Unification Utilities for Isabelle/ML

Kevin Kappelmann

March 17, 2025

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 λ-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 $n + 1 \equiv ?m + Suc\, 0$ 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.
4. Simplifier integration for e-unifiers.
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

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 2 https://logging.apache.org/log4j/2.x/manual/customloglevels.html.

⟨*ML*⟩

end

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

⟨*ML*⟩

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/surpress 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.

declare [[*ML-map-context* ‹*Logger.set-msg-filters Logger.root* (*match-string Third*)›]]
⟨*ML*⟩
declare [[*ML-map-context* ‹*Logger.set-msg-filters Logger.root* (*K true*)›]]

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

⟨*ML*⟩

To set up (local) loggers outside ML environments, *ML-Unification.Setup-Result-Commands* contains two commands, **setup-result** and **local-setup-result**.

experiment
begin
local-setup-result *local-logger* = ‹*Logger.new-logger Logger.root Local*›

⟨*ML*⟩
end

local-logger is no longer available. The follow thus does not work:

Let us create another logger in the global context.

setup-result *some-logger* = ‹*Logger.new-logger Logger.root Some-Logger*›
⟨*ML*⟩

Let us delete it again.

declare [[*ML-map-context* ‹*Logger.delete-logger some-logger*›]]

The logger can no longer be found in the logger hierarchy

⟨*ML*⟩

end

4 ML Attribute Utils

theory *ML-Attribute-Utils*
imports
Pure
begin

Summary Utilities for attributes.

⟨*ML*⟩

end

5 ML Conversion Utils

theory *ML-Conversion-Utils*
imports
Pure
begin

Summary Utilities for conversions.

lemma *meta-eq-symmetric*: $(A \equiv B) \equiv (B \equiv A)$
⟨*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

11 Priorities

theory *ML-Priorities*
imports *ML-Parsing-Utils*
begin

Summary Priorities for ML tactics.

⟨*ML*⟩

end

12 ML-Normalisations

theory *ML-Normalisations*
  imports
    ML-Conversion-Utils
begin

Summary Normalisation functions for terms, types, and theorems.

⟨*ML*⟩

end

13 ML-Binders

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

Summary Binders for ML.

⟨*ML*⟩

end

14 ML Term Utils

theory *ML-Term-Utils*
  imports *ML-Binders*
begin

Summary Utilities for terms.

⟨*ML*⟩

end

15 ML Theorem Utils

theory *ML-Theorem-Utils*
  imports *ML-Logger*
begin

Summary Utilities for theorems.

⟨*ML*⟩

end

16 ML Unification Basics

theory *ML-Unification-Base*
imports
ML-Logger
ML-Binders
ML-Normalisations
ML-Theorem-Utils
SpecCheck.SpecCheck-Show
begin

Summary Basic definitions and utilities for unification algorithms.

⟨*ML*⟩

end

17 ML Tactic Utils

theory *ML-Tactic-Utils*
imports
ML-Logger
ML-Term-Utils
ML-Conversion-Utils
ML-Unification-Base
begin

Summary Utilities for tactics.

⟨*ML*⟩

end

18 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

19 ML Unifiers

theory *ML-Unifiers-Base*
imports
ML-Unification-Base
ML-Tactic-Utils
begin

Summary Unification modulo equations and combinators for unifiers.

Combinators ⟨*ML*⟩

Type Unifiers ⟨*ML*⟩

Standard Unifiers ⟨*ML*⟩

Unification via Tactics ⟨*ML*⟩

end

20 Simps To

theory *Simps-To*
imports
ML-Unifiers-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 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* $\Longrightarrow$ $s \equiv t$ ⟨*proof*⟩

⟨*ML*⟩

Using Simplification On Left Term Followed By Unification **definition** *SIMPS-TO-UNIF s t* $\equiv$ $(s \equiv t)$

