

# A Reusable Isabelle/HOL Framework for Propositional Labelled Natural Deduction

Arthur Freitas Ramos      David Barros Hulak  
Ruy J. G. B. de Queiroz

June 5, 2026

## Abstract

A reusable Isabelle/HOL framework presenting labelled natural deduction as a conservative refinement of ordinary natural deduction, with generic structural metatheory and several concrete label interpretations. We define ordinary intuitionistic propositional natural deduction, a parametric labelled refinement, prove erasure soundness (labelled derivations are ordinary derivations after erasing labels) and a controlled lifting theorem (ordinary derivations admit some labelling). Standard structural metatheory — weakening, exchange, substitution, cut admissibility — is established in both the ordinary and labelled systems.

The framework is instantiated three times. Unit labels recover ordinary ND as a special case via conservativity. Provenance labels carry sets of assumption indices and give both a sound over-approximation of assumption dependencies and a completeness result equating ordinary derivability with the existence of a provenance-labelled derivation whose filtered sub-context still derives the conclusion. A possible-world modal instance derives the K axiom schema and necessitation and is connected to Kripke validity; the modal fragment is presented as an example instantiation, not as a full labelled modal proof theory.

The development is fully constructive, contains no unfinished proof commands or oracle invocations, and is intended as a foundation other entries can cite.

## Contents

<b>1</b>	<b>What this entry proves</b>	<b>3</b>
<b>2</b>	<b>Overview</b>	<b>3</b>
<b>3</b>	<b>Label algebra and adequacy</b>	<b>4</b>
<b>4</b>	<b>Modal K axiom and necessitation</b>	<b>4</b>



## 1 What this entry proves

The main named results, with the theory in which each is established, are:

- `labelled_sound_under_erasure`  
(`Erasure`) — every labelled derivation erases to an ordinary derivation.
- `lifting_general, ordinary_derivations_admit_labelling`  
(`Label_Algebra, Lifting`) — ordinary derivations admit a labelling under the assumed lift-coherence law.
- `framework_adequacy, framework_adequacy_iff`  
(`Label_Algebra`) — the locale-level equivalence between labelled derivability up to a label and ordinary derivability after erasure.
- `unit_labels_conservative`  
(`Unit_Labels`) — the unit-labels instance recovers ordinary ND.
- `provenance_overapproximates_dependencies`  
(`Provenance_Labels`) — the provenance instance gives a sound over-approximation of assumption dependencies.
- `ordinary_admits_safe_provenance, ordinary_admits_tight_provenance, provenance_completeness`  
(`Provenance_Labels`) — the provenance instance is dependency-tracking complete: ordinary derivability is equivalent to the existence of a provenance-labelled derivation whose selected filtered context still derives the conclusion.
- `modal_K_schema, modal_necessitation, modal_K_kripke_valid`  
(`Modal_K_Schema`) — the worked modal instance, where the K axiom schema is derived uniformly and connected to Kripke validity.

## 2 Overview

Labelled deductive systems were introduced by Gabbay as a general way to attach structured information to formulae and rules [2]. The modal example in this entry follows the standard possible-world reading of modal labels, closely related to labelled proof-theoretic presentations such as Simpson’s work on intuitionistic modal logic [3]. Labelled approaches to non-classical proof systems have been developed extensively by Viganò [4] and by Basin, Matthews, and Viganò [1]; our framework targets the simpler proof-theoretic core of propositional intuitionistic ND but recovers similar structural properties.

### 3 Label algebra and adequacy

The entry factors the parametric part of labelled derivability through a label-algebra locale. The locale records the proof obligations needed by any future label instance: that label constructors are uniform under erasure to the singleton type and that ordinary derivations can be lifted coherently over the labelled context. Its central theorem, `framework_adequacy`, states that labelled derivability up to a label is equivalent to ordinary derivability after erasing the context, with the chosen lift supplying the canonical labelled witness. The result is intentionally architectural rather than surprising: it packages the soundness/lifting obligations a new label instance must discharge, so that once those obligations are met the equivalence comes for free.

### 4 Modal K axiom and necessitation

The worked modal instance is presented as an example, not as a self-contained labelled modal proof theory, but it is strong enough to derive the K axiom schema uniformly in the formulae  $A$ ,  $B$ , the accessibility relation  $R$ , and the world  $w$ . The theory `Modal_K_Schema.thy` records this as `modal_k_schema`, proves `modal_necessitation` under the standard side-condition that necessitation applies only to formulae derivable from the empty context at every world and relation, and connects the labelled derivation to Kripke semantics through `modal_k_kripke_valid`.

### 5 Provenance completeness

The provenance instance is the entry's main dependency-tracking result. It is adequate in a strong sense: every ordinary derivation lifts to a labelled derivation whose provenance is contained in the indices of the ordinary context and whose selected sub-context is still sufficient to re-derive the conclusion. The relevant named theorems are:

- `ordinary_admits_safe_provenance` — records the index-bound and well-formedness properties of the lifted provenance derivation.
- `ordinary_admits_tight_provenance` — strengthens this to a sufficient filtered sub-context.
- `provenance_completeness` — states the resulting equivalence between ordinary derivability and provenance-labelled derivability with a sufficient filtered context.

```

theory Labelled-Formulas
  imports Main
begin

```

This theory fixes the propositional formula language and the elementary operations that erase labels from labelled formulae, databases, and list contexts. Labels are deliberately represented by a product component, so the logical shape of a labelled formula is recovered by taking its second projection.

```

datatype 'a fm =
  Atom 'a
  | Bot
  | Imp 'a fm 'a fm
  | Con 'a fm 'a fm
  | Dis 'a fm 'a fm

```

```

type-synonym ('l, 'a) lfm = 'l × 'a fm

```

```

definition erase-lfm :: ('l, 'a) lfm ⇒ 'a fm where
  erase-lfm x = snd x

```

```

definition erase-db :: ('l, 'a) lfm set ⇒ 'a fm set where
  erase-db Γ = snd ` Γ

```

```

definition erase-ctx :: ('l, 'a) lfm list ⇒ 'a fm list where
  erase-ctx Γ = map snd Γ

```

```

lemma erase-lfm-simps [simp]:
  erase-lfm (l, A) = A
  by (simp add: erase-lfm-def)

```

```

lemma erase-ctx-Nil [simp]:
  erase-ctx [] = []
  by (simp add: erase-ctx-def)

```

```

lemma erase-ctx-Cons [simp]:
  erase-ctx (x # Γ) = erase-lfm x # erase-ctx Γ
  by (cases x) (simp add: erase-ctx-def)

```

```

lemma erase-ctx-append [simp]:
  erase-ctx (Γ @ Δ) = erase-ctx Γ @ erase-ctx Δ
  by (simp add: erase-ctx-def)

```

```

lemma erase-db-insert [simp]:
  erase-db (insert x Γ) = insert (erase-lfm x) (erase-db Γ)
  by (cases x) (auto simp: erase-db-def erase-lfm-def)

```

```

lemma erase-db-Un [simp]:
  erase-db (Γ ∪ Δ) = erase-db Γ ∪ erase-db Δ

```

by (auto simp: erase-db-def)

end

**theory** *Natural-Deduction*  
  **imports** *Labelled-Formulas*  
**begin**

This theory defines ordinary intuitionistic propositional natural deduction over list contexts. The structural metatheory is stated in terms of the set of assumptions represented by a list, which keeps exchange and weakening explicit while avoiding dependence on a particular context order.

**primrec** *fm-subst* :: ('a  $\Rightarrow$  'b fm)  $\Rightarrow$  'a fm  $\Rightarrow$  'b fm **where**

*fm-subst*  $\sigma$  (Atom  $p$ ) =  $\sigma$   $p$   
| *fm-subst*  $\sigma$  Bot = Bot  
| *fm-subst*  $\sigma$  (Imp  $A$   $B$ ) = Imp (*fm-subst*  $\sigma$   $A$ ) (*fm-subst*  $\sigma$   $B$ )  
| *fm-subst*  $\sigma$  (Con  $A$   $B$ ) = Con (*fm-subst*  $\sigma$   $A$ ) (*fm-subst*  $\sigma$   $B$ )  
| *fm-subst*  $\sigma$  (Dis  $A$   $B$ ) = Dis (*fm-subst*  $\sigma$   $A$ ) (*fm-subst*  $\sigma$   $B$ )

**inductive** *derives* :: 'a fm list  $\Rightarrow$  'a fm  $\Rightarrow$  bool (- $\vdash$ - [50, 50] 50)

**where**

*Assm*:  $A \in \text{set } \Gamma \Longrightarrow \Gamma \vdash A$   
| *BotE*:  $\Gamma \vdash \text{Bot} \Longrightarrow \Gamma \vdash A$   
| *ImpI*:  $A \# \Gamma \vdash B \Longrightarrow \Gamma \vdash \text{Imp } A B$   
| *ImpE*:  $\Gamma \vdash \text{Imp } A B \Longrightarrow \Gamma \vdash A \Longrightarrow \Gamma \vdash B$   
| *ConI*:  $\Gamma \vdash A \Longrightarrow \Gamma \vdash B \Longrightarrow \Gamma \vdash \text{Con } A B$   
| *ConE1*:  $\Gamma \vdash \text{Con } A B \Longrightarrow \Gamma \vdash A$   
| *ConE2*:  $\Gamma \vdash \text{Con } A B \Longrightarrow \Gamma \vdash B$   
| *DisI1*:  $\Gamma \vdash A \Longrightarrow \Gamma \vdash \text{Dis } A B$   
| *DisI2*:  $\Gamma \vdash B \Longrightarrow \Gamma \vdash \text{Dis } A B$   
| *DisE*:  $\Gamma \vdash \text{Dis } A B \Longrightarrow A \# \Gamma \vdash C \Longrightarrow B \# \Gamma \vdash C \Longrightarrow \Gamma \vdash C$

**lemma** *weakening*:

**assumes**  $\Gamma \vdash A$  **and**  $\text{set } \Gamma \subseteq \text{set } \Delta$

**shows**  $\Delta \vdash A$

**using** *assms*

**proof** (*induction arbitrary:  $\Delta$  rule: derives.induct*)

**case** (*Assm*  $A$   $\Gamma$ )

**then show** ?*case*

**by** (*meson derives.Assm subsetD*)

**next**

**case** (*BotE*  $\Gamma$   $A$ )

**then show** ?*case*

**by** (*meson derives.BotE*)

**next**

**case** (*ImpI*  $A$   $\Gamma$   $B$ )

**show** ?*case*

**proof** (*rule derives.ImpI*)

```

    show  $A \# \Delta \vdash B$ 
      by (rule ImpI.IH) (use ImpI.prems in auto)
  qed
next
  case (ImpE  $\Gamma A B$ )
  then show ?case
    by (meson derives.ImpE)
next
  case (ConI  $\Gamma A B$ )
  then show ?case
    by (meson derives.ConI)
next
  case (ConE1  $\Gamma A B$ )
  then show ?case
    by (meson derives.ConE1)
next
  case (ConE2  $\Gamma A B$ )
  then show ?case
    by (meson derives.ConE2)
next
  case (DisI1  $\Gamma A B$ )
  then show ?case
    by (meson derives.DisI1)
next
  case (DisI2  $\Gamma B A$ )
  then show ?case
    by (meson derives.DisI2)
next
  case (DisE  $\Gamma A B C$ )
  show ?case
  proof (rule derives.DisE)
    show  $\Delta \vdash Dis A B$ 
      using DisE.IH(1) DisE.prems by blast
    show  $A \# \Delta \vdash C$ 
      by (rule DisE.IH(2)) (use DisE.prems in auto)
    show  $B \# \Delta \vdash C$ 
      by (rule DisE.IH(3)) (use DisE.prems in auto)
  qed
qed

```

**lemma** *weakening-cons*:

```

  assumes  $\Gamma \vdash A$ 
  shows  $B \# \Gamma \vdash A$ 
  using assms by (rule weakening) auto

```

**lemma** *exchange*:

```

  assumes  $A \# B \# \Gamma \vdash C$ 
  shows  $B \# A \# \Gamma \vdash C$ 
  using assms by (rule weakening) auto

