

Unification Utilities for Isabelle/ML

Kevin Kappelmann

May 13, 2024

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	3
2	ML Attributes	3
3	ML Logger	3
3.1	Setup Result Commands	4
3.2	Examples	4

4 ML Attribute Utils	5
5 ML Conversion Utils	6
6 ML Parsing Utils	6
7 ML Functor Instances	6
8 General ML Utils	7
9 ML Generic Data Utils	7
10 ML Method Utils	7
11 Priorities	7
12 ML-Normalisations	8
13 ML-Binders	8
14 ML Term Utils	8
15 ML Theorem Utils	8
16 ML Unification Basics	9
17 ML Tactic Utils	9
18 ML Utils	9
19 ML Unifiers	10
20 Simps To	10
21 Unification Parsers	12
21.1 Assumption Tactic	13
21.2 Resolution Tactics	14
21.3 Resolution Tactics	14
21.4 Fact Tactic	16
21.5 Fact Tactic	16
22 Unification Tactics	17
23 Unification Attributes	17
24 Term Indexing	18
25 Unification Hints	18

26 Unification Hints	19
27 Setup for HOL	20
28 E-Unification Examples	21
28.1 Using The Simplifier For Unification.	22
28.2 Providing Canonical Solutions With Unification Hints	22
28.3 Strengthen Unification With Unification Hints	23
28.4 Better Control Over Meta Variable Instantiations	24
29 Examples: Reification Via Unification Hints	24
29.1 Setup	24
29.2 Formulas with Quantifiers and Environment	25
29.3 Simple Arithmetic	27
29.4 Arithmetic with Environment	27

1 ML Code Utils

```
theory ML-Code-Utils
  imports Pure
begin
```

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

$\langle ML \rangle$

end

2 ML Attributes

```
theory ML-Attributes
  imports ML-Code-Utils
begin
```

Summary ML code as attributes.

$\langle ML \rangle$

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:  $(A \equiv B) \equiv (B \equiv A)$ 
   $\langle proof \rangle$ 
 $\langle ML \rangle$ 

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.

$\langle ML \rangle$

Example $\langle ML \rangle$

end

7 ML Functor Instances

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

Summary Utilities for ML functors that create context data.

$\langle ML \rangle$

Example $\langle ML \rangle$

end

8 General ML Utils

```
theory ML-General-Utils  
  imports Pure  
begin
```

Summary General ML utilities.

$\langle ML \rangle$

end

9 ML Generic Data Utils

```
theory ML-Generic-Data-Utils  
  imports Pure  
begin
```

Summary Utilities for `Generic_Data`.

$\langle ML \rangle$

end

10 ML Method Utils

```
theory ML-Method-Utils  
  imports Pure  
begin
```

Summary Utilities for methods.

$\langle ML \rangle$

end

11 Priorities

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

Summary Priorities for ML tactics.

$\langle ML \rangle$

end

12 ML-Normalisations

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

Summary Normalisation functions for terms, types, and theorems.

$\langle ML \rangle$

end

13 ML-Binders

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

Summary Binders for ML.

$\langle ML \rangle$

end

14 ML Term Utils

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

Summary Utilities for terms.

$\langle ML \rangle$

end

15 ML Theorem Utils

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

Summary Utilities for theorems.

$\langle ML \rangle$

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.

$\langle ML \rangle$

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.

$\langle ML \rangle$

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
begin

```

Summary Unification modulo equations and combinators for unifiers.

Combinators $\langle ML \rangle$

Type Unifiers $\langle ML \rangle$

Standard Unifiers $\langle ML \rangle$

```

end

```

20 Simps To

```

theory Simps-To
  imports
    ML-Tactic-Utils
    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* = $\langle \text{Logger.new-logger } \text{Logger.root } \text{Simps-To-Base} \rangle$

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

lemma *SIMPS-TO-eq*: *SIMPS-TO* $s \ t \equiv (s \equiv t)$
 $\langle \text{proof} \rangle$

