text{PROP } C \ x \)
\item \textbf{shows} \( \bigwedge x. \text{PROP } A \ x \implies \text{PROP } B \ x \implies \text{PROP } C \ x \)
\end{itemize}

\textbf{end}

\section{20.3 Fact Tactic}

\textbf{theory} \textit{Unify-Fact-Tactic}
\begin{itemize}
\item \textbf{imports} \textit{Unify-Resolve-Tactics}
\end{itemize}
\begin{itemize}
\item \textbf{begin}
\end{itemize}
\textbf{Summary}  Fact tactic with adjustable unifier.

\textbf{Examples}  experiment
\begin{itemize}
\item \textbf{begin}
\item \textbf{lemma}
\end{itemize}

\textit{ML}
assumes \( h \): \( \forall x \ y. \ \text{PROP} \ P \ x \ y \)
shows \( \text{PROP} \ P \ x \ y \)
\hspace{1cm} \langle \text{proof} \rangle

lemma
assumes \( P \ y. \ \text{PROP} \ P \ y \ x \)
shows \( \text{PROP} \ P \ x \)
\hspace{1cm} \langle \text{proof} \rangle

lemma
assumes \( \forall x \ y. \ \text{PROP} \ A \ x \Rightarrow \text{PROP} \ B \ x \Rightarrow \text{PROP} \ P \ x \)
shows \( \forall x \ y. \ \text{PROP} \ A \ x \Rightarrow \text{PROP} \ B \ x \Rightarrow \text{PROP} \ P \ x \)
\hspace{1cm} \langle \text{proof} \rangle
end

21 Unification Tactics

theory Unification-Tactics
imports
  Unify-Assumption-Tactic
  Unify-Resolve-Tactics
  Unify-Fact-Tactic
begin

Summary  Tactics with adjustable unifiers.
end

22 Unification Attributes

theory Unification-Attributes
imports Unify-Resolve-Tactics
begin

Summary  OF attribute with adjustable unifier.
\langle ML \rangle

Examples  experiment
begin
lemma
assumes \( h1 \): \( \text{PROP} \ A \Rightarrow \text{PROP} \ D \) \Rightarrow \text{PROP} \ E \Rightarrow \text{PROP} \ C \)
assumes \( h2 \): \( \text{PROP} \ B \Rightarrow \text{PROP} \ D \)
and \( h3 \): \( \text{PROP} \ F \Rightarrow \text{PROP} \ E \)
shows \( \text{PROP} \ A \Rightarrow \text{PROP} \ B \) \Rightarrow \text{PROP} \ F \Rightarrow \text{PROP} \ C \)
\hspace{1cm} \langle \text{proof} \rangle
lemma
assumes h1: \(PROP\ A \implies PROP\ A\)
assumes h2: \(PROP\ A \implies PROP\ A\) \implies PROP\ B
shows PROP\ B
⟨proof⟩

lemma
assumes h1: \(\forall x\ y\ z.\ PROP\ P\ x\ y \implies PROP\ P\ y\ y \implies (PROP\ A \implies PROP\ A) \implies (PROP\ A \implies PROP\ B) \implies PROP\ C\)
and h2: \(\forall x\ y.\ PROP\ P\ x\ y\)
and h3 : PROP\ A \implies PROP\ A
and h4 : PROP\ D \implies PROP\ B
shows \((PROP\ A \implies PROP\ D) \implies PROP\ C\)
⟨proof⟩

lemma
assumes h1: \(\forall P.\ PROP\ P \implies PROP\ E\ P\ x\)
and h2: \(\forall P.\ PROP\ P\)
shows PROP\ E\ P\ x
⟨proof⟩

We can also specify the unifier to be used:

lemma
assumes h1: \(\forall P.\ PROP\ P \implies PROP\ E\)
and h2: \(\forall P.\ PROP\ P\)
shows PROP\ E
⟨proof⟩

end

23 Term Indexing

theory ML-Term-Index
imports  ML-Normalisations
begin

Summary Termin indexes signatures and implementations.
⟨ML⟩

end
24 Unification Hints

theory ML-Unification-Hints
imports
  ML-Generic-Data-Utils
  ML-Term-Index
  ML-Unifiers
  ML-Unification-Parsers
begin

Summary  A generalisation of unification hints, originally introduced in
[1]. We support a generalisation that

1. allows additional universal variables in premises
2. allows non-atomic left-hand sides for premises
3. allows arbitrary functions to perform the matching/unification of a
   hint with a disagreement pair.