```

```

lemma cut-general:
  assumes  $\Gamma \vdash A$  and  $\Delta \vdash B$  and set  $\Delta \subseteq \text{insert } A \text{ (set } \Gamma)$ 
  shows  $\Gamma \vdash B$ 
  using assms(2,3,1)
proof (induction arbitrary:  $\Gamma$  A rule: derives.induct)
  case (Assm B  $\Delta$ )
  from Assm.hyps Assm.prem(1) have  $B = A \vee B \in \text{set } \Gamma$ 
    by auto
  then show ?case
  proof
    assume  $B = A$ 
    then show ?thesis
      using Assm.prem(2) by simp
  next
    assume  $B \in \text{set } \Gamma$ 
    then show ?thesis
      by (rule derives.Assm)
  qed
next
  case (BotE  $\Delta$  B)
  then show ?case
    by (meson derives.BotE)
next
  case (ImpI C  $\Delta$  D)
  show ?case
  proof (rule derives.ImpI)
    have deriv: C #  $\Gamma \vdash A$ 
      using ImpI.prem(2) by (rule weakening-cons)
    have sub: set (C #  $\Delta$ )  $\subseteq$  insert A (set (C #  $\Gamma$ ))
      using ImpI.prem(1) by auto
    show C #  $\Gamma \vdash D$ 
      by (rule ImpI.IH[OF sub deriv])
  qed
next
  case (ImpE  $\Delta$  C D)
  then show ?case
    by (meson derives.ImpE)
next
  case (ConI  $\Delta$  C D)
  then show ?case
    by (meson derives.ConI)
next
  case (ConE1  $\Delta$  C D)
  then show ?case
    by (meson derives.ConE1)
next
  case (ConE2  $\Delta$  C D)
  then show ?case

```

by (*meson derives.ConE2*)  
**next**  
 case (*DisI1*  $\Delta$  *C D*)  
 then show *?case*  
 by (*meson derives.DisI1*)  
**next**  
 case (*DisI2*  $\Delta$  *D C*)  
 then show *?case*  
 by (*meson derives.DisI2*)  
**next**  
 case (*DisE*  $\Delta$  *C D E*)  
 show *?case*  
**proof** (*rule derives.DisE*)  
 show  $\Gamma \vdash \text{Dis } C D$   
 using *DisE.IH(1) DisE.prem*s by *blast*  
 have *deriv-C*:  $C \# \Gamma \vdash A$   
 using *DisE.prem*s(2) by (*rule weakening-cons*)  
 have *sub-C*:  $\text{set } (C \# \Delta) \subseteq \text{insert } A (\text{set } (C \# \Gamma))$   
 using *DisE.prem*s(1) by *auto*  
 show  $C \# \Gamma \vdash E$   
 by (*rule DisE.IH(2)[OF sub-C deriv-C]*)  
 have *deriv-D*:  $D \# \Gamma \vdash A$   
 using *DisE.prem*s(2) by (*rule weakening-cons*)  
 have *sub-D*:  $\text{set } (D \# \Delta) \subseteq \text{insert } A (\text{set } (D \# \Gamma))$   
 using *DisE.prem*s(1) by *auto*  
 show  $D \# \Gamma \vdash E$   
 by (*rule DisE.IH(3)[OF sub-D deriv-D]*)  
**qed**  
**qed**

**lemma** *cut*:  
 assumes  $\Gamma \vdash A$  and  $A \# \Gamma \vdash B$   
 shows  $\Gamma \vdash B$   
 using *assms* by (*rule cut-general*) *auto*

**theorem** *deduction-theorem*:  
 $A \# \Gamma \vdash B \iff \Gamma \vdash \text{Imp } A B$   
**proof**  
 show  $A \# \Gamma \vdash B \implies \Gamma \vdash \text{Imp } A B$   
 by (*rule derives.ImpI*)  
**next**  
 assume  $\Gamma \vdash \text{Imp } A B$   
 then have  $A \# \Gamma \vdash \text{Imp } A B$   
 by (*rule weakening-cons*)  
 moreover have  $A \# \Gamma \vdash A$   
 by (*rule derives.Assm*) *simp*  
 ultimately show  $A \# \Gamma \vdash B$   
 by (*rule derives.ImpE*)  
**qed**

```

lemma substitution-atom:
  assumes  $\Gamma \vdash A$ 
  shows  $\text{map } (fm\text{-subst } \sigma) \Gamma \vdash fm\text{-subst } \sigma A$ 
  using assms
  by (induction rule: derives.induct) (auto intro: derives.intros)

```

**end**

```

theory Labelled-Natural-Deduction
  imports Natural-Deduction
begin

```

This theory introduces the parametric labelled natural-deduction kernel. A locale fixes inert label constructors for the logical rules, but imposes no equations on them; concrete label disciplines are supplied later by interpretation.

```

locale label-structure =
  fixes app :: 'l  $\Rightarrow$  'l  $\Rightarrow$  'l
    and lam :: 'l  $\Rightarrow$  'l  $\Rightarrow$  'l
    and pair :: 'l  $\Rightarrow$  'l  $\Rightarrow$  'l
    and fst-l :: 'l  $\Rightarrow$  'l
    and snd-l :: 'l  $\Rightarrow$  'l
    and abort :: 'l  $\Rightarrow$  'l
    and inl :: 'l  $\Rightarrow$  'l
    and inr :: 'l  $\Rightarrow$  'l
    and cases :: 'l  $\Rightarrow$  'l  $\Rightarrow$  'l  $\Rightarrow$  'l  $\Rightarrow$  'l
begin

```

```

inductive lderives :: ('l, 'a) lfm list  $\Rightarrow$  ('l, 'a) lfm  $\Rightarrow$  bool
  (-  $\vdash L$  - [50, 50] 50)

```

**where**

```

  LAssm:  $x \in \text{set } \Gamma \Longrightarrow \Gamma \vdash L x$ 
| LBotE:  $\Gamma \vdash L (l, \text{Bot}) \Longrightarrow \Gamma \vdash L (\text{abort } l, A)$ 
| LImpI:  $(l, A) \# \Gamma \vdash L (m, B) \Longrightarrow \Gamma \vdash L (\text{lam } l m, \text{Imp } A B)$ 
| LImpE:  $\Gamma \vdash L (l, \text{Imp } A B) \Longrightarrow \Gamma \vdash L (m, A) \Longrightarrow \Gamma \vdash L (\text{app } l m, B)$ 
| LConI:  $\Gamma \vdash L (l, A) \Longrightarrow \Gamma \vdash L (m, B) \Longrightarrow \Gamma \vdash L (\text{pair } l m, \text{Con } A B)$ 
| LConE1:  $\Gamma \vdash L (l, \text{Con } A B) \Longrightarrow \Gamma \vdash L (\text{fst-l } l, A)$ 
| LConE2:  $\Gamma \vdash L (l, \text{Con } A B) \Longrightarrow \Gamma \vdash L (\text{snd-l } l, B)$ 
| LDisI1:  $\Gamma \vdash L (l, A) \Longrightarrow \Gamma \vdash L (\text{inl } l, \text{Dis } A B)$ 
| LDisI2:  $\Gamma \vdash L (l, B) \Longrightarrow \Gamma \vdash L (\text{inr } l, \text{Dis } A B)$ 
| LDisE:
   $\Gamma \vdash L (l, \text{Dis } A B) \Longrightarrow$ 
   $(m, A) \# \Gamma \vdash L (n, C) \Longrightarrow$ 
   $(p, B) \# \Gamma \vdash L (q, C) \Longrightarrow$ 
   $\Gamma \vdash L (\text{cases } l m n p q, C)$ 

```

**lemma** *labelled-weakening*:

```

assumes  $\Gamma \vdash L x$  and set  $\Gamma \subseteq \text{set } \Delta$ 
shows  $\Delta \vdash L x$ 
using assms
proof (induction arbitrary:  $\Delta$  rule: lderives.induct)
  case (LAssm  $x \Gamma$ )
    then have  $x \in \text{set } \Delta$ 
      by auto
    then show ?case
      by (rule lderives.intros(1))
  next
    case (LBotE  $\Gamma l A$ )
    then show ?case
      by (meson lderives.intros)
  next
    case (LImpI  $l A \Gamma m B$ )
    show ?case
    proof (rule lderives.intros(3))
      show  $(l, A) \# \Delta \vdash L (m, B)$ 
        by (rule LImpI.IH) (use LImpI.prem in auto)
    qed
  next
    case (LImpE  $\Gamma l A B m$ )
    then show ?case
      by (meson lderives.intros)
  next
    case (LConI  $\Gamma l A m B$ )
    then show ?case
      by (meson lderives.intros)
  next
    case (LConE1  $\Gamma l A B$ )
    then show ?case
      by (meson lderives.intros)
  next
    case (LConE2  $\Gamma l A B$ )
    then show ?case
      by (meson lderives.intros)
  next
    case (LDisI1  $\Gamma l A B$ )
    then show ?case
      by (meson lderives.intros)
  next
    case (LDisI2  $\Gamma l B A$ )
    then show ?case
      by (meson lderives.intros)
  next
    case (LDisE  $\Gamma l A B m n C p q$ )
    show ?case
    proof (rule lderives.intros(10))
      show  $\Delta \vdash L (l, \text{Dis } A B)$ 

```

```

    using LDisE.IH(1) LDisE.premis by blast
  show (m, A) # Δ ⊢L (n, C)
    by (rule LDisE.IH(2)) (use LDisE.premis in auto)
  show (p, B) # Δ ⊢L (q, C)
    by (rule LDisE.IH(3)) (use LDisE.premis in auto)
qed
qed

lemma labelled-weakening-cons:
  assumes Γ ⊢L x
  shows y # Γ ⊢L x
  using assms by (rule labelled-weakening) auto

lemma labelled-exchange:
  assumes x # y # Γ ⊢L z
  shows y # x # Γ ⊢L z
  using assms by (rule labelled-weakening) auto

lemma labelled-cut-general:
  assumes Γ ⊢L x
    and Δ ⊢L y
    and set Δ ⊆ insert x (set Σ)
    and set Γ ⊆ set Σ
  shows Σ ⊢L y
  using assms(2,3,1,4)
proof (induction arbitrary: Γ x Σ rule: lderives.induct)
  case (LAssm y Δ)
  from LAssm.hyps LAssm.premis(1) have y = x ∨ y ∈ set Σ
  by auto
  then show ?case
  proof
    assume y = x
    then show ?thesis
    using LAssm.premis(2,3) labelled-weakening by blast
  next
    assume y ∈ set Σ
    then show ?thesis
    by (rule lderives.intros(1))
  qed
next
  case (LBotE Δ l A)
  then show ?case
  by (meson lderives.intros)
next
  case (LImpI l A Δ m B)
  show ?case
  proof (rule lderives.intros(3))
    have sub: set ((l, A) # Δ) ⊆ insert x (set ((l, A) # Σ))
    using LImpI.premis(1) by auto

```

```

    have sub-context: set  $\Gamma \subseteq \text{set } ((l, A) \# \Sigma)$ 
      using LImpI.prem3 by auto
    show  $(l, A) \# \Sigma \vdash L (m, B)$ 
      by (rule LImpI.IH[OF sub LImpI.prem2] sub-context])
  qed
next
case (LImpE  $\Delta l A B m$ )
then show ?case
  by (meson lderives.intros)
next
case (LConI  $\Delta l A m B$ )
then show ?case
  by (meson lderives.intros)
next
case (LConE1  $\Delta l A B$ )
then show ?case
  by (meson lderives.intros)
next
case (LConE2  $\Delta l A B$ )
then show ?case
  by (meson lderives.intros)
next
case (LDisI1  $\Delta l A B$ )
then show ?case
  by (meson lderives.intros)
next
case (LDisI2  $\Delta l B A$ )
then show ?case
  by (meson lderives.intros)
next
case (LDisE  $\Delta l A B m n C p q$ )
show ?case
proof (rule lderives.intros(10))
  show  $\Sigma \vdash L (l, \text{Dis } A B)$ 
    using LDisE.IH(1) LDisE.prem3 by blast
  have sub-m: set  $((m, A) \# \Delta) \subseteq \text{insert } x (\text{set } ((m, A) \# \Sigma))$ 
    using LDisE.prem1 by auto
  have sub-context-m: set  $\Gamma \subseteq \text{set } ((m, A) \# \Sigma)$ 
    using LDisE.prem3 by auto
  show  $(m, A) \# \Sigma \vdash L (n, C)$ 
    by (rule LDisE.IH(2)[OF sub-m LDisE.prem2] sub-context-m])
  have sub-p: set  $((p, B) \# \Delta) \subseteq \text{insert } x (\text{set } ((p, B) \# \Sigma))$ 
    using LDisE.prem1 by auto
  have sub-context-p: set  $\Gamma \subseteq \text{set } ((p, B) \# \Sigma)$ 
    using LDisE.prem3 by auto
  show  $(p, B) \# \Sigma \vdash L (q, C)$ 
    by (rule LDisE.IH(3)[OF sub-p LDisE.prem2] sub-context-p])
qed
qed

```

**lemma** *labelled-cut*:  
**assumes**  $\Gamma \vdash L x$  **and**  $x \# \Delta \vdash L y$   
**shows**  $\Gamma @ \Delta \vdash L y$   
**proof** (*rule labelled-cut-general*)  
**show**  $\Gamma \vdash L x$   
**by** (*rule assms(1)*)  
**show**  $x \# \Delta \vdash L y$   
**by** (*rule assms(2)*)  
**show**  $\text{set } (x \# \Delta) \subseteq \text{insert } x (\text{set } (\Gamma @ \Delta))$   
**by** *auto*  
**show**  $\text{set } \Gamma \subseteq \text{set } (\Gamma @ \Delta)$   
**by** *auto*  
**qed**