Prevent simplification of second/right argument

lemma *SIMPS-TO-cong* [*cong*]: $s \equiv s' \implies \text{SIMPS-TO } s \ t \equiv \text{SIMPS-TO } s' \ t$
 $\langle \text{proof} \rangle$

lemma *SIMPS-TOI*: *PROP SIMPS-TO* $s \ s \ \langle \text{proof} \rangle$

lemma *SIMPS-TOD*: *PROP SIMPS-TO* $s \ t \implies s \equiv t \ \langle \text{proof} \rangle$

$\langle ML \rangle$

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-UNIF1*: *PROP SIMPS-TO* $s\ s' \implies s' \equiv t \implies$ *PROP SIMPS-TO-UNIF* $s\ t$ `<proof>`

lemma *SIMPS-TO-UNIFD*: *PROP SIMPS-TO-UNIF* $s\ t \implies s \equiv t$ `<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`

`end`

theory *ML-Unifiers*

imports

ML-Functor-Instances

ML-Priorities

ML-Unifiers-Base

Simps-To

`begin`

Summary More unifiers.

Derived Unifiers `<ML>`

Unification via Tactics $\langle ML \rangle$

Unification via Simplification **lemma** *eq-if-SIMPS-TO-UNIF-if-SIMPS-TO*:
 assumes *PROP SIMPS-TO* *t t'*
 and *PROP SIMPS-TO-UNIF* *s t'*
 shows *s* \equiv *t*
 $\langle proof \rangle$

$\langle ML \rangle$

Combining Unifiers $\langle ML \rangle$

Mixture of Unifiers $\langle ML \rangle$

```
declare [[ucombine add =  $\langle$ 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  $\langle$ var-hop-unif $\rangle$  Prio.HIGH) $\rangle$ ]]

declare [[ucombine add =  $\langle$ Standard-Unification-Combine.eunif-data
  (Simplifier-Unification.simp-unify-progress Envir.aecon Simplifier-Unification.simp-unify
  (Unification-Util.inst-norm-term'
    Standard-Mixed-Unification.norms-first-higherp-decomp-comb-higher-unify)
    Standard-Mixed-Unification.norms-first-higherp-decomp-comb-higher-unify
    Standard-Mixed-Unification.first-higherp-decomp-comb-higher-unify
  |> Type-Unification.e-unify Unification-Util.unify-types
  |> K)
  (Standard-Unification-Combine.default-metadata binding  $\langle$ simp-unif $\rangle$ )
   $\rangle$ ]]

end
```

21 Unification Parsers

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

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

$\langle ML \rangle$

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.

$\langle ML \rangle$

end

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

Summary Setup of assumption tactic and examples.

$\langle ML \rangle$

Examples **experiment**
begin

```
lemma PROP P  $\implies$  PROP P
   $\langle proof \rangle$ 
```

```
lemma
  assumes h:  $\bigwedge P. PROP P$ 
  shows PROP P x
   $\langle proof \rangle$ 
```

```
schematic-goal  $\bigwedge x. PROP P (c :: 'a) \implies PROP ?Y (x :: 'a)$ 
   $\langle proof \rangle$ 
```

```
schematic-goal a: PROP ?P (y :: 'a)  $\implies$  PROP ?P (?x :: 'a)
   $\langle proof \rangle$ 
```

```
schematic-goal
  PROP ?P (x :: 'a)  $\implies$  PROP P (?x :: 'a)
   $\langle proof \rangle$ 
```

```
schematic-goal
   $\bigwedge x. PROP D \implies (\bigwedge y. PROP P y x) \implies PROP C \implies PROP P x$ 
```

$\langle proof \rangle$

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 $\bigwedge x. PROP D \implies (\bigwedge y. PROP A y \implies PROP B x) \implies PROP C \implies PROP A x \implies PROP B x$
 $\langle proof \rangle$

lemma $\bigwedge x. PROP D \implies (\bigwedge y. PROP A y \implies PROP B x) \implies PROP A x \implies PROP C \implies PROP B x$
 $\langle proof \rangle$

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.