   General shape of a hint: \( \forall y_1 \ldots y_n. (\forall x_1 \ldots x_{n-1}. \text{lhs}_1 \equiv \text{rhs}_1) \Rightarrow \ldots \Rightarrow (\forall x_1 \ldots x_{n_k}. \text{lhs}_k \equiv \text{rhs}_k) \Rightarrow \text{lhs} \equiv \text{rhs} \)

⟨ML⟩

Standard unification hints are accessible via \texttt{uhint}.

declare \[ \texttt{uhint where hint-preprocessor = \{Standard-Unification-Combine.obj-logic-hint-preprocessor @\{thm atomize-eq\}[symmetric]\} (Conv.rewr-conv @\{thm eq-eq-True\})}\]

Examples see \texttt{../Examples}.

end

25 Setup for HOL

theory ML-Unification-HOL-Setup
imports
  HOL.HOL
  ML-Unification-Hints
begin

lemma \texttt{eq-eq-True}: \( P \equiv (P \equiv \text{Trueprop True}) \) (proof)
declare \[ \texttt{uhint where hint-preprocessor = \{Unification-Hints-Base.obj-logic-hint-preprocessor @\{thm atomize-eq\}[symmetric]\} (Conv.rewr-conv @\{thm eq-eq-True\})}\]

lemma \texttt{eq-TrueI}: \( \text{PROP } P \Rightarrow \text{PROP } P \equiv \text{Trueprop True} \) (proof)
26 E-Unification Examples

theory E-Unification-Examples
  imports
    Main
    ML-Unification-HOL-Setup
    Unify-Fact-Tactic
begin

Summary  Sample applications of e-unifiers, methods, etc. introduced in this session.

experiment  begin

26.1 Using The Simplifier For Unification.

inductive-set even :: nat set where
  zero: 0 ∈ even |
  step: n ∈ even ⇒ Suc (Suc n) ∈ even

  Premises of the form SIMPS-TO-UNIF lhs rhs are solved by Simplifier_Unification. It first normalises lhs and then unifies the normalisation with rhs. See also ML-Unification.ML-Unification-HOL-Setup.

lemma [uhint where prio = Prio.LOW]: n ≠ 0 ⇒ PROP SIMPS-TO-UNIF (n − 1) m ⇒ n ≡ Suc m
   ⟨proof⟩

  By default, below unification methods use Standard_Mixed_Unification.first_higherp_first_comb_higher_unify, which is a combination of various practical unification algorithms.
\[ (\forall x. x + 4 = n) \implies \text{Suc } ?x = n \]

\textbf{lemma 6} \in \text{even} \\
\langle \text{proof} \rangle

\textbf{lemma} \ (220 + (80 - 2 * 2)) \in \text{even} \\
\langle \text{proof} \rangle

\textbf{lemma} \\
\text{assumes} \ [a,b,c] = [c,b,a] \\
\text{shows} \ [a] @ [b,c] = [c,b,a] \\
\langle \text{proof} \rangle

\textbf{lemma} \\
x \in \{(z, y, x) \cup S) \cap \{x} \\
\langle \text{proof} \rangle

\textbf{lemma} \\
(x + (y :: \text{nat}))^2 \leq x^2 + 2 \times x \times y + y^2 + 4 \times y + x - y \\
\langle \text{proof} \rangle

\textbf{lemma} \\
\text{assumes} \ \bigwedge s. P (\text{Suc (Suc 0)}) (s(x := (1 :: \text{nat}), x := 1 + 1 \times 4 - 3)) \\
\text{shows} \ P 2 (s(x := 2)) \\
\langle \text{proof} \rangle