Prevent simplification

lemma *SIMPS-TO-UNIF-cong* [*cong*]: *SIMPS-TO-UNIF s t* $\equiv$ *SIMPS-TO-UNIF s t* ⟨*proof*⟩

lemma *SIMPS-TO-UNIF-eq*: *SIMPS-TO-UNIF s t* $\equiv$ $(s \equiv t)$ ⟨*proof*⟩

lemma *SIMPS-TO-UNIFI*: *PROP SIMPS-TO s s'* $\Longrightarrow$ $s' \equiv t$ $\Longrightarrow$ *PROP SIMPS-TO-UNIF s t*
⟨*proof*⟩
lemma *SIMPS-TO-UNIFD*: *PROP SIMPS-TO-UNIF s t* $\Longrightarrow$ $s \equiv t$
⟨*proof*⟩

⟨*ML*⟩

Examples **experiment**
begin

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

end

end

theory *ML-Unifiers*
imports
ML-Functor-Instances
ML-Priorities
ML-Unifiers-Base
Simps-To
begin

Summary More unifiers.

Derived Unifiers ⟨*ML*⟩

Unification via Simplification **lemma** *eq-if-SIMPS-TO-UNIF-if-SIMPS-TO-UNIF*:
assumes *PROP SIMPS-TO-UNIF t t′*
and *PROP SIMPS-TO-UNIF s t′*
shows *s ≡ t*
⟨*proof*⟩

⟨*ML*⟩

Combining Unifiers ⟨*ML*⟩

Mixture of Unifiers ⟨*ML*⟩

```
declare [[ucombine add = ‹Standard-Unification-Combine.eunif-data
  (Var-Higher-Order-Pattern-Unification.e-unify Unification-Combinator.fail-unify
  |> Unification-Combinator.norm-unifier
    (Unification-Util.inst-norm-term′
      Standard-Mixed-Unification.norms-first-higherp-decomp-comb-higher-unify)
  |> K)
  (Standard-Unification-Combine.metadata binding ‹var-hop-unif› Prio.HIGH)›]]

declare [[ucombine add = ‹
  let
    open Term-Normalisation
    (*ignore changes of schematic variables to avoid loops due to index−raising of
some tactics*)
    val eq-beta-eta-dummy-vars = apply2 (beta-eta-short #> dummy-vars) #> op
aconv
    val unif = Standard-Mixed-Unification.first-higherp-decomp-comb-higher-unify
    val norms = Standard-Mixed-Unification.norms-first-higherp-decomp-comb-higher-unify
  in
    Standard-Unification-Combine.eunif-data
      (Simplifier-Unification.simp-unify-progress eq-beta-eta-dummy-vars
        (Simplifier-Unification.simp-unify norms unif norms)
        (Unification-Util.inst-norm-term′ norms)
        unif
      |> Type-Unification.e-unify Unification-Util.unify-types
      |> K)
      (Standard-Unification-Combine.default-metadata binding ‹simp-unif›)
  end›]]
```

end

21 Unification Parsers

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

Summary Common parsers needed for unification attributes, tactics, methods.

⟨*ML*⟩

end

21.1 Assumption Tactic

theory *Unify-Assumption-Tactic-Base*
imports
ML-Functor-Instances
ML-Tactic-Utils
ML-Unification-Parsers
begin

Summary Assumption tactic and method with adjustable unifier.

⟨*ML*⟩

end

theory *Unify-Assumption-Tactic*
imports
Unify-Assumption-Tactic-Base
ML-Unifiers
begin

Summary Setup of assumption tactic and examples.