$\langle ML \rangle$

end

21.3 Resolution Tactics

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

Summary Setup of resolution tactics and examples.

$\langle ML \rangle$

Examples **experiment**
begin

lemma
 assumes $h: \bigwedge x. PROP D x \implies PROP C x$
 shows $\bigwedge x. PROP A x \implies PROP B x \implies PROP C x$

$\langle proof \rangle$

lemma

assumes $h: PROP\ C\ x$

shows $PROP\ C\ x$

$\langle proof \rangle$

lemma

assumes $h: \bigwedge x. PROP\ A\ x \implies PROP\ D\ x$

shows $\bigwedge x. PROP\ A\ x \implies PROP\ B\ x \implies PROP\ C\ x$

— use (r,e,d,f) to specify the resolution mode (resolution, elim, dest, forward)

$\langle proof \rangle$

lemma

assumes $h1: \bigwedge x. PROP\ A\ x \implies PROP\ D\ x$

and $h2: \bigwedge x. PROP\ D\ x \implies PROP\ E\ x$

shows $\bigwedge x. PROP\ A\ x \implies PROP\ B\ x \implies PROP\ C\ x$

— use (rr,re,rd,rf) to use repetition; in particular: (*urule* (*rr*)) \simeq *intro*

$\langle proof \rangle$

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 $h1: \bigwedge x. (PROP\ F\ x \implies PROP\ E\ x) \implies PROP\ C\ x$

and $h2: \bigwedge x. PROP\ F\ x \implies PROP\ E\ x$

shows $\bigwedge x. PROP\ A\ x \implies PROP\ B\ x \implies PROP\ C\ x$

— Compare all of the following calls:

$\langle proof \rangle$

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

lemma

assumes $h1: \bigwedge x\ y. PROP\ C\ y \implies PROP\ D\ x \implies PROP\ C\ x$

and $h2: \bigwedge x\ y. PROP\ C\ x \implies PROP\ D\ x$

and $h3: \bigwedge x\ y. PROP\ C\ x$

shows $\bigwedge x. PROP\ A\ x \implies PROP\ B\ x \implies PROP\ C\ x$

$\langle proof \rangle$

lemma

assumes $h1: \bigwedge x\ y. PROP\ C\ y \implies PROP\ A\ x \implies PROP\ C\ x$

and $h2: \bigwedge x\ y. PROP\ C\ x \implies PROP\ B\ x \implies PROP\ D\ x$

and $h3: \bigwedge x\ y. PROP\ C\ x$

shows $\bigwedge x. PROP\ A\ x \implies PROP\ B\ x \implies PROP\ C\ x$

$\langle proof \rangle$

end

end

21.4 Fact Tactic

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

Summary Fact tactic with adjustable unifier.

$\langle ML \rangle$

end

21.5 Fact Tactic

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

Summary Setup of fact tactic and examples.

$\langle ML \rangle$

Examples experiment

begin

lemma

```
  assumes  $h: \bigwedge x y. PROP P x y$ 
  shows  $PROP P x y$ 
   $\langle proof \rangle$ 
```

lemma

```
  assumes  $\bigwedge^P y. PROP P y x$ 
  shows  $PROP P x$ 
   $\langle proof \rangle$ 
```

lemma

```
  assumes  $\bigwedge x y. PROP A x \implies PROP B x \implies PROP P x$ 
  shows  $\bigwedge x y. PROP A x \implies PROP B x \implies PROP P x$ 
   $\langle proof \rangle$ 
```

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.

$\langle ML \rangle$

end

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

Summary Setup of OF attribute with adjustable unifier.