**lemma** *assumption-substitution-lift*:  
**assumes**  $x \in \text{set } \Gamma$   
**shows**  $(\text{fst } x, \text{fm-subst } \sigma (\text{snd } x)) \in \text{set } (\text{map } (\text{map-prod id } (\text{fm-subst } \sigma)) \Gamma)$   
**using** *assms* **by** (*induction*  $\Gamma$ ) *auto*

**lemma** *labelled-substitution-atom-aux*:  
**assumes**  $\Gamma \vdash L x$   
**shows**  $\text{map } (\text{map-prod id } (\text{fm-subst } \sigma)) \Gamma \vdash L (\text{fst } x, \text{fm-subst } \sigma (\text{snd } x))$   
**using** *assms*  
**proof** (*induction rule: lderives.induct*)  
**case** (*LAssm*  $x \Gamma$ )  
**then have**  $(\text{fst } x, \text{fm-subst } \sigma (\text{snd } x)) \in$   
 $\text{set } (\text{map } (\text{map-prod id } (\text{fm-subst } \sigma)) \Gamma)$   
**by** (*rule assumption-substitution-lift*)  
**then show** *?case*  
**by** (*rule lderives.LAssm*)  
**next**  
**case** (*LBotE*  $\Gamma l A$ )  
**then show** *?case*  
**by** (*auto intro: lderives.LBotE*)  
**next**  
**case** (*LImpI*  $l A \Gamma m B$ )  
**then have**  $(l, \text{fm-subst } \sigma A) \# \text{map } (\text{map-prod id } (\text{fm-subst } \sigma)) \Gamma$   
 $\vdash L (m, \text{fm-subst } \sigma B)$   
**by** (*simp add: id-def*)  
**then have**  $\text{map } (\text{map-prod id } (\text{fm-subst } \sigma)) \Gamma$   
 $\vdash L (\text{lam } l m, \text{Imp } (\text{fm-subst } \sigma A) (\text{fm-subst } \sigma B))$   
**by** (*rule lderives.LImpI*)  
**then show** *?case*  
**by** (*simp add: id-def*)  
**next**  
**case** (*LImpE*  $\Gamma l A B m$ )  
**then show** *?case*  
**by** (*auto intro: lderives.LImpE*)

```

next
  case (LConI  $\Gamma$   $l$   $A$   $m$   $B$ )
  then show ?case
    by (auto intro: lderives.LConI)
next
  case (LConE1  $\Gamma$   $l$   $A$   $B$ )
  then show ?case
    by (auto intro: lderives.LConE1)
next
  case (LConE2  $\Gamma$   $l$   $A$   $B$ )
  then show ?case
    by (auto intro: lderives.LConE2)
next
  case (LDisI1  $\Gamma$   $l$   $A$   $B$ )
  then show ?case
    by (auto intro: lderives.LDisI1)
next
  case (LDisI2  $\Gamma$   $l$   $B$   $A$ )
  then show ?case
    by (auto intro: lderives.LDisI2)
next
  case (LDisE  $\Gamma$   $l$   $A$   $B$   $m$   $n$   $C$   $p$   $q$ )
  then show ?case
    by (auto intro: lderives.LDisE)
qed

lemma labelled-substitution-atom:
  assumes  $\Gamma \vdash L (l, A)$ 
  shows  $\text{map } (\text{map-prod id } (\text{fm-subst } \sigma)) \Gamma \vdash L (l, \text{fm-subst } \sigma A)$ 
  using labelled-substitution-atom-aux[OF assms] by simp

end

end

theory Erasure
  imports Labelled-Natural-Deduction
begin

  This theory proves that labels are conservative annotations for the parametric kernel: every labelled derivation erases to an ordinary natural deduction derivation of the underlying formula from the erased context.

  context label-structure
  begin

  theorem labelled-sound-under-erasure:
    assumes  $\Gamma \vdash L x$ 
    shows  $\text{erase-ctx } \Gamma \vdash \text{erase-lfm } x$ 

```

```

using assms
proof (induction rule: lderives.induct)
  case (LAssm  $x \Gamma$ )
    have erase-lfm  $x \in \text{set}(\text{erase-ctx } \Gamma)$ 
      using LAssm.hyps by (induction  $\Gamma$ ) (auto simp: erase-ctx-def erase-lfm-def)
    then show ?case
      by (rule derives.Assm)
  next
    case (LBotE  $\Gamma l A$ )
    then show ?case
      by (simp add: derives.BotE)
  next
    case (LImpI  $l A \Gamma m B$ )
    then show ?case
      by (simp add: derives.ImpI)
  next
    case (LImpE  $\Gamma l A B m$ )
    then show ?case
      by (simp add: derives.ImpE)
  next
    case (LConI  $\Gamma l A m B$ )
    then show ?case
      by (simp add: derives.ConI)
  next
    case (LConE1  $\Gamma l A B$ )
    then show ?case
      by (simp add: derives.ConE1)
  next
    case (LConE2  $\Gamma l A B$ )
    then show ?case
      by (simp add: derives.ConE2)
  next
    case (LDisI1  $\Gamma l A B$ )
    then show ?case
      by (simp add: derives.DisI1)
  next
    case (LDisI2  $\Gamma l B A$ )
    then show ?case
      by (simp add: derives.DisI2)
  next
    case (LDisE  $\Gamma l A B m n C p q$ )
    then show ?case
      by (simp add: derives.DisE)
qed

```

**lemmas** *erasure-sound = labelled-sound-under-erasure*

**end**

**end**

**theory** *Label-Algebra*  
**imports** *Erasure*  
**begin**

The label-algebra layer packages the two generic adequacy requirements used by the concrete interpretations. Label erasure targets *unit*, so every erasure map is extensionally the unique one. The lift operation may inspect the labelled context, which is needed for calculi whose assumption labels carry semantic data.

**lemma** *erase-ctx-mem-label*:

**assumes**  $A \in \text{set } (\text{erase-ctx } \Gamma)$   
**obtains**  $l$  **where**  $(l, A) \in \text{set } \Gamma$

**proof** –

**from** *assms* **obtain**  $x$  **where**  $x \in \text{set } \Gamma$  **and**  $\text{snd } x = A$

**unfolding** *erase-ctx-def* **by** *auto*

**obtain**  $l$   $B$  **where**  $x = (l, B)$

**by** (*cases x*)

**with**  $\langle x \in \text{set } \Gamma \rangle \langle \text{snd } x = A \rangle$  **have**  $(l, A) \in \text{set } \Gamma$

**by** *simp*

**then show** *thesis*

**by** (*rule that*)

**qed**

**context** *label-structure*  
**begin**

**lemma** *lifting-general-aux*:

**assumes**  $\Delta \vdash A$  **and**  $\text{erase-ctx } \Gamma = \Delta$

**shows**  $\exists l. \Gamma \vdash L (l, A)$

**using** *assms*

**proof** (*induction arbitrary:  $\Gamma$  rule: derives.induct*)

**case** (*Assm A  $\Delta$* )

**then have**  $A \in \text{set } (\text{erase-ctx } \Gamma)$

**by** *simp*

**then obtain**  $l$  **where**  $(l, A) \in \text{set } \Gamma$

**by** (*rule erase-ctx-mem-label*)

**then have**  $\Gamma \vdash L (l, A)$

**by** (*rule LAssm*)

**then show** *?case*

**by** *blast*

**next**

**case** (*BotE  $\Delta$  A*)

**then obtain**  $l$  **where**  $\Gamma \vdash L (l, \text{Bot})$

**by** *blast*

**then have**  $\Gamma \vdash L (\text{abort } l, A)$

**by** (*rule LBotE*)

```

then show ?case
  by blast
next
case (ImpI A Δ B)
have erase-ctx ((undefined, A) # Γ) = A # Δ
  using ImpI.premis by simp
then obtain m where (undefined, A) # Γ ⊢L (m, B)
  using ImpI.IH by blast
then have Γ ⊢L (lam undefined m, Imp A B)
  by (rule LImpI)
then show ?case
  by blast
next
case (ImpE Δ A B)
then obtain l where imp: Γ ⊢L (l, Imp A B)
  by blast
from ImpE obtain m where arg: Γ ⊢L (m, A)
  by blast
from imp arg have Γ ⊢L (app l m, B)
  by (rule LImpE)
then show ?case
  by blast
next
case (ConI Δ A B)
then obtain l where left: Γ ⊢L (l, A)
  by blast
from ConI obtain m where right: Γ ⊢L (m, B)
  by blast
from left right have Γ ⊢L (pair l m, Con A B)
  by (rule LConI)
then show ?case
  by blast
next
case (ConE1 Δ A B)
then obtain l where Γ ⊢L (l, Con A B)
  by blast
then have Γ ⊢L (fst-l l, A)
  by (rule LConE1)
then show ?case
  by blast
next
case (ConE2 Δ A B)
then obtain l where Γ ⊢L (l, Con A B)
  by blast
then have Γ ⊢L (snd-l l, B)
  by (rule LConE2)
then show ?case
  by blast
next

```

```

case (DisI1  $\Delta$  A B)
then obtain l where  $\Gamma \vdash L$  (l, A)
  by blast
then have  $\Gamma \vdash L$  (inl l, Dis A B)
  by (rule LDisI1)
then show ?case
  by blast
next
case (DisI2  $\Delta$  B A)
then obtain l where  $\Gamma \vdash L$  (l, B)
  by blast
then have  $\Gamma \vdash L$  (inr l, Dis A B)
  by (rule LDisI2)
then show ?case
  by blast
next
case (DisE  $\Delta$  A B C)
then obtain l where disj:  $\Gamma \vdash L$  (l, Dis A B)
  by blast
have left-ctx: erase-ctx ((undefined, A) #  $\Gamma$ ) = A #  $\Delta$ 
  using DisE.prem1 by simp
then obtain n where left: (undefined, A) #  $\Gamma \vdash L$  (n, C)
  using DisE.IH(2) by blast
have right-ctx: erase-ctx ((undefined, B) #  $\Gamma$ ) = B #  $\Delta$ 
  using DisE.prem2 by simp
then obtain q where right: (undefined, B) #  $\Gamma \vdash L$  (q, C)
  using DisE.IH(3) by blast
from disj left right have  $\Gamma \vdash L$  (cases l undefined n undefined q, C)
  by (rule LDisE)
then show ?case
  by blast
qed

```

```

theorem lifting-general:
  assumes erase-ctx  $\Gamma \vdash A$ 
  shows  $\exists l. \Gamma \vdash L$  (l, A)
  using lifting-general-aux[OF assms refl] .

```

end

```

locale label-algebra =
  fixes app :: 'l  $\Rightarrow$  'l  $\Rightarrow$  'l
    and lam :: 'l  $\Rightarrow$  'l  $\Rightarrow$  'l
    and pair :: 'l  $\Rightarrow$  'l  $\Rightarrow$  'l
    and fst-l :: 'l  $\Rightarrow$  'l
    and snd-l :: 'l  $\Rightarrow$  'l
    and abort :: 'l  $\Rightarrow$  'l
    and inl :: 'l  $\Rightarrow$  'l
    and inr :: 'l  $\Rightarrow$  'l

```

**and** *cases* :: 'l ⇒ 'l ⇒ 'l ⇒ 'l ⇒ 'l ⇒ 'l  
**and** *lderives* :: ('l, 'f) lfm list ⇒ ('l, 'f) lfm ⇒ bool  
(- ⊢L - [50, 50] 50)  
**and** *erase-label* :: 'l ⇒ unit  
**and** *lift-label* :: ('l, 'f) lfm list ⇒ 'f fm ⇒ 'l  
**assumes** *erase-label-unique* [simp]: *erase-label* l = ()  
**and** *erasure-sound-law*:  
Γ ⊢L x ⇒ *erase-ctx* Γ ⊢ *erase-lfm* x  
**and** *lift-coherence*:  
*erase-ctx* Γ ⊢ A ⇒ Γ ⊢L (*lift-label* Γ A, A)  
**begin**

**definition** *erase-judgement* ::  
(('l, 'f) lfm list × ('l, 'f) lfm) ⇒ ('f fm list × 'f fm)  
**where** *erase-judgement* J = (*erase-ctx* (fst J), *erase-lfm* (snd J))

**lemma** *erase-judgement-simps* [simp]:  
*erase-judgement* (Γ, (l, A)) = (*erase-ctx* Γ, A)  
**by** (*simp add: erase-judgement-def*)

**lemma** *erasure-uniformity*:  
*erase-label* (*app* l m) = ()  
*erase-label* (*lam* l m) = ()  
*erase-label* (*pair* l m) = ()  
*erase-label* (*fst-l* l) = ()  
*erase-label* (*snd-l* l) = ()  
*erase-label* (*abort* l) = ()  
*erase-label* (*inl* l) = ()  
*erase-label* (*inr* l) = ()  
*erase-label* (*cases* l m n p q) = ()  
**by** *simp-all*