⟨*ML*⟩

Examples **experiment**
begin

lemma $PROP\ P \Longrightarrow PROP\ P$
⟨*proof*⟩

lemma
assumes h: $\bigwedge P.\ PROP\ P$
shows $PROP\ P\ x$
⟨*proof*⟩

schematic-goal $\bigwedge x.\ PROP\ P\ (c :: 'a) \Longrightarrow PROP\ ?Y\ (x :: 'a)$
⟨*proof*⟩

schematic-goal a: $PROP\ ?P\ (y :: 'a) \Longrightarrow PROP\ ?P\ (?x :: 'a)$
⟨*proof*⟩

schematic-goal
$PROP\ ?P\ (x :: 'a) \Longrightarrow PROP\ P\ (?x :: 'a)$
⟨*proof*⟩

schematic-goal
$\bigwedge x.\ PROP\ D \Longrightarrow (\bigwedge P\ y.\ PROP\ P\ y\ x) \Longrightarrow PROP\ C \Longrightarrow PROP\ P\ x$
⟨*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 $\bigwedge x.\ PROP\ D \Longrightarrow (\bigwedge y.\ PROP\ A\ y \Longrightarrow PROP\ B\ x) \Longrightarrow PROP\ C \Longrightarrow PROP\ A\ x \Longrightarrow PROP\ B\ x$
⟨*proof*⟩

lemma $\bigwedge x.\ PROP\ D \Longrightarrow (\bigwedge y.\ PROP\ A\ y \Longrightarrow PROP\ B\ x) \Longrightarrow PROP\ A\ x \Longrightarrow PROP\ C \Longrightarrow PROP\ B\ x$
⟨*proof*⟩

end

end

21.2 Resolution Tactics

theory *Unify-Resolve-Tactics-Base*
imports
Unify-Assumption-Tactic-Base
ML-Unifiers-Base
ML-Method-Utils
begin

Summary Resolution tactics and methods with adjustable unifier.

⟨*ML*⟩

end

21.3 Resolution Tactics

theory *Unify-Resolve-Tactics*
imports
Unify-Resolve-Tactics-Base
ML-Unifiers
begin

Summary Setup of resolution tactics and examples.

⟨*ML*⟩

Examples **experiment**
begin

lemma
assumes *h*: $\bigwedge x.$ *PROP D x* $\Longrightarrow$ *PROP C x*
shows $\bigwedge x.$ *PROP A x* $\Longrightarrow$ *PROP B x* $\Longrightarrow$ *PROP C x*
⟨*proof*⟩

lemma
assumes *h*: *PROP C x*
shows *PROP C x*
⟨*proof*⟩

lemma
assumes *h*: $\bigwedge x.$ *PROP A x* $\Longrightarrow$ *PROP D x*
shows $\bigwedge x.$ *PROP A x* $\Longrightarrow$ *PROP B x* $\Longrightarrow$ *PROP C x*
— use (r,e,d,f) to specify the resolution mode (resolution, elim, dest, forward)
⟨*proof*⟩

lemma
assumes *h1*: $\bigwedge x.$ *PROP A x* $\Longrightarrow$ *PROP D x*
and *h2*: $\bigwedge x.$ *PROP D x* $\Longrightarrow$ *PROP E x*
shows $\bigwedge x.$ *PROP A x* $\Longrightarrow$ *PROP B x* $\Longrightarrow$ *PROP C x*
— use (rr,re,rd,rf) to use repetition; in particular: (*urule* (*rr*)) $\simeq$ *intro*
⟨*proof*⟩

You can specify how chained facts should be used. By default, *urule* works like *rule*: it uses chained facts to resolve against the premises of the passed rules.

lemma
assumes *h1*: $\bigwedge x.$ (*PROP F x* $\Longrightarrow$ *PROP E x*) $\Longrightarrow$ *PROP C x*
and *h2*: $\bigwedge x.$ *PROP F x* $\Longrightarrow$ *PROP E x*
shows $\bigwedge x.$ *PROP A x* $\Longrightarrow$ *PROP B x* $\Longrightarrow$ *PROP C x*
— Compare all of the following calls:

⟨*proof*⟩

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

lemma
assumes *h1*: $\bigwedge x\ y.$ *PROP C y* $\Longrightarrow$ *PROP D x* $\Longrightarrow$ *PROP C x*
and *h2*: $\bigwedge x\ y.$ *PROP C x* $\Longrightarrow$ *PROP D x*
and *h3*: $\bigwedge x\ y.$ *PROP C x*
shows $\bigwedge x.$ *PROP A x* $\Longrightarrow$ *PROP B x* $\Longrightarrow$ *PROP C x*
⟨*proof*⟩

lemma
assumes *h1*: $\bigwedge x\ y.\ PROP\ C\ y \Longrightarrow PROP\ A\ x \Longrightarrow PROP\ C\ x$
and *h2*: $\bigwedge x\ y.\ PROP\ C\ x \Longrightarrow PROP\ B\ x \Longrightarrow PROP\ D\ x$
and *h3*: $\bigwedge x\ y.\ PROP\ C\ x$
shows $\bigwedge x.\ PROP\ A\ x \Longrightarrow PROP\ B\ x \Longrightarrow PROP\ C\ x$
⟨*proof*⟩

end

end

21.4 Fact Tactic

theory *Unify-Fact-Tactic-Base*
imports
Unify-Resolve-Tactics-Base
begin

Summary Fact tactic with adjustable unifier.

⟨*ML*⟩

end

21.5 Fact Tactic

theory *Unify-Fact-Tactic*
imports
Unify-Fact-Tactic-Base
ML-Unifiers
begin

Summary Setup of fact tactic and examples.

⟨*ML*⟩

Examples experiment
begin
lemma
assumes *h*: $\bigwedge x\ y.\ PROP\ P\ x\ y$
shows $PROP\ P\ x\ y$
⟨*proof*⟩

lemma
assumes $\bigwedge P\ y.\ PROP\ P\ y\ x$
shows $PROP\ P\ x$
⟨*proof*⟩

lemma

assumes $\bigwedge x\ y.\ PROP\ A\ x \Longrightarrow PROP\ B\ x \Longrightarrow PROP\ P\ x$
shows $\bigwedge x\ y.\ PROP\ A\ x \Longrightarrow PROP\ B\ x \Longrightarrow PROP\ P\ x$
⟨*proof*⟩
end

end

22 Unification Tactics

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

Summary Tactics with adjustable unifiers.

end

23 Unification Attributes

theory *Unification-Attributes-Base*
imports *Unify-Resolve-Tactics-Base*
begin

Summary OF attribute with adjustable unifier.

⟨*ML*⟩

end

theory *Unification-Attributes*
imports
Unification-Attributes-Base
ML-Unifiers
begin

Summary Setup of OF attribute with adjustable unifier.

⟨*ML*⟩

Examples **experiment**
begin
lemma
assumes *h1*: $(PROP\ A \Longrightarrow PROP\ D) \Longrightarrow PROP\ E \Longrightarrow PROP\ C$
assumes *h2*: $PROP\ B \Longrightarrow PROP\ D$
and *h3*: $PROP\ F \Longrightarrow PROP\ E$
shows $(PROP\ A \Longrightarrow PROP\ B) \Longrightarrow PROP\ F \Longrightarrow PROP\ C$

⟨*proof*⟩

lemma
assumes *h1*: $(PROP\ A \Longrightarrow PROP\ A)$
assumes *h2*: $(PROP\ A \Longrightarrow PROP\ A) \Longrightarrow PROP\ B$
shows $PROP\ B$
⟨*proof*⟩

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

lemma
assumes *h1*: $\bigwedge P\ x.\ PROP\ P\ x \Longrightarrow PROP\ E\ P\ x$
and *h2*: $PROP\ P\ x$
shows $PROP\ E\ P\ x$
⟨*proof*⟩

We can also specify the unifier to be used:

lemma
assumes *h1*: $\bigwedge P.\ PROP\ P \Longrightarrow PROP\ E$
and *h2*: $\bigwedge P.\ PROP\ P$
shows $PROP\ E$
⟨*proof*⟩

end

end

24 Term Indexing

theory *ML-Term-Index*
imports
ML-Normalisations
begin

Summary Termin indexes signatures and implementations.