26.2 Providing Canonical Solutions With Unification Hints

\textbf{lemma} \ [\text{uhint}]: xs \equiv [] \implies \text{length } xs \equiv 0 \ \langle \text{proof} \rangle

\textbf{schematic-goal} \ \text{length } ?xs = 0 \\
\langle \text{proof} \rangle

\textbf{lemma} \ [\text{uhint}]: (n :: \text{nat}) \equiv m \implies n - m \equiv 0 \ \langle \text{proof} \rangle

\textbf{schematic-goal} \ n - ?m = (0 :: \text{nat}) \\
\langle \text{proof} \rangle

The following fails because, by default, Standard\_Unification\_Hints\_try\_hints uses the higher-order pattern unifier to unify hints against a given disagreement pair, and 0 :: 'a cannot be higher-order pattern unified with length \[]. The unification of the hint requires the use of yet another hint, namely length xs = 0 (cf. above).

\textbf{schematic-goal} \ n - ?m = \text{length } [] \\
-- by (ufact refl) \\
\langle \text{proof} \rangle

There are two ways to fix this:

1. We allow the recursive uses of unification hints when searching for suitable unification hints.
2. We use a different unification hint that the recursive use of hints explicit.

Solution 1: recursive usages of hints. Warning: such recursive applications easily loop.

schematic-goal $n - ?m = \text{length} []$
(proof)

Solution 2: make the recursion explicit in the hint.

lemma [uhint]: $k \equiv 0 \Longrightarrow (n :: \text{nat}) \equiv m \Longrightarrow n - m \equiv k$ (proof)

schematic-goal $n - ?m = \text{length} []$
(proof)

26.3 Strengthen Unification With Unification Hints

lemma
assumes [uhint]: $n = m$
shows $n - m = (0 :: \text{nat})$
(proof)

lemma
assumes $x = y$
shows $y = x$
(proof)

Unfolding definitions. definition $\text{mysuc} \ n = \text{Suc} \ n$

lemma
assumes $\forall m. \text{Suc} \ n > \text{mysuc} \ m$
shows $\text{mysuc} \ n > \text{Suc} \ 3$
(proof)

Discharging meta implications with object-level implications lemma [uhint]:

$\text{Trueprop} \ A \equiv A' \Longrightarrow \text{Trueprop} \ B \equiv B' \Longrightarrow \text{Trueprop} \ (A \rightarrow B) \equiv (PROP \ A' \Longrightarrow PROP \ B')$
(proof)

lemma
assumes $A \rightarrow (B \rightarrow C) \rightarrow D$
shows $A \ equivalence (B \ equivalence C) \ equivalence D$
(proof)

26.4 Better Control Over Meta Variable Instantiations

Consider the following type-inference problem.

schematic-goal
assumes \( \text{app-typeI}: \bigwedge x. (\bigwedge x. \mathit{ArgT} x \rightarrow \mathit{DomT} x (f x)) \rightarrow \mathit{ArgT} x \rightarrow \mathit{DomT} x (f x) \)
and \( \text{f-type}: \bigwedge x. \mathit{ArgT} x \rightarrow \mathit{DomT} x (f x) \)
and \( \text{x-type}: \mathit{ArgT} x \)
shows \( \forall T (f x) \)
⟨proof⟩

end
end

27 Examples: Reification Via Unification Hints

theory Unification-Hints-Reification-Examples
imports
  HOL.Rat
  ML-Unification-HOL-Setup
  Unify-Fact-Tactic
begin

Summary Reification via unification hints. For an introduction to unification hints refer to \([1]\). We support a generalisation of unification hints as described in \textit{ML-Unification.ML-Unification-Hints}.