**lemma** *erasure-uniformity-judgement*:  
*erase-judgement* (Γ, (*app* l m, A)) = (*erase-ctx* Γ, A)  
*erase-judgement* (Γ, (*lam* l m, *Imp* A B)) = (*erase-ctx* Γ, *Imp* A B)  
*erase-judgement* (Γ, (*pair* l m, *Con* A B)) = (*erase-ctx* Γ, *Con* A B)  
*erase-judgement* (Γ, (*fst-l* l, A)) = (*erase-ctx* Γ, A)  
*erase-judgement* (Γ, (*snd-l* l, B)) = (*erase-ctx* Γ, B)  
*erase-judgement* (Γ, (*abort* l, A)) = (*erase-ctx* Γ, A)  
*erase-judgement* (Γ, (*inl* l, *Dis* A B)) = (*erase-ctx* Γ, *Dis* A B)  
*erase-judgement* (Γ, (*inr* l, *Dis* A B)) = (*erase-ctx* Γ, *Dis* A B)  
*erase-judgement* (Γ, (*cases* l m n p q, C)) = (*erase-ctx* Γ, C)  
**by** *simp-all*

**theorem** *framework-adequacy*:  
**fixes** ΓL :: ('l, 'f) lfm list  
**and** A :: 'f fm  
**shows** (∃ l. ΓL ⊢L (l, A)) ↔  
(*erase-ctx* ΓL ⊢ A ∧ ΓL ⊢L (*lift-label* ΓL A, A))

```

proof
  assume  $\exists l. \Gamma L \vdash L (l, A)$ 
  then obtain  $l$  where  $deriv: \Gamma L \vdash L (l, A)$ 
    by blast
  have  $erase-ctx \Gamma L \vdash erase-lfm (l, A)$ 
    using  $deriv$  by (rule erasure-sound-law)
  then have  $ordinary: erase-ctx \Gamma L \vdash A$ 
    by simp
  moreover have  $\Gamma L \vdash L (lift-label \Gamma L A, A)$ 
    using  $ordinary$  by (rule lift-coherence)
  ultimately show  $erase-ctx \Gamma L \vdash A \wedge \Gamma L \vdash L (lift-label \Gamma L A, A)$ 
    by blast
next
  assume  $erase-ctx \Gamma L \vdash A \wedge \Gamma L \vdash L (lift-label \Gamma L A, A)$ 
  then show  $\exists l. \Gamma L \vdash L (l, A)$ 
    by blast
qed

```

```

corollary framework-adequacy-iff:
  fixes  $\Gamma L :: ('l, 'f) lfm\ list$ 
    and  $A :: 'f\ fm$ 
  shows  $(\exists l. \Gamma L \vdash L (l, A)) \longleftrightarrow erase-ctx \Gamma L \vdash A$ 

```

```

proof
  assume  $\exists l. \Gamma L \vdash L (l, A)$ 
  then show  $erase-ctx \Gamma L \vdash A$ 
    using framework-adequacy by blast
next
  assume  $ordinary: erase-ctx \Gamma L \vdash A$ 
  then have  $\Gamma L \vdash L (lift-label \Gamma L A, A)$ 
    by (rule lift-coherence)
  with  $ordinary$  show  $\exists l. \Gamma L \vdash L (l, A)$ 
    using framework-adequacy by blast
qed

```

**end**

**end**

```

theory Lifting
  imports Label-Algebra
begin

```

This theory supplies a concrete datatype of proof labels and interprets the abstract label-structure locale with its constructors. With these labels, ordinary derivations can always be decorated to obtain a labelled derivation over any labelled context whose erasure is the ordinary context.

```

datatype label =
  Hyp nat

```

```

| App label label
| Lam label label
| Pair label label
| FstL label
| SndL label
| Inl label
| Inr label
| Cases label label label label label
| Abort label

```

**interpretation** *default-labels: label-structure*

```

where app = App
and lam = Lam
and pair = Pair
and fst-l = FstL
and snd-l = SndL
and abort = Abort
and inl = Inl
and inr = Inr
and cases = Cases

```

.

**definition** *default-erase-label :: label  $\Rightarrow$  unit* **where**

```

default-erase-label l = ()

```

**definition** *default-lift-label :: (label, 'a) lfm list  $\Rightarrow$  'a fm  $\Rightarrow$  label* **where**

```

default-lift-label  $\Gamma$  A =
  (SOME l. default-labels.lderives  $\Gamma$  (l, A))

```

**interpretation** *default-labels: label-algebra*

```

where app = App
and lam = Lam
and pair = Pair
and fst-l = FstL
and snd-l = SndL
and abort = Abort
and inl = Inl
and inr = Inr
and cases = Cases
and lderives = default-labels.lderives
and erase-label = default-erase-label
and lift-label = default-lift-label

```

**proof**

```

fix l
show default-erase-label l = ()
  by (simp add: default-erase-label-def)

```

**next**

```

fix  $\Gamma$  :: (label, 'a) lfm list and x :: (label, 'a) lfm
assume default-labels.lderives  $\Gamma$  x

```

```

then show erase-ctx  $\Gamma \vdash \text{erase-lfm } x$ 
  by (rule default-labels.erasure-sound)
next
fix  $\Gamma :: (\text{label}, 'a) \text{ lfm list and } A :: 'a \text{ fm}$ 
assume ordinary: erase-ctx  $\Gamma \vdash A$ 
have  $\exists l. \text{default-labels.lderives } \Gamma (l, A)$ 
  by (rule default-labels.lifting-general[OF ordinary])
then show default-labels.lderives  $\Gamma (\text{default-lift-label } \Gamma A, A)$ 
  unfolding default-lift-label-def by (rule someI-ex)
qed

lemma erase-ctx-mem-ex:
assumes  $A \in \text{set } (\text{erase-ctx } \Gamma)$ 
shows  $\exists l. (l, A) \in \text{set } \Gamma$ 
using assms
by (induction  $\Gamma$ ) (auto simp: erase-ctx-def)

lemma erase-ctx-mem-witness:
assumes  $A \in \text{set } (\text{erase-ctx } \Gamma)$ 
obtains  $l$  where  $(l, A) \in \text{set } \Gamma$ 
using erase-ctx-mem-ex[OF assms] by blast

lemma ordinary-derivations-have-labelling-aux:
assumes  $\Delta \vdash A$  and erase-ctx  $\Gamma = \Delta$ 
shows  $\exists l. \text{default-labels.lderives } \Gamma (l, A)$ 
using assms
proof (induction arbitrary:  $\Gamma$  rule: derives.induct)
case (Assm  $A \Delta$ )
then have  $A \in \text{set } (\text{erase-ctx } \Gamma)$ 
  by simp
then obtain  $l$  where  $(l, A) \in \text{set } \Gamma$ 
  by (rule erase-ctx-mem-witness)
then have default-labels.lderives  $\Gamma (l, A)$ 
  by (rule default-labels.LAssm)
then show ?case
  by blast
next
case (BotE  $\Delta A$ )
then obtain  $l$  where default-labels.lderives  $\Gamma (l, \text{Bot})$ 
  by blast
then have default-labels.lderives  $\Gamma (\text{Abort } l, A)$ 
  by (rule default-labels.LBotE)
then show ?case
  by blast
next
case (ImpI  $A \Delta B$ )
have erase-ctx  $((\text{Hyp } 0, A) \# \Gamma) = A \# \Delta$ 
  using ImpI.premis by simp
then obtain  $m$  where default-labels.lderives  $((\text{Hyp } 0, A) \# \Gamma) (m, B)$ 

```

```

    using ImpI.IH by blast
  then have default-labels.lderives  $\Gamma$  (Lam (Hyp 0) m, Imp A B)
    by (rule default-labels.LImpI)
  then show ?case
    by blast
next
case (ImpE  $\Delta$  A B)
then obtain l where imp: default-labels.lderives  $\Gamma$  (l, Imp A B)
  by blast
from ImpE obtain m where arg: default-labels.lderives  $\Gamma$  (m, A)
  by blast
from imp arg have default-labels.lderives  $\Gamma$  (App l m, B)
  by (rule default-labels.LImpE)
then show ?case
  by blast
next
case (ConI  $\Delta$  A B)
then obtain l where left: default-labels.lderives  $\Gamma$  (l, A)
  by blast
from ConI obtain m where right: default-labels.lderives  $\Gamma$  (m, B)
  by blast
from left right have default-labels.lderives  $\Gamma$  (Pair l m, Con A B)
  by (rule default-labels.LConI)
then show ?case
  by blast
next
case (ConE1  $\Delta$  A B)
then obtain l where default-labels.lderives  $\Gamma$  (l, Con A B)
  by blast
then have default-labels.lderives  $\Gamma$  (FstL l, A)
  by (rule default-labels.LConE1)
then show ?case
  by blast
next
case (ConE2  $\Delta$  A B)
then obtain l where default-labels.lderives  $\Gamma$  (l, Con A B)
  by blast
then have default-labels.lderives  $\Gamma$  (SndL l, B)
  by (rule default-labels.LConE2)
then show ?case
  by blast
next
case (DisI1  $\Delta$  A B)
then obtain l where default-labels.lderives  $\Gamma$  (l, A)
  by blast
then have default-labels.lderives  $\Gamma$  (Inl l, Dis A B)
  by (rule default-labels.LDisI1)
then show ?case
  by blast

```

```

next
  case (DisI2  $\Delta$  B A)
  then obtain l where default-labels.lderives  $\Gamma$  (l, B)
    by blast
  then have default-labels.lderives  $\Gamma$  (Inr l, Dis A B)
    by (rule default-labels.LDisI2)
  then show ?case
    by blast
next
  case (DisE  $\Delta$  A B C)
  then obtain l where disj: default-labels.lderives  $\Gamma$  (l, Dis A B)
    by blast
  have left-ctx: erase-ctx ((Hyp 0, A) #  $\Gamma$ ) = A #  $\Delta$ 
    using DisE.prem1 by simp
  then obtain n where left: default-labels.lderives ((Hyp 0, A) #  $\Gamma$ ) (n, C)
    using DisE.IH(2) by blast
  have right-ctx: erase-ctx ((Hyp 1, B) #  $\Gamma$ ) = B #  $\Delta$ 
    using DisE.prem2 by simp
  then obtain q where right: default-labels.lderives ((Hyp 1, B) #  $\Gamma$ ) (q, C)
    using DisE.IH(3) by blast
  from disj left right have default-labels.lderives  $\Gamma$  (Cases l (Hyp 0) n (Hyp 1) q,
  C)
    by (rule default-labels.LDisE)
  then show ?case
    by blast
qed

theorem ordinary-derivations-have-labelling:
  assumes erase-ctx  $\Gamma \vdash A$ 
  shows  $\exists l.$  default-labels.lderives  $\Gamma$  (l, A)
proof -
  have default-labels.lderives  $\Gamma$  (default-lift-label  $\Gamma$  A, A)
    using assms by (rule default-labels.lift-coherence)
  with assms show ?thesis
    using default-labels.framework-adequacy by blast
qed

theorem ordinary-derivations-admit-labelling:
  assumes erase-ctx  $\Gamma \vdash A$ 
  shows  $\exists l.$  default-labels.lderives  $\Gamma$  (l, A)
  using assms ordinary-derivations-have-labelling by blast

end

theory Unit-Labels
  imports Label-Algebra
begin

```

This theory instantiates the labelled kernel with a singleton label type.

Since all label constructors return the unique label, labelled derivability over the lifted context is equivalent to ordinary natural deduction.