⟨*ML*⟩

end

25 Unification Hints

theory *ML-Unification-Hints-Base*
  imports
    ML-Conversion-Utils
    ML-Functor-Instances
    ML-Generic-Data-Utils
    ML-Priorities
    ML-Term-Index
    ML-Tactic-Utils
    ML-Term-Utils
    ML-Unifiers-Base
    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: $\bigwedge y1 ... yn.\ (\bigwedge x1 ... xn1.\ lhs1 \equiv rhs1) \Longrightarrow ... \Longrightarrow (\bigwedge x1 ... xnk.\ lhsk \equiv rhsk) \Longrightarrow lhs \equiv rhs$

⟨*ML*⟩

end

26 Unification Hints

theory *ML-Unification-Hints*
  imports
    ML-Unification-Hints-Base
    ML-Unifiers
begin

Summary Setup of unification hints.

We now set up two unifiers using unification hints. The first one allows for recursive applications of unification hints when unifying a hint's conclusion $lhs \equiv rhs$ with a goal $lhs' \equiv rhs'$. The second disallows recursive applications of unification hints. Recursive applications have to be made explicit in the hint itself (cf. `../Examples`).

While the former can be convenient for local hint registrations and quick developments, it is advisable to use the second for global hints to avoid unexpected looping behaviour.

⟨*ML*⟩

Standard unification hints using `Standard_Mixed_Unification.first_higherp_decomp_comb_h` when looking for hints are accessible via *rec-uhint*.

Note: when we retrieve a potential unification hint with conclusion $lhs \equiv rhs$ for a goal $lhs' \equiv rhs'$, we consider those hints whose lhs or rhs potentially higher-order unifies with lhs' or rhs' *without using hints*. For otherwise, any hint $lhs \equiv rhs$ applied to a goal $rhs \equiv lhs$ leads to an immediate loop. The retrieval can be further restricted and modified by via the retrieval setting of *rec-uhint*.

```
declare [[ucombine add = ‹Standard-Unification-Combine.eunif-data
  (Standard-Unification-Hints-Rec.try-hints
  |> Unification-Combinator.norm-unifier
    (Unification-Util.inst-norm-term'
      Standard-Mixed-Unification.norms-first-higherp-decomp-comb-higher-unify)
  |> K)
  (Standard-Unification-Combine.metadata Standard-Unification-Hints-Rec.binding
Prio.LOW)›]]
```

⟨*ML*⟩

```
declare [[uhint where concl-unifier = ‹fn binders =>
  Standard-Unification-Combine.delete-eunif-data
   (Standard-Unification-Combine.metadata Standard-Unification-Hints.binding (Prio.inc
Prio.LOW))
  (*TODO: should we also remove the recursive hint unifier here? time will tell...*)
  (*#> Standard-Unification-Combine.delete-eunif-data
   (Standard-Unification-Combine.metadata Standard-Unification-Hints-Rec.binding
Prio.LOW)*)
  |> Context.proof-map
 #> Standard-Mixed-Unification.first-higherp-decomp-comb-higher-unify binders›]]
```

Standard unification hints using `Standard_Mixed_Unification.first_higherp_decomp_comb_h` when looking for hints, without using fallback list of unifiers, are accessible via *uhint*.

Note: there will be no recursive usage of unification hints when searching for potential unification hints in this case. See also `../Examples`.

```
declare [[ucombine add = ‹Standard-Unification-Combine.eunif-data
  (Standard-Unification-Hints.try-hints
  |> Unification-Combinator.norm-unifier
    (Unification-Util.inst-norm-term'
      Standard-Mixed-Unification.norms-first-higherp-decomp-comb-higher-unify)
  |> K)
  (Standard-Unification-Combine.metadata Standard-Unification-Hints.binding (Prio.inc
Prio.LOW))›]]
```

Examples see `../Examples`.