27.1 Setup

One-time setup to obtain a unifier with unification hints for the purpose of reification. We could also simply use the standard unification hints \textit{uhint}, but having separate instances is a cleaner approach.
⟨ML⟩
Premises of hints should again be unified by the reification unifier.
declare [[reify-uhint where prems-unifier = reify-unify]]

27.2 Formulas with Quantifiers and Environment

The following example is taken from \textit{HOL-Library.Reflection_Examples}. It is recommended to compare the approach presented here with the reflection tactic presented in said theory.

\textbf{datatype} form =
  TrueF
| FalseF
| Less nat nat
| And form form
| Or form form
| Neg form
| ExQ form

primrec interp :: form ⇒ ('a::ord) list ⇒ bool
where
interp TrueF vs ←→ True
interp FalseF vs ←→ False
interp (Less i j) vs ←→ vs ! i < vs ! j
interp (And f1 f2) vs ←→ interp f1 vs ∧ interp f2 vs
interp (Or f1 f2) vs ←→ interp f1 vs ∨ interp f2 vs
interp (Neg f) vs ←→ ¬ interp f vs
interp (ExQ f) vs ←→ (∃ v. interp f (v # vs))

Reification with unification and recursive hint unification for conclusion
The following illustrates how to use the equations interp TrueF ?vs = True
interp FalseF ?vs = False
interp (And ?f1.0 ?f2.0) ?vs = (interp ?f1.0 ?vs ∧ interp ?f2.0 ?vs)
interp (Or ?f1.0 ?f2.0) ?vs = (interp ?f1.0 ?vs ∨ interp ?f2.0 ?vs)
interp (Neg ?f) ?vs = (¬ interp ?f ?vs)
interp (ExQ ?f) ?vs = (∃ v. interp ?f (v # ?vs)) directly as unification

Hints for list lookup.
declare List.nth-Cons-Suc[reify-uhint where prio = Prio.LOW]
and List.nth-Cons-0[reify-uhint]

Hints to reify formulas of type bool into formulas of type form.
declare interp.simps[reify-uhint]

We have to allow the hint unifier to recursively look for hints during
unification of the hint’s conclusion.
declare [[reify-uhint where concl-unifier = reify-unify]]

schematic-goal
interp ?f (?vs :: ('a :: ord) list) = (∃(x :: 'a). x < y ∧ ¬(∃(z :: 'a). v < z ∨ ¬False))
(proof)

While this all works nicely if set up correctly, it can be rather difficult
to understand and debug the recursive unification process for a hint’s con-
clusion. In the next paragraph, we present an alternative that is closer to the examples presented in the original unification hints paper [1].

end

Reification with matching without recursion for conclusion  We disallow the hint unifier to recursively look for hints while unifying the conclusion; instead, we only allow the hint unifier to match the hint’s conclusion against the disagreement terms.

declare [[reify-uhint where concl-unifier = Higher-Order- pattern-Unification.match]]

However, this also means that we now have to write our hints such that the hint’s conclusion can successfully be matched against the disagreement terms. In particular, the disagreement terms may still contain meta variables that we want to instantiate with the help of the unification hints. Essentially, a hint then describes a canonical instantiation for these meta variables.

experiment

begin

lemma [reify-uhint where prio = Prio.LOW]:
  \[ n \equiv \text{Suc } n' \implies vs \equiv v \# vs' \implies vs' ! n' \equiv x \implies vs ! n \equiv x \]
  ⟨proof⟩

lemma [reify-uhint]:
  \[ n \equiv 0 \implies vs \equiv x \# vs' \implies vs ! n \equiv x \]
  ⟨proof⟩

lemma [reify-uhint]:
  \[ e \equiv \text{ExQ } f; \ \land v. \ \text{interp } f \ (v \# vs) \equiv P \ v \]\n  \[ \implies \text{interp } e \ vs \equiv \exists v. \ P \ v \]
  ⟨proof⟩

lemma [reify-uhint]:
  \[ e \equiv \text{Less } i \ j; \ \text{interp } f1 \ vs \equiv r1; \ \text{interp } f2 \ vs \equiv r2 \]\n  \[ \implies \text{interp } e \ vs \equiv r1 \land r2 \]
  ⟨proof⟩