$\langle ML \rangle$

Examples experiment

```
begin
lemma
  assumes  $h1: (PROP\ A \implies PROP\ D) \implies PROP\ E \implies PROP\ C$ 
  assumes  $h2: PROP\ B \implies PROP\ D$ 
  and  $h3: PROP\ F \implies PROP\ E$ 
  shows  $(PROP\ A \implies PROP\ B) \implies PROP\ F \implies PROP\ C$ 
   $\langle proof \rangle$ 
```

```
lemma
  assumes  $h1: (PROP\ A \implies PROP\ A)$ 
  assumes  $h2: (PROP\ A \implies PROP\ A) \implies PROP\ B$ 
  shows  $PROP\ B$ 
   $\langle proof \rangle$ 
```

```

lemma
  assumes  $h1: \bigwedge x y z. PROP\ P\ x\ y \implies PROP\ P\ y\ y \implies (PROP\ A \implies PROP\ A)$ 
  shows  $(PROP\ A \implies PROP\ B) \implies PROP\ C$ 
  and  $h2: \bigwedge x y. PROP\ P\ x\ y$ 
  and  $h3: PROP\ A \implies PROP\ A$ 
  and  $h4: PROP\ D \implies PROP\ B$ 
  shows  $(PROP\ A \implies PROP\ D) \implies PROP\ C$ 
   $\langle proof \rangle$ 

```

```

lemma
  assumes  $h1: \bigwedge P x. PROP\ P\ x \implies PROP\ E\ P\ x$ 
  and  $h2: PROP\ P\ x$ 
  shows  $PROP\ E\ P\ x$ 
   $\langle proof \rangle$ 

```

We can also specify the unifier to be used:

```

lemma
  assumes  $h1: \bigwedge P. PROP\ P \implies PROP\ E$ 
  and  $h2: \bigwedge P. PROP\ P$ 
  shows  $PROP\ E$ 
   $\langle proof \rangle$ 

```

end

end

24 Term Indexing

```

theory ML-Term-Index
  imports
    ML-Normalisations
  begin

```

Summary Termin indexes signatures and implementations.

$\langle ML \rangle$

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-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: $\bigwedge y1 \dots yn. (\bigwedge x1 \dots xn1. lhs1 \equiv rhs1) \implies \dots \implies (\bigwedge x1 \dots xnk. lhsk \equiv rhsk) \implies lhs \equiv rhs$

$\langle ML \rangle$

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.

$\langle ML \rangle$

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 only consider those hints whose lhs 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.

```
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.LOW
+ 1))
  (*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.LOW
+ 1))⟩]]
```

Examples see `../Examples`.

end

27 Setup for HOL

```
theory ML-Unification-HOL-Setup
imports
  HOL.HOL
```

ML-Unification-Hints
begin

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

declare [[*ucombine add* = *<Standard-Unification-Combine.eunif-data*
(*Simplifier-Unification.simp-unify-progress* *Envir.aeconv*
(*Simplifier-Unification.SIMPS-TO-UNIF-unify* @{thm eq-TrueI})
(*Unification-Util.inst-norm-term'*
Standard-Mixed-Unification.norms-first-higherp-decomp-comb-higher-unify)
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

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: $0 \in \text{even}$ |
step: $n \in \text{even} \implies \text{Suc } (\text{Suc } n) \in \text{even}$

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

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

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

lemma $6 \in \text{even}$
 ⟨*proof*⟩

lemma $(220 + (80 - 2 * 2)) \in \text{even}$
 ⟨*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 $(x + (y :: \text{nat}))^2 \leq x^2 + 2*x*y + y^2 + 4 * y + x - y$
 ⟨*proof*⟩

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

28.2 Providing Canonical Solutions With Unification Hints

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

schematic-goal $n - ?m = (0 :: \text{nat})$
 ⟨*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 $0::'a$ cannot be unified with *length*

[]]. However, the simplifier can rewrite $\text{length } []$ to $0 :: 'a$ and the hint can then be applied.

declare [[*ML-map-context* $\langle \text{Logger.set-log-levels Logger.root Logger.TRACE} \rangle$]]

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

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 $\text{length } [] = 0$ 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 $\text{length } [] = 0$ as an additional hint.

lemma *length-nil-eq* [*uhint*]: $\text{length } [] = 0 \langle \text{proof} \rangle$

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

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

Solution 2: make the recursion explicit in the hint.