end

27 Setup for HOL

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

lemma *eq-eq-True*: $P \equiv (P \equiv \mathit{Trueprop\ True})$ ⟨*proof*⟩
declare [[*uhint* **where** *hint-preprocessor* = ‹*Unification-Hints-Base.obj-logic-hint-preprocessor*
  @{*thm atomize-eq*[*symmetric*]} (*Conv.rewr-conv* @{*thm eq-eq-True*})›]]
and [[*rec-uhint* **where** *hint-preprocessor* = ‹*Unification-Hints-Base.obj-logic-hint-preprocessor*
  @{*thm atomize-eq*[*symmetric*]} (*Conv.rewr-conv* @{*thm eq-eq-True*})›]]

lemma *eq-TrueI*: $\mathit{PROP}\ P \Longrightarrow \mathit{PROP}\ P \equiv \mathit{Trueprop\ True}$ ⟨*proof*⟩
declare [[*ucombine add* = ‹*Standard-Unification-Combine.eunif-data*
  (*Simplifier-Unification.SIMPS-TO-unify* @{*thm eq-TrueI*}
  |> *Unification-Combinator.norm-unifier* (*Unification-Util.inst-norm-term′*
    Standard-Mixed-Unification.norms-first-higherp-decomp-comb-higher-unify)
  |> *K*)
 (*Standard-Unification-Combine.metadata* ***binding*** ‹*SIMPS-TO-unif*› *Prio.HIGH*)›]]

declare [[*ucombine add* = ‹
  let
    open Term-Normalisation
    (∗*ignore changes of schematic variables to avoid loops due to index−raising of*
some tactics∗)
    val eq-beta-eta-dummy-vars = *apply2* (*beta-eta-short* #> *dummy-vars*) #> *op*
aconv
  in
    Standard-Unification-Combine.eunif-data
      (*Simplifier-Unification.simp-unify-progress eq-beta-eta-dummy-vars*
        (*Simplifier-Unification.SIMPS-TO-UNIF-unify* @{*thm eq-TrueI*}
          Standard-Mixed-Unification.norms-first-higherp-decomp-comb-higher-unify)
        (*Unification-Util.inst-norm-term′*
          Standard-Mixed-Unification.norms-first-higherp-decomp-comb-higher-unify)
        Standard-Mixed-Unification.first-higherp-decomp-comb-higher-unify
      |> *K*)
        (*Standard-Unification-Combine.metadata* ***binding*** ‹*SIMPS-TO-UNIF-unif*›
Prio.HIGH)
  end›]]

end

28 E-Unification Examples

theory *E-Unification-Examples*
  imports
    Main
    ML-Unification-HOL-Setup
    Unify-Assumption-Tactic
    Unify-Fact-Tactic
    Unify-Resolve-Tactics
begin

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

experiment
begin

28.1 Using The Simplifier For Unification.

inductive-set *even* :: *nat set* **where**
zero: $0 \in even$ |
step: $n \in even \Longrightarrow Suc\ (Suc\ n) \in even$

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*]: $n \neq 0 \Longrightarrow PROP\ SIMPS\text{-}TO\text{-}UNIF\ (n - 1)\ m \Longrightarrow n \equiv Suc\ m$
  ⟨*proof*⟩

By default, below unification methods use `Standard_Mixed_Unification.first_higherp_decom` which is a combination of various practical unification algorithms.

schematic-goal $(\bigwedge x.\ x + 4 = n) \Longrightarrow Suc\ ?x = n$
  ⟨*proof*⟩

lemma $6 \in even$
  ⟨*proof*⟩

lemma $(220 + (80 - 2 * 2)) \in even$
  ⟨*proof*⟩

lemma
  assumes $[a,b,c] = [c,b,a]$
  shows $[a]\ @\ [b,c] = [c,b,a]$
  ⟨*proof*⟩

lemma $x \in (\{z, y, x\} \cup S) \cap \{x\}$
  ⟨*proof*⟩

schematic-goal $(x + (y :: nat))\hat{\ }2 \leq x\hat{\ }2 + 2{*}x{*}y + y\hat{\ }2 + 4 * y + x - y$
⟨*proof*⟩

lemma
assumes $\bigwedge s.\ P\ (Suc\ (Suc\ 0))\ (s(x := (1 :: nat),\ x := 1 + 1 * 4 - 3))$
shows $P\ 2\ (s(x := 2))$
⟨*proof*⟩