**datatype** *unit-label* = *Star*

**interpretation** *unit-labels*: *label-structure*

**where** *app* =  $\lambda- \cdot. Star$   
**and** *lam* =  $\lambda- \cdot. Star$   
**and** *pair* =  $\lambda- \cdot. Star$   
**and** *fst-l* =  $\lambda-. Star$   
**and** *snd-l* =  $\lambda-. Star$   
**and** *abort* =  $\lambda-. Star$   
**and** *inl* =  $\lambda-. Star$   
**and** *inr* =  $\lambda-. Star$   
**and** *cases* =  $\lambda- - - -. Star$

**definition** *unit-erase-label* :: *unit-label*  $\Rightarrow$  *unit* **where**

*unit-erase-label* *l* = ()

**definition** *unit-lift-label* :: (*unit-label*, 'a) *lfm list*  $\Rightarrow$  'a *fm*  $\Rightarrow$  *unit-label* **where**

*unit-lift-label*  $\Gamma$  *A* = *Star*

**interpretation** *unit-labels*: *label-algebra*

**where** *app* =  $\lambda- \cdot. Star$   
**and** *lam* =  $\lambda- \cdot. Star$   
**and** *pair* =  $\lambda- \cdot. Star$   
**and** *fst-l* =  $\lambda-. Star$   
**and** *snd-l* =  $\lambda-. Star$   
**and** *abort* =  $\lambda-. Star$   
**and** *inl* =  $\lambda-. Star$   
**and** *inr* =  $\lambda-. Star$   
**and** *cases* =  $\lambda- - - -. Star$   
**and** *lderives* = *unit-labels.lderives*  
**and** *erase-label* = *unit-erase-label*  
**and** *lift-label* = *unit-lift-label*

**proof**

**fix** *l*  
**show** *unit-erase-label* *l* = ()  
**by** (*simp add: unit-erase-label-def*)

**next**

**fix**  $\Gamma$  :: (*unit-label*, 'a) *lfm list* **and** *x* :: (*unit-label*, 'a) *lfm*  
**assume** *unit-labels.lderives*  $\Gamma$  *x*  
**then show** *erase-ctx*  $\Gamma \vdash$  *erase-lfm* *x*  
**by** (*rule unit-labels.erasure-sound*)

**next**

**fix**  $\Gamma$  :: (*unit-label*, 'a) *lfm list* **and** *A* :: 'a *fm*  
**assume** *ordinary: erase-ctx*  $\Gamma \vdash$  *A*  
**then obtain** *l* **where** *deriv: unit-labels.lderives*  $\Gamma$  (*l*, *A*)  
**using** *unit-labels.lifting-general* **by** *blast*

```

then show unit-labels.lderives  $\Gamma$  (unit-lift-label  $\Gamma$  A, A)
  by (cases l) (simp add: unit-lift-label-def)
qed

definition unit-lift :: 'a fm list  $\Rightarrow$  (unit-label  $\times$  'a fm) list where
  unit-lift  $\Gamma$  = map ( $\lambda A.$  (Star, A))  $\Gamma$ 

lemma erase-ctx-unit-lift [simp]:
  erase-ctx (unit-lift  $\Gamma$ ) =  $\Gamma$ 
  by (simp add: unit-lift-def erase-ctx-def comp-def)

lemma unit-labels-from-ordinary:
  assumes  $\Gamma \vdash A$ 
  shows unit-labels.lderives (unit-lift  $\Gamma$ ) (Star, A)
  using assms
proof (induction rule: derives.induct)
  case (Assm A  $\Gamma$ )
  then have (Star, A)  $\in$  set (unit-lift  $\Gamma$ )
    by (auto simp: unit-lift-def)
  then show ?case
    by (rule unit-labels.LAssm)
next
  case (BotE  $\Gamma$  A)
  have unit-labels.lderives (unit-lift  $\Gamma$ ) (( $\lambda -.$  Star) Star, A)
    using BotE.IH by (rule unit-labels.LBotE)
  then show ?case
    by simp
next
  case (ImpI A  $\Gamma$  B)
  have unit-labels.lderives ((Star, A) # unit-lift  $\Gamma$ ) (Star, B)
    using ImpI.IH by (simp add: unit-lift-def)
  then show ?case
    by (rule unit-labels.LImpI)
next
  case (ImpE  $\Gamma$  A B)
  have unit-labels.lderives (unit-lift  $\Gamma$ ) (( $\lambda -.$  Star) Star Star, B)
    using ImpE.IH by (rule unit-labels.LImpE)
  then show ?case
    by simp
next
  case (ConI  $\Gamma$  A B)
  have unit-labels.lderives (unit-lift  $\Gamma$ ) (( $\lambda -.$  Star) Star Star, Con A B)
    using ConI.IH by (rule unit-labels.LConI)
  then show ?case
    by simp
next
  case (ConE1  $\Gamma$  A B)
  have unit-labels.lderives (unit-lift  $\Gamma$ ) (( $\lambda -.$  Star) Star, A)
    using ConE1.IH by (rule unit-labels.LConE1)

```

```

then show ?case
  by simp
next
case (ConE2  $\Gamma$  A B)
have unit-labels.lderives (unit-lift  $\Gamma$ ) (( $\lambda$ -. Star) Star, B)
  using ConE2.IH by (rule unit-labels.LConE2)
then show ?case
  by simp
next
case (DisI1  $\Gamma$  A B)
have unit-labels.lderives (unit-lift  $\Gamma$ ) (( $\lambda$ -. Star) Star, Dis A B)
  using DisI1.IH by (rule unit-labels.LDisI1)
then show ?case
  by simp
next
case (DisI2  $\Gamma$  B A)
have unit-labels.lderives (unit-lift  $\Gamma$ ) (( $\lambda$ -. Star) Star, Dis A B)
  using DisI2.IH by (rule unit-labels.LDisI2)
then show ?case
  by simp
next
case (DisE  $\Gamma$  A B C)
have disj: unit-labels.lderives (unit-lift  $\Gamma$ ) (Star, Dis A B)
  by (rule DisE.IH(1))
have left: unit-labels.lderives ((Star, A) # unit-lift  $\Gamma$ ) (Star, C)
  using DisE.IH(2) by (simp add: unit-lift-def)
have right: unit-labels.lderives ((Star, B) # unit-lift  $\Gamma$ ) (Star, C)
  using DisE.IH(3) by (simp add: unit-lift-def)
have unit-labels.lderives (unit-lift  $\Gamma$ ) (( $\lambda$ -. - - -. Star) Star Star Star Star Star,
C)
  using disj left right by (rule unit-labels.LDisE)
then show ?case
  by simp
qed

```

**theorem** unit-labels-conservative:

$\Gamma \vdash A \longleftrightarrow$  unit-labels.lderives (unit-lift  $\Gamma$ ) (Star, A)

**proof**

**assume** ordinary:  $\Gamma \vdash A$

**then have** ordinary-erased: erase-ctx (unit-lift  $\Gamma$ )  $\vdash$  A

**by simp**

**have** unit-labels.lderives

(unit-lift  $\Gamma$ ) (unit-lift-label (unit-lift  $\Gamma$ ) A, A)

**using** ordinary-erased **by** (rule unit-labels.lift-coherence)

**with** ordinary-erased **have**  $\exists l$ . unit-labels.lderives (unit-lift  $\Gamma$ ) (l, A)

**using** unit-labels.framework-adequacy **by** blast

**then obtain** l **where** unit-labels.lderives (unit-lift  $\Gamma$ ) (l, A)

**by** blast

**then show** unit-labels.lderives (unit-lift  $\Gamma$ ) (Star, A)

```

    by (cases l) simp
next
assume unit-labels.lderives (unit-lift  $\Gamma$ ) (Star, A)
then have  $\exists l. \text{unit-labels.lderives } (\text{unit-lift } \Gamma) (l, A)$ 
  by blast
then have erase-ctx (unit-lift  $\Gamma$ )  $\vdash$  A
  using unit-labels.framework-adequacy by blast
then show  $\Gamma \vdash A$ 
  by simp
qed

end

```

```

theory Provenance-Labels
  imports Label-Algebra
begin

```

This theory instantiates labels by sets of natural-number assumption indices. Application and pairing combine provenance by union, implication introduction removes the discharged assumption label, and disjunction elimination records the analysed disjunction together with the branch dependencies that survive discharge.

```

type-synonym prov = nat set

```

```

interpretation provenance: label-structure

```

```

  where app =  $\lambda(S::\text{prov}) T. S \cup T$ 
    and lam =  $\lambda(S::\text{prov}) T. T - S$ 
    and pair =  $\lambda(S::\text{prov}) T. S \cup T$ 
    and fst-l =  $\lambda(S::\text{prov}). S$ 
    and snd-l =  $\lambda(S::\text{prov}). S$ 
    and abort =  $\lambda(S::\text{prov}). S$ 
    and inl =  $\lambda(S::\text{prov}). S$ 
    and inr =  $\lambda(S::\text{prov}). S$ 
    and cases =  $\lambda(S::\text{prov}) (M::\text{prov}) (N::\text{prov}) (P::\text{prov}) (Q::\text{prov}).$ 
       $S \cup (N - M) \cup (Q - P)$ 

```

```

definition provenance-erase-label :: prov  $\Rightarrow$  unit where
  provenance-erase-label S = ()

```

```

definition provenance-lift-label :: (prov, 'a) lfm list  $\Rightarrow$  'a fm  $\Rightarrow$  prov where
  provenance-lift-label  $\Gamma$  A =
    (SOME S. provenance.lderives  $\Gamma$  (S, A))

```

```

interpretation provenance: label-algebra

```

```

  where app =  $\lambda(S::\text{prov}) T. S \cup T$ 
    and lam =  $\lambda(S::\text{prov}) T. T - S$ 
    and pair =  $\lambda(S::\text{prov}) T. S \cup T$ 

```

```

and fst-l =  $\lambda(S::prov). S$ 
and snd-l =  $\lambda(S::prov). S$ 
and abort =  $\lambda(S::prov). S$ 
and inl =  $\lambda(S::prov). S$ 
and inr =  $\lambda(S::prov). S$ 
and cases =  $\lambda(S::prov) (M::prov) (N::prov) (P::prov) (Q::prov).$ 
   $S \cup (N - M) \cup (Q - P)$ 
and lderives = provenance.lderives
and erase-label = provenance-erase-label
and lift-label = provenance-lift-label
proof
  fix S :: prov
  show provenance-erase-label S = ()
    by (simp add: provenance-erase-label-def)
next
  fix  $\Gamma$  :: (prov, 'a) lfm list and x :: (prov, 'a) lfm
  assume provenance.lderives  $\Gamma$  x
  then show erase-ctx  $\Gamma \vdash$  erase-lfm x
    by (rule provenance.erasure-sound)
next
  fix  $\Gamma$  :: (prov, 'a) lfm list and A :: 'a fm
  assume ordinary: erase-ctx  $\Gamma \vdash$  A
  have  $\exists S. \textit{provenance.lderives}$   $\Gamma$  (S, A)
    by (rule provenance.lifting-general[OF ordinary])
  then show provenance.lderives  $\Gamma$  (provenance-lift-label  $\Gamma$  A, A)
    unfolding provenance-lift-label-def by (rule someI-ex)
qed

fun indexed :: nat  $\Rightarrow$  'a fm list  $\Rightarrow$  (prov  $\times$  'a fm) list where
  indexed n [] = []
| indexed n (A #  $\Gamma$ ) = ({n}, A) # indexed (Suc n)  $\Gamma$ 

definition ctx-prov :: (prov, 'a) lfm list  $\Rightarrow$  prov where
  ctx-prov  $\Gamma$  =  $\bigcup$  (fst ' set  $\Gamma$ )

lemma ctx-prov-Cons [simp]:
  ctx-prov ((M, A) #  $\Gamma$ ) = M  $\cup$  ctx-prov  $\Gamma$ 
  unfolding ctx-prov-def
  by auto

lemma erase-ctx-indexed [simp]:
  erase-ctx (indexed n  $\Gamma$ ) =  $\Gamma$ 
  by (induction  $\Gamma$  arbitrary: n) auto

lemma finite-ctx-prov-indexed [simp]:
  finite (ctx-prov (indexed n  $\Gamma$ ))
  by (induction  $\Gamma$  arbitrary: n) (auto simp: ctx-prov-def)

definition filter-indexed-by :: nat set  $\Rightarrow$  'a fm list  $\Rightarrow$  'a fm list where

```

*filter-indexed-by*  $S \Gamma =$   
 $\text{map snd } (\text{filter } (\lambda(i, A). i \in S) (\text{zip } [0..<\text{length } \Gamma] \Gamma))$

**lemma** *filter-indexed-by-mono*:  
**assumes**  $S \subseteq T$   
**shows**  $\text{set } (\text{filter-indexed-by } S \Gamma) \subseteq \text{set } (\text{filter-indexed-by } T \Gamma)$   
**using** *assms*  
**unfolding** *filter-indexed-by-def*  
**by** *auto*

**lemma** *filter-indexed-by-subset-ctx*:  
**shows**  $\text{set } (\text{filter-indexed-by } S \Gamma) \subseteq \text{set } \Gamma$   
**unfolding** *filter-indexed-by-def*  
**by** (*auto simp: in-set-zip*)

**lemma** *filter-indexed-by-empty* [*simp*]:  
 $\text{filter-indexed-by } \{\} \Gamma = []$   
**unfolding** *filter-indexed-by-def*  
**by** *auto*

**lemma** *filter-indexed-by-union*:  
 $\text{set } (\text{filter-indexed-by } (S \cup T) \Gamma) =$   
 $\text{set } (\text{filter-indexed-by } S \Gamma) \cup \text{set } (\text{filter-indexed-by } T \Gamma)$   
**unfolding** *filter-indexed-by-def*  
**by** *auto*

**definition** *filter-labeled-by* ::  $\text{prov} \Rightarrow (\text{prov} \times 'a \text{ fm}) \text{ list} \Rightarrow 'a \text{ fm list}$  **where**  
 $\text{filter-labeled-by } S \Gamma =$   
 $\text{map snd } (\text{filter } (\lambda(M, A). M \cap S \neq \{\}) \Gamma)$

**definition** *nonempty-prov-ctx* ::  $(\text{prov} \times 'a \text{ fm}) \text{ list} \Rightarrow \text{bool}$  **where**  
 $\text{nonempty-prov-ctx } \Gamma \longleftrightarrow (\forall x \in \text{set } \Gamma. \text{fst } x \neq \{\})$