lemma [reify-uhint]:
  \[ e \equiv \text{Neg } f \]\n  \[ \implies \text{interp } e \ vs \equiv \neg r \]
  ⟨proof⟩

lemma [reify-uhint]:
  \[ e \equiv \text{TrueF} \]\n  \[ \implies \text{interp } e \ vs \equiv \text{True} \]
  ⟨proof⟩

lemma [reify-uhint]:
  \[ e \equiv \text{FalseF} \]\n  \[ \implies \text{interp } e \ vs \equiv \text{False} \]
  ⟨proof⟩

schematic-goal
  \[ \text{interp } f \ f (\forall vs :: ('a :: ord) \ list) = (\exists (x :: 'a). \ x < y \land \neg(\exists (z :: 'a). \ v < z \lor \neg(\text{False})) \]
  ⟨proof⟩

end

The next examples are modification from [1].

27.3 Simple Arithmetic

datatype add-expr = Var int | Add add-expr add-expr
fun eval-add-expr :: add-expr ⇒ int where
    eval-add-expr (Var i) = i
| eval-add-expr (Add ex1 ex2) = eval-add-expr ex1 + eval-add-expr ex2

lemma eval-add-expr-Var [reify-ahint where prio = Prio.LOW]:
    e ≡ Var i ⇒ eval-add-expr e ≡ i ⟨proof⟩

lemma eval-add-expr-add [reify-ahint]:
    e ≡ Add e1 e2 ⇒ eval-add-expr e1 ≡ m ⇒ eval-add-expr e2 ≡ n ⇒ eval-add-expr e ≡ m + n ⟨proof⟩

⟨ML⟩
schematic-goal eval-add-expr ?e
    = (1 + (2 + 7) :: int) ⟨proof⟩

27.4 Arithmetic with Environment
datatype mul-expr =
    Unit
| Var nat
| Mul mul-expr mul-expr
| Inv mul-expr

fun eval-mul-expr :: mul-expr × rat list ⇒ rat where
    eval-mul-expr (Unit, Γ) = 1
| eval-mul-expr (Var i, Γ) = Γ ! i
| eval-mul-expr (Mul e1 e2, Γ) = eval-mul-expr e1, Γ) * eval-mul-expr e2, Γ)
| eval-mul-expr (Inv e, Γ) = inverse (eval-mul-expr e, Γ))

    Split e into an expression and an environment.
lemma [reify-ahint where prio = Prio.VERY-LOW]:
    e ≡ (e1, Γ) ⇒ eval-mul-expr e1, Γ) ≡ n ⇒ eval-mul-expr e ≡ n ⟨proof⟩

Hints for environment lookup.
lemma [reify-ahint where prio = Prio.LOW]:
    e ≡ Var (Suc p) ⇒ Γ ≡ s # Δ ⇒ n ≡ eval-mul-expr (Var p, Δ) ⇒ eval-mul-expr e, Γ) ≡ n ⟨proof⟩

lemma [reify-ahint]: e ≡ Var 0 ⇒ Γ ≡ n # Θ ⇒ eval-mul-expr e, Γ) ≡ n ⟨proof⟩

lemma [reify-ahint]: e1 ≡ Inv e2 ⇒ n ≡ eval-mul-expr e2, Γ) ⇒ eval-mul-expr e1, Γ) ≡ inverse n
    e ≡ Mul e1 e2 ⇒ m ≡ eval-mul-expr e1, Γ) ⇒ n ≡ eval-mul-expr e2, Γ) ⇒ eval-mul-expr e, Γ) ≡ m * n
\[ e \equiv \text{Unit} \implies \text{eval-mul-expr} \ (e, \Gamma) \equiv 1 \]

\[ \langle \text{proof} \rangle \]

\[ \langle \text{ML} \rangle \]

\textbf{schematic-goal} \ \text{eval-mul-expr} \ ?e = (1 \ast \text{inverse} \ 3 \ast 5 :: \text{rat})

\[ \langle \text{proof} \rangle \]

\text{end}

\textbf{References}