28.2 Providing Canonical Solutions With Unification Hints

lemma *sub-self-eq-zero* [*uhint*]: $(n :: nat) - n \equiv 0$ ⟨*proof*⟩

schematic-goal $n - ?m = (0 :: nat)$
⟨*proof*⟩

The following example shows a non-trivial interplay of the simplifier and unification hints: Using just unification, the hint $?n - ?n \equiv 0$ is not applicable in the following example since 0 cannot be unified with *length* []. However, the simplifier can rewrite *length* [] to 0 and the hint can then be applied.

schematic-goal $n - ?m = length$ []
⟨*proof*⟩

There are also two ways to solve this using only unification hints:

1. We allow the recursive use of unification hints when unifying $?n - ?n \equiv 0$ and our goal and register *length* [] $= 0$ as an additional hint.
2. We use an alternative for $?n - ?n \equiv 0$ that makes the recursive use of unification hints explicit and register *length* [] $= 0$ as an additional hint.

lemma *length-nil-eq* [*uhint*]: *length* [] $= 0$ ⟨*proof*⟩

Solution 1: we can use *rec-uhint* for recursive usages of hints. Warning: recursive hint applications easily loop.

schematic-goal $n - ?m = length$ []
⟨*proof*⟩

Solution 2: make the recursion explicit in the hint.

lemma [*uhint*]: $k \equiv 0 \Longrightarrow (n :: nat) \equiv m \Longrightarrow n - m \equiv k$ ⟨*proof*⟩

schematic-goal $n - ?m = length$ []
⟨*proof*⟩

28.3 Strenghten Unification With Unification Hints

lemma

assumes [*uhint*]: $n = m$
shows $n - m = (0 :: nat)$
⟨*proof*⟩

lemma
assumes $x = y$
shows $y = x$
⟨*proof*⟩

Unfolding definitions. **definition** $mysuc\ n = Suc\ n$

lemma
assumes $\bigwedge m.\ Suc\ n > mysuc\ m$
shows $mysuc\ n > Suc\ 3$
⟨*proof*⟩

Discharging meta impliciations with object-level implications **lemma** [*uhint*]:
$Trueprop\ A \equiv A' \Longrightarrow Trueprop\ B \equiv B' \Longrightarrow Trueprop\ (A \longrightarrow B) \equiv (PROP\ A' \Longrightarrow PROP\ B')$
⟨*proof*⟩

lemma
assumes $A \longrightarrow (B \longrightarrow C) \longrightarrow D$
shows $A \Longrightarrow (B \Longrightarrow C) \Longrightarrow D$
⟨*proof*⟩

lemma
assumes $A \longrightarrow ((B \longrightarrow C) \longrightarrow D) \longrightarrow E$
shows $A \Longrightarrow ((B \Longrightarrow C) \Longrightarrow D) \Longrightarrow E$
⟨*proof*⟩

28.4 Better Control Over Meta Variable Instantiations

Consider the following type-inference problem.

schematic-goal
assumes *app-typeI*: $\bigwedge f\ x.\ (\bigwedge x.\ ArgT\ x \Longrightarrow DomT\ x\ (f\ x)) \Longrightarrow ArgT\ x \Longrightarrow DomT\ x\ (f\ x)$
and *f-type*: $\bigwedge x.\ ArgT\ x \Longrightarrow DomT\ x\ (f\ x)$
and *x-type*: $ArgT\ x$
shows $?T\ (f\ x)$
⟨*proof*⟩

end

end

29 Examples: Reification Via Unification Hints

theory *Unification-Hints-Reification-Examples*
  imports
    HOL.Rat
    ML-Unification-HOL-Setup
    Unify-Fact-Tactic
    Unify-Resolve-Tactics
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 *ML-Unification.ML-Unification-Hints*.