**lemma** *nonempty-prov-ctx-Cons* [*simp*]:  
 $\text{nonempty-prov-ctx } ((S, A) \# \Gamma) \longleftrightarrow S \neq \{\} \wedge \text{nonempty-prov-ctx } \Gamma$   
**unfolding** *nonempty-prov-ctx-def*  
**by** *auto*

**lemma** *nonempty-prov-ctx-indexed* [*simp*]:  
 $\text{nonempty-prov-ctx } (\text{indexed } n \Gamma)$   
**by** (*induction*  $\Gamma$  *arbitrary: n*) (*auto simp: nonempty-prov-ctx-def*)

**lemma** *filter-labeled-by-mono*:  
**assumes**  $S \subseteq T$   
**shows**  $\text{set } (\text{filter-labeled-by } S \Gamma) \subseteq \text{set } (\text{filter-labeled-by } T \Gamma)$   
**using** *assms*  
**unfolding** *filter-labeled-by-def*  
**by** *auto*

**lemma** *filter-labeled-by-union-left*:  
 $set (filter-labeled-by S \Gamma) \subseteq set (filter-labeled-by (S \cup T) \Gamma)$   
**by** (rule *filter-labeled-by-mono*) *auto*

**lemma** *filter-labeled-by-union-right*:  
 $set (filter-labeled-by T \Gamma) \subseteq set (filter-labeled-by (S \cup T) \Gamma)$   
**by** (rule *filter-labeled-by-mono*) *auto*

**lemma** *ctx-prov-contains-label*:  
**assumes**  $(M, A) \in set \Gamma$   
**shows**  $M \subseteq ctx-prov \Gamma$   
**using** *assms*  
**unfolding** *ctx-prov-def*  
**by** *auto*

**lemma** *filter-labeled-by-minus-fresh*:  
**assumes**  $M \cap ctx-prov \Gamma = \{\}$   
**shows**  $set (filter-labeled-by N \Gamma) \subseteq set (filter-labeled-by (N - M) \Gamma)$   
**proof**  
**fix**  $A$   
**assume**  $A \in set (filter-labeled-by N \Gamma)$   
**then obtain**  $y$  **where**  $y-in: y \in set \Gamma$   
**and**  $y-used: (case\ y\ of\ (L, B) \Rightarrow L \cap N \neq \{\})$   
**and**  $y-snd: snd\ y = A$   
**unfolding** *filter-labeled-by-def*  
**by** *auto*  
**obtain**  $L\ B$  **where**  $y: y = (L, B)$   
**by** (*cases y*)  
**have**  $in-ctx: (L, A) \in set \Gamma$   
**using**  $y\ y-in\ y-snd$  **by** *auto*  
**have**  $used: L \cap N \neq \{\}$   
**using**  $y\ y-used$  **by** *simp*  
**have**  $L \cap M = \{\}$   
**using** *assms ctx-prov-contains-label[OF in-ctx]* **by** *auto*  
**with**  $used$  **have**  $L \cap (N - M) \neq \{\}$   
**by** *auto*  
**with**  $in-ctx$  **have**  $(L, A) \in$   
 $set (filter (\lambda(K, B). K \cap (N - M) \neq \{\}) \Gamma)$   
**by** *auto*  
**then show**  $A \in set (filter-labeled-by (N - M) \Gamma)$   
**unfolding** *filter-labeled-by-def* **by** *force*  
**qed**

**lemma** *filter-labeled-by-cons-minus-subset*:  
**assumes**  $M \cap ctx-prov \Gamma = \{\}$   
**shows**  $set (filter-labeled-by N ((M, A) \# \Gamma)) \subseteq$   
 $set (A \# filter-labeled-by (N - M) \Gamma)$   
**proof** –  
**have**  $set (filter-labeled-by N ((M, A) \# \Gamma)) \subseteq$

```

      insert A (set (filter-labeled-by N Γ))
    unfolding filter-labeled-by-def
    by auto
  also have ... ⊆ set (A # filter-labeled-by (N - M) Γ)
    using filter-labeled-by-minus-fresh[OF assms] by auto
  finally show ?thesis .
qed

```

```

lemma filter-labeled-by-cons-minus-mono-subset:
  assumes M ∩ ctx-prov Γ = {} and N - M ⊆ U
  shows set (filter-labeled-by N ((M, A) # Γ)) ⊆
    set (A # filter-labeled-by U Γ)
proof -
  have set (filter-labeled-by N ((M, A) # Γ)) ⊆
    set (A # filter-labeled-by (N - M) Γ)
    using assms(1) by (rule filter-labeled-by-cons-minus-subset)
  moreover have set (filter-labeled-by (N - M) Γ) ⊆
    set (filter-labeled-by U Γ)
    using assms(2) by (rule filter-labeled-by-mono)
  ultimately show ?thesis
    by auto
qed

```

```

lemma filter-labeled-by-indexed-upt:
  filter-labeled-by S (indexed n Γ) =
    map snd (filter (λ(i, A). i ∈ S) (zip [n.. $n + \text{length } \Gamma$ ] Γ))
proof (induction Γ arbitrary: n)
  case Nil
  then show ?case
    by (simp add: filter-labeled-by-def)
next
  case (Cons A Γ)
  have upt: [n.. $n + \text{length } (A \# \Gamma)$ ] =
    n # [Suc n.. $\text{Suc } n + \text{length } \Gamma$ ]
  proof -
    have [n.. $n + \text{length } (A \# \Gamma)$ ] = n # [Suc n.. $n + \text{length } (A \# \Gamma)$ ]
      by (rule upt-conv-Cons) simp
    also have ... = n # [Suc n.. $\text{Suc } n + \text{length } \Gamma$ ]
      by simp
    finally show ?thesis .
  qed
  show ?case
    using Cons.IH[of Suc n] upt
    by (auto simp: filter-labeled-by-def)
qed

```

```

lemma filter-labeled-by-indexed-0:
  filter-labeled-by S (indexed 0 Γ) = filter-indexed-by S Γ
  by (simp add: filter-labeled-by-indexed-upt filter-indexed-by-def)

```

**inductive** *prov-safe* :: (prov × 'a fm) list ⇒ prov × 'a fm ⇒ bool  
 (- ⊢P - [50, 50] 50)

**where**

*PAssm*:

$x \in \text{set } \Gamma \implies \Gamma \vdash P x$

| *PBotE*:

$\Gamma \vdash P (S, \text{Bot}) \implies \Gamma \vdash P (S, A)$

| *PImpI*:

$(M, A) \# \Gamma \vdash P (N, B) \implies$

$M \cap \text{ctx-prov } \Gamma = \{\} \implies$

$M \neq \{\} \implies$

$\Gamma \vdash P (N - M, \text{Imp } A B)$

| *PImpE*:

$\Gamma \vdash P (S, \text{Imp } A B) \implies$

$\Gamma \vdash P (T, A) \implies$

$\Gamma \vdash P (S \cup T, B)$

| *PConI*:

$\Gamma \vdash P (S, A) \implies$

$\Gamma \vdash P (T, B) \implies$

$\Gamma \vdash P (S \cup T, \text{Con } A B)$

| *PConE1*:

$\Gamma \vdash P (S, \text{Con } A B) \implies \Gamma \vdash P (S, A)$

| *PConE2*:

$\Gamma \vdash P (S, \text{Con } A B) \implies \Gamma \vdash P (S, B)$

| *PDisI1*:

$\Gamma \vdash P (S, A) \implies \Gamma \vdash P (S, \text{Dis } A B)$

| *PDisI2*:

$\Gamma \vdash P (S, B) \implies \Gamma \vdash P (S, \text{Dis } A B)$

| *PDisE*:

$\Gamma \vdash P (S, \text{Dis } A B) \implies$

$(M, A) \# \Gamma \vdash P (N, C) \implies$

$(P, B) \# \Gamma \vdash P (Q, C) \implies$

$M \cap \text{ctx-prov } \Gamma = \{\} \implies M \neq \{\} \implies$

$P \cap \text{ctx-prov } \Gamma = \{\} \implies P \neq \{\} \implies$

$M \cap P = \{\} \implies$

$\Gamma \vdash P (S \cup (N - M) \cup (Q - P), C)$

**lemma** *provenance-labels-in-context*:

**assumes** *provenance.l*derives  $\Gamma x$

**shows**  $\text{fst } x \subseteq \text{ctx-prov } \Gamma$

**using** *assms*

**by** (*induction rule*: *provenance.l*derives.induct) (*auto simp*: *ctx-prov-def*)

**lemma** *ctx-prov-indexed-bound*:

**assumes**  $i \in \text{ctx-prov}$  (*indexed*  $n \Gamma$ )

**shows**  $i < n + \text{length } \Gamma$

**using** *assms*

**proof** (*induction*  $\Gamma$  *arbitrary*:  $n i$ )

```

case Nil
then show ?case
  by (simp add: ctx-prov-def)
next
case (Cons A  $\Gamma$ )
then consider  $i = n \mid i \in \text{ctx-prov } (\text{indexed } (\text{Suc } n) \Gamma)$ 
  by (auto simp: ctx-prov-def)
then show ?case
proof cases
case 1
then show ?thesis
  by simp
next
case 2
then have  $i < \text{Suc } n + \text{length } \Gamma$ 
  by (rule Cons.IH)
then show ?thesis
  by simp
qed
qed

```

**theorem** *provenance-overapproximates-dependencies:*  
**assumes** *provenance.lderives* (*indexed 0  $\Gamma 0$* ) (*S, A*)  
**shows**  $\forall i. i \in S \longrightarrow i < \text{length } \Gamma 0$

```

proof
fix i
show  $i \in S \longrightarrow i < \text{length } \Gamma 0$ 
proof
assume  $i \in S$ 
moreover have  $S \subseteq \text{ctx-prov } (\text{indexed } 0 \Gamma 0)$ 
  using provenance-labels-in-context[OF assms] by simp
ultimately have  $i \in \text{ctx-prov } (\text{indexed } 0 \Gamma 0)$ 
  by auto
then show  $i < \text{length } \Gamma 0$ 
  using ctx-prov-indexed-bound[of i 0  $\Gamma 0$ ] by simp
qed
qed

```

**lemma** *prov-safe-implies-lderives:*

```

assumes  $\Gamma \vdash P x$ 
shows provenance.lderives  $\Gamma x$ 
using assms
by (induction rule: prov-safe.induct)
  (auto intro: provenance.LAssm provenance.LBotE provenance.LImpI
provenance.LImpE provenance.LConI provenance.LConE1 provenance.LConE2
provenance.LDisI1 provenance.LDisI2 provenance.LDisE)

```

**lemma** *provenance-drop-unused-general:*

```

assumes  $\Gamma \vdash P x$  and nonempty-prov-ctx  $\Gamma$ 

```

```

shows filter-labeled-by (fst x)  $\Gamma \vdash \text{snd } x$ 
using assms
proof (induction rule: prov-safe.induct)
case (PAssm x  $\Gamma$ )
obtain M A where x: x = (M, A)
by (cases x)
then have in-ctx: (M, A)  $\in$  set  $\Gamma$ 
using PAssm by simp
moreover have M  $\neq$  {}
using PAssm x
unfolding nonempty-prov-ctx-def filter-labeled-by-def
by auto
then have M  $\cap$  M  $\neq$  {}
by auto
ultimately have (M, A)  $\in$ 
set (filter ( $\lambda(L, B). L \cap M \neq \{\}$ )  $\Gamma$ )
by auto
then have A  $\in$  set (filter-labeled-by M  $\Gamma$ )
unfolding filter-labeled-by-def by force
then show ?case
using x by (auto intro: derives.Assm)
next
case (PBotE  $\Gamma$  S A)
then show ?case
by (auto intro: derives.BotE)
next
case (PImpI M A  $\Gamma$  N B)
have ne: nonempty-prov-ctx ((M, A) #  $\Gamma$ )
using PImpI by simp
have body: filter-labeled-by N ((M, A) #  $\Gamma$ )  $\vdash$  B
using PImpI.IH[OF ne] by simp
have sub: set (filter-labeled-by N ((M, A) #  $\Gamma$ ))  $\subseteq$ 
set (A # filter-labeled-by (N - M)  $\Gamma$ )
using PImpI by (intro filter-labeled-by-cons-minus-subset) auto
have filter-labeled-by (N - M)  $\Gamma \vdash$  Imp A B
proof (rule derives.ImpI)
show A # filter-labeled-by (N - M)  $\Gamma \vdash$  B
by (rule weakening[OF body sub])
qed
then show ?case
by simp
next
case (PImpE  $\Gamma$  S A B T)
have imp0: filter-labeled-by S  $\Gamma \vdash$  Imp A B
using PImpE.IH(1)[OF PImpE.prem] by simp
have arg0: filter-labeled-by T  $\Gamma \vdash$  A
using PImpE.IH(2)[OF PImpE.prem] by simp
have imp: filter-labeled-by (S  $\cup$  T)  $\Gamma \vdash$  Imp A B
by (rule weakening[OF imp0 filter-labeled-by-union-left])