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

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

28.3 Strengthen Unification With Unification Hints

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

lemma
assumes $x = y$
shows $y = x$
 $\langle \text{proof} \rangle$

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

lemma
assumes $\bigwedge m. \text{Suc } n > \text{mysuc } m$
shows $\text{mysuc } n > \text{Suc } 3$
 $\langle \text{proof} \rangle$

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

```

  Trueprop A  $\equiv$  A'  $\implies$  Trueprop B  $\equiv$  B'  $\implies$  Trueprop (A  $\longrightarrow$  B)  $\equiv$  (PROP A'
 $\implies$  PROP B')
  <proof>

```

lemma

```

  assumes A  $\longrightarrow$  (B  $\longrightarrow$  C)  $\longrightarrow$  D
  shows A  $\implies$  (B  $\implies$  C)  $\implies$  D
  <proof>

```

28.4 Better Control Over Meta Variable Instantiations

Consider the following type-inference problem.

schematic-goal

```

  assumes app-typeI:  $\bigwedge f x. (\bigwedge x. ArgT x \implies DomT x (f x)) \implies ArgT x \implies$ 
    DomT x (f x)
  and f-type:  $\bigwedge x. ArgT x \implies DomT x (f x)$ 
  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. We could also simply use the standard unification hints *uhint* and *rec-uhint*, but having separate instances is a cleaner approach.

$\langle ML \rangle$

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

declare $[[reify-uhint \textbf{ where } prems-unifier = reify-unify]]$

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* \Rightarrow (*a::ord*) *list* \Rightarrow *bool*

where

interp TrueF vs \longleftrightarrow *True*
 | *interp FalseF vs* \longleftrightarrow *False*
 | *interp (Less i j) vs* \longleftrightarrow *vs ! i < vs ! j*
 | *interp (And f1 f2) vs* \longleftrightarrow *interp f1 vs* \wedge *interp f2 vs*
 | *interp (Or f1 f2) vs* \longleftrightarrow *interp f1 vs* \vee *interp f2 vs*
 | *interp (Neg f) vs* \longleftrightarrow \neg *interp f vs*
 | *interp (ExQ f) vs* \longleftrightarrow $(\exists v. \text{interp } f \text{ (v \# vs)})$

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 \wedge interp ?f2.0 ?vs)
interp (Or ?f1.0 ?f2.0) ?vs = (interp ?f1.0 ?vs \vee interp ?f2.0 ?vs)
interp (Neg ?f) ?vs = (\neg interp ?f ?vs)
interp (ExQ ?f) ?vs = ($\exists v. \text{interp } ?f \text{ (v \# ?vs)})$ directly as unification

hints for reification.

experiment

begin

Hints for list lookup.

declare *List.nth-Cons-Suc* $[reify-uhint \textbf{ where } prio = Prio.LOW]$

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*) = ($\exists (x :: 'a). x < y \wedge \neg(\exists (z :: 'a). v < z \vee \neg \text{False})$)
 <*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*]]

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*]:

$\llbracket e \equiv \text{ExQ } f; \bigwedge v. \text{interp } f (v \# vs) \equiv P v \rrbracket \implies \text{interp } e \text{ vs} \equiv \exists v. P v$

$\llbracket e \equiv \text{Less } i j; x \equiv vs ! i; y \equiv vs ! j \rrbracket \implies \text{interp } e \text{ vs} \equiv x < y$

$\llbracket e \equiv \text{And } f1 f2; \text{interp } f1 \text{ vs} \equiv r1; \text{interp } f2 \text{ vs} \equiv r2 \rrbracket \implies \text{interp } e \text{ vs} \equiv r1 \wedge r2$