29.1 Setup

One-time setup to obtain a unifier with unification hints for the purpose of reification.

⟨*ML*⟩

Premises of hints should again be unified by the reification unifier.

declare [[*reify-uhint* **where** *prems-unifier* = *reify-unify*]]

29.2 Formulas with Quantifiers and Environment

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

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 (Less ?i ?j) ?vs = (?vs ! ?i < ?vs ! ?j)
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 reification.

experiment
begin

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 conclusion. 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 |> Type-Unification.e-match Unification-Util.match-types*›
and *retrieval =* ‹*Term-Index-Unification-Hints-Args.mk-retrieval-sym*

(*Term-Index-Unification-Hints-Args.retrieve-left Reification-Unification-Hints.TI.unifiables*)
Reification-Unification-Hints.TI.norm-term›]]

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*]:
$n \equiv Suc\ n' \Longrightarrow vs \equiv v\ \#\ vs' \Longrightarrow vs'\ !\ n' \equiv x \Longrightarrow vs\ !\ n \equiv x$
⟨*proof*⟩

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

lemma [*reify-uhint*]:
$[\![e \equiv ExQ\ f;\ \bigwedge v.\ interp\ f\ (v\ \#\ vs) \equiv P\ v]\!] \Longrightarrow interp\ e\ vs \equiv \exists v.\ P\ v$
$[\![e \equiv Less\ i\ j;\ x \equiv vs\ !\ i;\ y \equiv vs\ !\ j]\!] \Longrightarrow interp\ e\ vs \equiv x < y$
$[\![e \equiv And\ f1\ f2;\ interp\ f1\ vs \equiv r1;\ interp\ f2\ vs \equiv r2]\!] \Longrightarrow interp\ e\ vs \equiv r1 \wedge r2$
$[\![e \equiv Or\ f1\ f2;\ interp\ f1\ vs \equiv r1;\ interp\ f2\ vs \equiv r2]\!] \Longrightarrow interp\ e\ vs \equiv r1 \vee r2$
$e \equiv Neg\ f \Longrightarrow interp\ f\ vs \equiv r \Longrightarrow interp\ e\ vs \equiv \neg r$
$e \equiv TrueF \Longrightarrow interp\ e\ vs \equiv True$
$e \equiv FalseF \Longrightarrow interp\ e\ vs \equiv False$
⟨*proof*⟩

schematic-goal
$interp\ ?f\ (?vs :: ('a :: ord)\ list) = (\exists (x :: 'a).\ x < y \wedge \neg(\exists (z :: 'a).\ v < z \vee \neg False))$
⟨*proof*⟩

end

The next examples are modification from [1].

29.3 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-uhint* **where** *prio* = *Prio.LOW*]:
$e \equiv Var\ i \Longrightarrow eval\text{-}add\text{-}expr\ e \equiv i$ ⟨*proof*⟩

lemma *eval-add-expr-add* [*reify-uhint*]:

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*⟩

29.4 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-uhint* **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-uhint* **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-uhint*]: *e* ≡ *Var 0* ⟹ Γ ≡ *n* # Θ ⟹ *eval-mul-expr* (*e*, Γ) ≡ *n*
⟨*proof*⟩

lemma [*reify-uhint*]:
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 ≡ *Unit* ⟹ *eval-mul-expr* (*e*, Γ) ≡ *1*
⟨*proof*⟩

⟨*ML*⟩

schematic-goal *eval-mul-expr ?e* = (*1* ∗ *inverse 3* ∗ *5* :: *rat*)
⟨*proof*⟩

end

References

[1] A. Asperti, W. Ricciotti, C. Sacerdoti Coen, and E. Tassi. Hints in unification. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Theorem Proving in Higher Order Logics*, pages 84–98, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[2] K. Kappelmann, L. Bulwahn, and S. Willenbrink. Speccheck - specification-based testing for isabelle/ml. *Archive of Formal Proofs*, July 2021. https://isa-afp.org/entries/SpecCheck.html, Formal proof development.