```

```

have arg: filter-labeled-by ( $S \cup T$ )  $\Gamma \vdash A$ 
  by (rule weakening[OF arg0 filter-labeled-by-union-right])
have filter-labeled-by ( $S \cup T$ )  $\Gamma \vdash B$ 
  by (rule derives.ImpE[OF imp arg])
then show ?case
  by simp
next
case (PConI  $\Gamma S A T B$ )
have left0: filter-labeled-by  $S \Gamma \vdash A$ 
  using PConI.IH(1)[OF PConI.prems] by simp
have right0: filter-labeled-by  $T \Gamma \vdash B$ 
  using PConI.IH(2)[OF PConI.prems] by simp
have left: filter-labeled-by ( $S \cup T$ )  $\Gamma \vdash A$ 
  by (rule weakening[OF left0 filter-labeled-by-union-left])
have right: filter-labeled-by ( $S \cup T$ )  $\Gamma \vdash B$ 
  by (rule weakening[OF right0 filter-labeled-by-union-right])
have filter-labeled-by ( $S \cup T$ )  $\Gamma \vdash \text{Con } A B$ 
  by (rule derives.ConI[OF left right])
then show ?case
  by simp
next
case (PConE1  $\Gamma S A B$ )
then show ?case
  by (auto intro: derives.ConE1)
next
case (PConE2  $\Gamma S A B$ )
then show ?case
  by (auto intro: derives.ConE2)
next
case (PDisI1  $\Gamma S A B$ )
then show ?case
  by (auto intro: derives.DisI1)
next
case (PDisI2  $\Gamma S B A$ )
then show ?case
  by (auto intro: derives.DisI2)
next
case (PDisE  $\Gamma S A B M N C P Q$ )
let ?U =  $S \cup (N - M) \cup (Q - P)$ 
have ne-left: nonempty-prov-ctx ( $(M, A) \# \Gamma$ )
  using PDisE by simp
have ne-right: nonempty-prov-ctx ( $(P, B) \# \Gamma$ )
  using PDisE by simp
have disj0: filter-labeled-by  $S \Gamma \vdash \text{Dis } A B$ 
  using PDisE.IH(1)[OF PDisE.prems] by simp
have left0: filter-labeled-by  $N ((M, A) \# \Gamma) \vdash C$ 
  using PDisE.IH(2)[OF ne-left] by simp
have right0: filter-labeled-by  $Q ((P, B) \# \Gamma) \vdash C$ 
  using PDisE.IH(3)[OF ne-right] by simp

```

```

have disj-sub: set (filter-labeled-by S  $\Gamma$ )  $\subseteq$ 
  set (filter-labeled-by ?U  $\Gamma$ )
  by (rule filter-labeled-by-mono) auto
have disj: filter-labeled-by ?U  $\Gamma \vdash$  Dis A B
  by (rule weakening[OF disj0 disj-sub])
have left-sub: set (filter-labeled-by N ((M, A) #  $\Gamma$ ))  $\subseteq$ 
  set (A # filter-labeled-by ?U  $\Gamma$ )
  using PDisE
  by (intro filter-labeled-by-cons-minus-mono-subset) auto
have left: A # filter-labeled-by ?U  $\Gamma \vdash$  C
  by (rule weakening[OF left0 left-sub])
have right-sub: set (filter-labeled-by Q ((P, B) #  $\Gamma$ ))  $\subseteq$ 
  set (B # filter-labeled-by ?U  $\Gamma$ )
  using PDisE
  by (intro filter-labeled-by-cons-minus-mono-subset) auto
have right: B # filter-labeled-by ?U  $\Gamma \vdash$  C
  by (rule weakening[OF right0 right-sub])
have filter-labeled-by ?U  $\Gamma \vdash$  C
  by (rule derives.DisE[OF disj left right])
then show ?case
  by simp
qed

```

```

theorem provenance-drop-unused:
  assumes indexed 0  $\Gamma 0 \vdash$  P (S, A)
  shows filter-indexed-by S  $\Gamma 0 \vdash$  A
proof –
  have filter-labeled-by S (indexed 0  $\Gamma 0$ )  $\vdash$  A
    using provenance-drop-unused-general[OF assms] by simp
  then show ?thesis
    by (simp add: filter-labeled-by-indexed-0)
qed

```

```

lemma fresh-nat:
  assumes finite F
  obtains i :: nat where i  $\notin$  F
proof –
  have  $\exists i$  :: nat. i  $\notin$  F
    using assms ex-new-if-finite[OF infinite-UNIV-nat] by blast
  then show ?thesis
    using that by blast
qed

```

```

lemma fresh-nat2:
  assumes finite F
  obtains i j :: nat where i  $\notin$  F and j  $\notin$  F and i  $\neq$  j
proof –
  obtain i where i: i  $\notin$  F
    using assms by (rule fresh-nat)

```

```

from assms have finite (insert i F)
  by simp
then obtain j where j: j  $\notin$  insert i F
  by (rule fresh-nat)
from j have j  $\notin$  F and i  $\neq$  j
  by auto
with i show ?thesis
  by (rule that)
qed

lemma ordinary-admits-safe-provenance-labeled:
  assumes  $\Delta \vdash A$ 
    and erase-ctx  $\Gamma = \Delta$ 
    and nonempty-prov-ctx  $\Gamma$ 
    and finite (ctx-prov  $\Gamma$ )
  shows  $\exists S. \Gamma \vdash P(S, A) \wedge S \subseteq \text{ctx-prov } \Gamma$ 
  using assms
proof (induction arbitrary:  $\Gamma$  rule: derives.induct)
  case (Assm A  $\Delta$ )
  then have A  $\in$  set (erase-ctx  $\Gamma$ )
    by simp
  then obtain S where in-ctx: (S, A)  $\in$  set  $\Gamma$ 
    by (rule erase-ctx-mem-label)
  have  $\Gamma \vdash P(S, A)$ 
    using in-ctx by (rule prov-safe.PAssm)
  moreover have S  $\subseteq$  ctx-prov  $\Gamma$ 
    using in-ctx by (rule ctx-prov-contains-label)
  ultimately show ?case
    by blast
next
  case (BotE  $\Delta$  A)
  then obtain S where deriv:  $\Gamma \vdash P(S, \text{Bot})$ 
    and sub: S  $\subseteq$  ctx-prov  $\Gamma$ 
    by blast
  have  $\Gamma \vdash P(S, A)$ 
    by (rule prov-safe.PBotE[OF deriv])
  with sub show ?case
    by blast
next
  case (ImpI A  $\Delta$  B)
  obtain i where fresh: i  $\notin$  ctx-prov  $\Gamma$ 
    using ImpI.prem3 by (rule fresh-nat)
  let ?M =  $\{i\}$ 
  have erase: erase-ctx ((?M, A)  $\#$   $\Gamma$ ) = A  $\#$   $\Delta$ 
    using ImpI.prem1 by simp
  have nonempty: nonempty-prov-ctx ((?M, A)  $\#$   $\Gamma$ )
    using ImpI.prem2 by simp
  have finite: finite (ctx-prov ((?M, A)  $\#$   $\Gamma$ ))
    using ImpI.prem3 by simp

```

```

obtain  $N$  where  $body: (?M, A) \# \Gamma \vdash P(N, B)$ 
  and  $sub-body: N \subseteq ctx-prov ((?M, A) \# \Gamma)$ 
  using  $ImpI.IH[OF\ erase\ nonempty\ finite]$  by  $blast$ 
have  $deriv: \Gamma \vdash P(N - ?M, Imp\ A\ B)$ 
  by  $(rule\ prov-safe.PImpI[OF\ body])$   $(use\ fresh\ in\ auto)$ 
have  $N - ?M \subseteq ctx-prov\ \Gamma$ 
  using  $sub-body\ fresh$  by  $auto$ 
with  $deriv$  show  $?case$ 
  by  $blast$ 
next
case  $(ImpE\ \Delta\ A\ B)$ 
obtain  $S$  where  $imp: \Gamma \vdash P(S, Imp\ A\ B)$ 
  and  $sub-imp: S \subseteq ctx-prov\ \Gamma$ 
  using  $ImpE.IH(1)[OF\ ImpE.premis]$  by  $blast$ 
obtain  $T$  where  $arg: \Gamma \vdash P(T, A)$ 
  and  $sub-arg: T \subseteq ctx-prov\ \Gamma$ 
  using  $ImpE.IH(2)[OF\ ImpE.premis]$  by  $blast$ 
have  $\Gamma \vdash P(S \cup T, B)$ 
  by  $(rule\ prov-safe.PImpE[OF\ imp\ arg])$ 
moreover have  $S \cup T \subseteq ctx-prov\ \Gamma$ 
  using  $sub-imp\ sub-arg$  by  $auto$ 
ultimately show  $?case$ 
  by  $blast$ 
next
case  $(ConI\ \Delta\ A\ B)$ 
obtain  $S$  where  $left: \Gamma \vdash P(S, A)$ 
  and  $sub-left: S \subseteq ctx-prov\ \Gamma$ 
  using  $ConI.IH(1)[OF\ ConI.premis]$  by  $blast$ 
obtain  $T$  where  $right: \Gamma \vdash P(T, B)$ 
  and  $sub-right: T \subseteq ctx-prov\ \Gamma$ 
  using  $ConI.IH(2)[OF\ ConI.premis]$  by  $blast$ 
have  $\Gamma \vdash P(S \cup T, Con\ A\ B)$ 
  by  $(rule\ prov-safe.PConI[OF\ left\ right])$ 
moreover have  $S \cup T \subseteq ctx-prov\ \Gamma$ 
  using  $sub-left\ sub-right$  by  $auto$ 
ultimately show  $?case$ 
  by  $blast$ 
next
case  $(ConE1\ \Delta\ A\ B)$ 
then obtain  $S$  where  $deriv: \Gamma \vdash P(S, Con\ A\ B)$ 
  and  $sub: S \subseteq ctx-prov\ \Gamma$ 
  by  $blast$ 
have  $\Gamma \vdash P(S, A)$ 
  by  $(rule\ prov-safe.PConE1[OF\ deriv])$ 
with  $sub$  show  $?case$ 
  by  $blast$ 
next
case  $(ConE2\ \Delta\ A\ B)$ 
then obtain  $S$  where  $deriv: \Gamma \vdash P(S, Con\ A\ B)$ 

```

```

    and sub:  $S \subseteq \text{ctx-prov } \Gamma$ 
  by blast
  have  $\Gamma \vdash P (S, B)$ 
  by (rule prov-safe.PConE2[OF deriv])
  with sub show ?case
  by blast
next
case (DisI1  $\Delta A B$ )
then obtain  $S$  where deriv:  $\Gamma \vdash P (S, A)$ 
  and sub:  $S \subseteq \text{ctx-prov } \Gamma$ 
  by blast
  have  $\Gamma \vdash P (S, \text{Dis } A B)$ 
  by (rule prov-safe.PDisI1[OF deriv])
  with sub show ?case
  by blast
next
case (DisI2  $\Delta B A$ )
then obtain  $S$  where deriv:  $\Gamma \vdash P (S, B)$ 
  and sub:  $S \subseteq \text{ctx-prov } \Gamma$ 
  by blast
  have  $\Gamma \vdash P (S, \text{Dis } A B)$ 
  by (rule prov-safe.PDisI2[OF deriv])
  with sub show ?case
  by blast
next
case (DisE  $\Delta A B C$ )
obtain  $i j$  where fresh-i:  $i \notin \text{ctx-prov } \Gamma$ 
  and fresh-j:  $j \notin \text{ctx-prov } \Gamma$ 
  and neq:  $i \neq j$ 
  using DisE.prem(3) by (rule fresh-nat2)
let ?M = { $i$ }
let ?P = { $j$ }
obtain  $S$  where disj:  $\Gamma \vdash P (S, \text{Dis } A B)$ 
  and sub-disj:  $S \subseteq \text{ctx-prov } \Gamma$ 
  using DisE.IH(1)[OF DisE.prem] by blast
  have erase-left:  $\text{erase-ctx } ((?M, A) \# \Gamma) = A \# \Delta$ 
  using DisE.prem(1) by simp
  have nonempty-left:  $\text{nonempty-prov-ctx } ((?M, A) \# \Gamma)$ 
  using DisE.prem(2) by simp
  have finite-left:  $\text{finite } (\text{ctx-prov } ((?M, A) \# \Gamma))$ 
  using DisE.prem(3) by simp
  obtain  $N$  where left:  $(?M, A) \# \Gamma \vdash P (N, C)$ 
  and sub-left:  $N \subseteq \text{ctx-prov } ((?M, A) \# \Gamma)$ 
  using DisE.IH(2)[OF erase-left nonempty-left finite-left] by blast
  have erase-right:  $\text{erase-ctx } ((?P, B) \# \Gamma) = B \# \Delta$ 
  using DisE.prem(1) by simp
  have nonempty-right:  $\text{nonempty-prov-ctx } ((?P, B) \# \Gamma)$ 
  using DisE.prem(2) by simp
  have finite-right:  $\text{finite } (\text{ctx-prov } ((?P, B) \# \Gamma))$ 