```


$$\llbracket e \equiv \text{Or } f1 \ f2; \text{interp } f1 \ vs \equiv r1; \text{interp } f2 \ vs \equiv r2 \rrbracket \implies \text{interp } e \ vs \equiv r1 \vee r2$$


$$e \equiv \text{Neg } f \implies \text{interp } f \ vs \equiv r \implies \text{interp } e \ vs \equiv \neg r$$


$$e \equiv \text{TrueF} \implies \text{interp } e \ vs \equiv \text{True}$$


$$e \equiv \text{FalseF} \implies \text{interp } e \ vs \equiv \text{False}$$


$$\langle \text{proof} \rangle$$


```

schematic-goal

```


$$\text{interp } ?f \ ( ?vs :: ('a :: \text{ord}) \text{ list} ) = (\exists (x :: 'a). x < y \wedge \neg(\exists (z :: 'a). v < z \vee \neg \text{False}))$$


$$\langle \text{proof} \rangle$$


```

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  $\Rightarrow$  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]:

```

```

   $e \equiv \text{Var } i \implies \text{eval-add-expr } e \equiv i \ \langle \text{proof} \rangle$ 

```

```

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

```

```

   $e \equiv \text{Add } e1 \ e2 \implies \text{eval-add-expr } e1 \equiv m \implies \text{eval-add-expr } e2 \equiv n \implies$ 
 $\text{eval-add-expr } e \equiv m + n$ 
 $\langle \text{proof} \rangle$ 

```

$\langle ML \rangle$

```

schematic-goal eval-add-expr ?e =  $(1 + (2 + 7) :: \text{int})$ 

```

$\langle \text{proof} \rangle$

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  $\times$  rat list  $\Rightarrow$  rat where

```

```

  eval-mul-expr (Unit,  $\Gamma$ ) = 1
| eval-mul-expr (Var i,  $\Gamma$ ) =  $\Gamma ! i$ 
| eval-mul-expr (Mul e1 e2,  $\Gamma$ ) = eval-mul-expr (e1,  $\Gamma$ ) * eval-mul-expr (e2,  $\Gamma$ )
| eval-mul-expr (Inv e,  $\Gamma$ ) = inverse (eval-mul-expr (e,  $\Gamma$ ))

```

Split e into an expression and an environment.

lemma *[reify-uhint where prio = Prio.VERY-LOW]*:
 $e \equiv (e1, \Gamma) \implies eval_mul_expr (e1, \Gamma) \equiv n \implies eval_mul_expr e \equiv n$
 $\langle proof \rangle$

Hints for environment lookup.

lemma *[reify-uhint where prio = Prio.LOW]*:
 $e \equiv Var (Suc p) \implies \Gamma \equiv s \# \Delta \implies n \equiv eval_mul_expr (Var p, \Delta) \implies$
 $eval_mul_expr (e, \Gamma) \equiv n$
 $\langle proof \rangle$

lemma *[reify-uhint]*: $e \equiv Var 0 \implies \Gamma \equiv n \# \Theta \implies eval_mul_expr (e, \Gamma) \equiv n$
 $\langle proof \rangle$

lemma *[reify-uhint]*:
 $e1 \equiv Inv e2 \implies n \equiv eval_mul_expr (e2, \Gamma) \implies eval_mul_expr (e1, \Gamma) \equiv inverse$
 n
 $e \equiv Mul e1 e2 \implies m \equiv eval_mul_expr (e1, \Gamma) \implies n \equiv eval_mul_expr (e2, \Gamma)$
 \implies
 $eval_mul_expr (e, \Gamma) \equiv m * n$
 $e \equiv Unit \implies eval_mul_expr (e, \Gamma) \equiv 1$
 $\langle proof \rangle$

$\langle ML \rangle$

schematic-goal $eval_mul_expr ?e = (1 * inverse 3 * 5 :: rat)$
 $\langle proof \rangle$

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.