```

**using** *DisE.premis(3)* **by** *simp*  
**obtain** *Q* **where** *right: (?P, B) # Γ ⊢ P (Q, C)*  
**and** *sub-right: Q ⊆ ctx-prov ((?P, B) # Γ)*  
**using** *DisE.IH(3)[OF erase-right nonempty-right finite-right]* **by** *blast*  
**have** *deriv: Γ ⊢ P (S ∪ (N - ?M) ∪ (Q - ?P), C)*  
**by** (*rule prov-safe.PDisE[OF disj left right]*) (*use fresh-i fresh-j neq in auto*)  
**have** *S ∪ (N - ?M) ∪ (Q - ?P) ⊆ ctx-prov Γ*  
**using** *sub-disj sub-left sub-right fresh-i fresh-j* **by** *auto*  
**with** *deriv* **show** *?case*  
**by** *blast*  
**qed**

**theorem** *ordinary-admits-safe-provenance:*

**assumes**  $\Gamma \vdash A$   
**shows**  $\exists S. (\text{indexed } 0 \Gamma \vdash P (S, A))$   
 $\wedge S \subseteq \{0..<\text{length } \Gamma\}$   
 $\wedge \text{set } (\text{filter-indexed-by } S \Gamma) \subseteq \text{set } \Gamma$

**proof** –

**obtain** *S* **where** *deriv: indexed 0 Γ ⊢ P (S, A)*  
**and** *sub-ctx: S ⊆ ctx-prov (indexed 0 Γ)*  
**using** *ordinary-admits-safe-provenance-labeled[OF assms, of indexed 0 Γ]*  
**by** *simp blast*

**have** *sub-indices: S ⊆ {0..<length Γ}*

**proof**

**fix** *i*  
**assume**  $i \in S$   
**then have**  $i \in \text{ctx-prov } (\text{indexed } 0 \Gamma)$   
**using** *sub-ctx* **by** *auto*  
**then have**  $i < \text{length } \Gamma$   
**using** *ctx-prov-indexed-bound[of i 0 Γ]* **by** *simp*  
**then show**  $i \in \{0..<\text{length } \Gamma\}$   
**by** *simp*

**qed**

**have** *set (filter-indexed-by S Γ) ⊆ set Γ*

**by** (*rule filter-indexed-by-subset-ctx*)

**with** *deriv sub-indices* **show** *?thesis*

**by** *blast*

**qed**

**corollary** *ordinary-admits-tight-provenance:*

**assumes**  $\Gamma \vdash A$   
**shows**  $\exists S. (\text{indexed } 0 \Gamma \vdash P (S, A))$   
 $\wedge S \subseteq \{0..<\text{length } \Gamma\}$   
 $\wedge \text{filter-indexed-by } S \Gamma \vdash A$

**proof** –

**obtain** *S* **where** *deriv: indexed 0 Γ ⊢ P (S, A)*  
**and** *sub: S ⊆ {0..<length Γ}*  
**using** *ordinary-admits-safe-provenance[OF assms]* **by** *blast*  
**have** *filter-indexed-by S Γ ⊢ A*

```

    by (rule provenance-drop-unused[OF deriv])
  with deriv sub show ?thesis
    by blast
qed

theorem provenance-completeness:
  shows  $(\Gamma \vdash A) \longleftrightarrow$ 
         $(\exists S. (\text{indexed } 0 \Gamma \vdash P(S, A)) \wedge \text{filter-indexed-by } S \Gamma \vdash A)$ 
proof
  assume  $\Gamma \vdash A$ 
  then show  $\exists S. (\text{indexed } 0 \Gamma \vdash P(S, A)) \wedge \text{filter-indexed-by } S \Gamma \vdash A$ 
    using ordinary-admits-tight-provenance by blast
next
  assume  $\exists S. (\text{indexed } 0 \Gamma \vdash P(S, A)) \wedge \text{filter-indexed-by } S \Gamma \vdash A$ 
  then obtain  $S$  where deriv:  $\text{indexed } 0 \Gamma \vdash P(S, A)$ 
    by blast
  have  $\text{filter-indexed-by } S \Gamma \vdash A$ 
    by (rule provenance-drop-unused[OF deriv])
  moreover have  $\text{set}(\text{filter-indexed-by } S \Gamma) \subseteq \text{set } \Gamma$ 
    by (rule filter-indexed-by-subset-ctx)
  ultimately show  $\Gamma \vdash A$ 
    by (rule weakening)
qed

end

```

```

theory Modal-Labels-Example
  imports Label-Algebra
begin

```

This theory instantiates the framework locale as *modal-labels* and then extends the resulting propositional labelled calculus with modal *BoxI* and *BoxE* rules. Modal labels carry possible-world annotations; the propositional constructors preserve the current world, while box elimination projects to an accessible world.

```

datatype modal-label =
  MWorld nat
  | MApp modal-label modal-label
  | MLam modal-label modal-label
  | MPair modal-label modal-label
  | MFst modal-label
  | MSnd modal-label
  | MInl modal-label
  | MInr modal-label
  | MCases modal-label modal-label modal-label modal-label modal-label
  | MAbort modal-label
  | MBoxI modal-label
  | MBoxE modal-label

```

```

fun world-of :: modal-label  $\Rightarrow$  nat where
  world-of (MWorld w) = w
| world-of (MApp l m) = world-of l
| world-of (MLam l m) = world-of m
| world-of (MPair l m) = world-of l
| world-of (MFst l) = world-of l
| world-of (MSnd l) = world-of l
| world-of (MInl l) = world-of l
| world-of (MInr l) = world-of l
| world-of (MCases l m n p q) = world-of l
| world-of (MAbort l) = world-of l
| world-of (MBoxI l) = world-of l
| world-of (MBoxE l) = world-of l

```

**interpretation** modal-labels: label-structure

```

where app = MApp
  and lam = MLam
  and pair = MPair
  and fst-l = MFst
  and snd-l = MSnd
  and abort = MAbort
  and inl = MInl
  and inr = MInr
  and cases = MCases

```

.

**definition** modal-erase-label :: modal-label  $\Rightarrow$  unit **where**

```

modal-erase-label l = ()

```

**definition** modal-lift-label :: (modal-label, 'a) lfm list  $\Rightarrow$  'a fm  $\Rightarrow$  modal-label

**where**

```

modal-lift-label  $\Gamma$  A =
  (SOME l. modal-labels.lderives  $\Gamma$  (l, A))

```

**interpretation** modal-labels: label-algebra

```

where app = MApp
  and lam = MLam
  and pair = MPair
  and fst-l = MFst
  and snd-l = MSnd
  and abort = MAbort
  and inl = MInl
  and inr = MInr
  and cases = MCases
  and lderives = modal-labels.lderives
  and erase-label = modal-erase-label
  and lift-label = modal-lift-label

```

**proof**

```

fix l
show modal-erase-label l = ()
  by (simp add: modal-erase-label-def)
next
fix  $\Gamma$  :: (modal-label, 'a) lfm list and x :: (modal-label, 'a) lfm
assume modal-labels.lderives  $\Gamma$  x
then show erase-ctx  $\Gamma \vdash$  erase-lfm x
  by (rule modal-labels.erasure-sound)
next
fix  $\Gamma$  :: (modal-label, 'a) lfm list and A :: 'a fm
assume ordinary: erase-ctx  $\Gamma \vdash$  A
have  $\exists l$ . modal-labels.lderives  $\Gamma$  (l, A)
  by (rule modal-labels.lifting-general[OF ordinary])
then show modal-labels.lderives  $\Gamma$  (modal-lift-label  $\Gamma$  A, A)
  unfolding modal-lift-label-def by (rule someI-ex)
qed

```

```

datatype 'a mfm =
  MAtom 'a
  | MBot
  | MImp 'a mfm 'a mfm
  | MCon 'a mfm 'a mfm
  | MDis 'a mfm 'a mfm
  | Box 'a mfm

```

```

primrec modalise-fm :: 'a fm  $\Rightarrow$  'a mfm where
  modalise-fm (Atom p) = MAtom p
  | modalise-fm Bot = MBot
  | modalise-fm (Imp A B) = MImp (modalise-fm A) (modalise-fm B)
  | modalise-fm (Con A B) = MCon (modalise-fm A) (modalise-fm B)
  | modalise-fm (Dis A B) = MDis (modalise-fm A) (modalise-fm B)

```

```

definition modalise-lfm :: nat  $\Rightarrow$  (modal-label, 'a) lfm  $\Rightarrow$  modal-label  $\times$  'a mfm
where
  modalise-lfm w x = (MWorld w, modalise-fm (snd x))

```

```

primrec modal-kripke-sat ::
  (nat  $\Rightarrow$  'a  $\Rightarrow$  bool)  $\Rightarrow$  (nat  $\Rightarrow$  nat  $\Rightarrow$  bool)  $\Rightarrow$  nat  $\Rightarrow$  'a mfm  $\Rightarrow$  bool
where
  modal-kripke-sat V R w (MAtom p) = V w p
  | modal-kripke-sat V R w MBot = False
  | modal-kripke-sat V R w (MImp A B) =
    (modal-kripke-sat V R w A  $\longrightarrow$  modal-kripke-sat V R w B)
  | modal-kripke-sat V R w (MCon A B) =
    (modal-kripke-sat V R w A  $\wedge$  modal-kripke-sat V R w B)
  | modal-kripke-sat V R w (MDis A B) =
    (modal-kripke-sat V R w A  $\vee$  modal-kripke-sat V R w B)
  | modal-kripke-sat V R w (Box A) =
    ( $\forall v$ . R w v  $\longrightarrow$  modal-kripke-sat V R v A)

```

**inductive** *mlderesives* ::

(*modal-label* × 'a *mfm*) *list* ⇒  
 (nat ⇒ nat ⇒ bool) ⇒ (*modal-label* × 'a *mfm*) ⇒ bool

**where**

*MAssm*:  $x \in \text{set } \Gamma \implies \text{mlderesives } \Gamma R x$

| *MBotE*:  $\text{mlderesives } \Gamma R (l, \text{MBot}) \implies \text{mlderesives } \Gamma R (\text{MAbort } l, A)$

| *MImpI*:

$\text{world-of } l = \text{world-of } m \implies$   
 $\text{mlderesives } ((l, A) \# \Gamma) R (m, B) \implies$   
 $\text{mlderesives } \Gamma R (\text{MLam } l m, \text{MImp } A B)$

| *MImpE*:

$\text{world-of } l = \text{world-of } m \implies$   
 $\text{mlderesives } \Gamma R (l, \text{MImp } A B) \implies$   
 $\text{mlderesives } \Gamma R (m, A) \implies$   
 $\text{mlderesives } \Gamma R (\text{MApp } l m, B)$

| *MConI*:

$\text{world-of } l = \text{world-of } m \implies$   
 $\text{mlderesives } \Gamma R (l, A) \implies$   
 $\text{mlderesives } \Gamma R (m, B) \implies$   
 $\text{mlderesives } \Gamma R (\text{MPair } l m, \text{MCon } A B)$

| *MConE1*:  $\text{mlderesives } \Gamma R (l, \text{MCon } A B) \implies \text{mlderesives } \Gamma R (\text{MFst } l, A)$

| *MConE2*:  $\text{mlderesives } \Gamma R (l, \text{MCon } A B) \implies \text{mlderesives } \Gamma R (\text{MSnd } l, B)$

| *MDisI1*:  $\text{mlderesives } \Gamma R (l, A) \implies \text{mlderesives } \Gamma R (\text{MInl } l, \text{MDis } A B)$

| *MDisI2*:  $\text{mlderesives } \Gamma R (l, B) \implies \text{mlderesives } \Gamma R (\text{MInr } l, \text{MDis } A B)$

| *MDisE*:

$\text{world-of } l = \text{world-of } m \implies$   
 $\text{world-of } l = \text{world-of } n \implies$   
 $\text{world-of } l = \text{world-of } p \implies$   
 $\text{world-of } l = \text{world-of } q \implies$   
 $\text{mlderesives } \Gamma R (l, \text{MDis } A B) \implies$   
 $\text{mlderesives } ((m, A) \# \Gamma) R (n, C) \implies$   
 $\text{mlderesives } ((p, B) \# \Gamma) R (q, C) \implies$   
 $\text{mlderesives } \Gamma R (\text{MCases } l m n p q, C)$

| *MBoxI*:

$(\bigwedge v. R (\text{world-of } l) v) \implies$   
 $\exists m. \text{world-of } m = v \wedge \text{mlderesives } \Gamma R (m, A) \implies$   
 $\text{mlderesives } \Gamma R (\text{MBoxI } l, \text{Box } A)$

| *MBoxE*:

$\text{mlderesives } \Gamma R (l, \text{Box } A) \implies$   
 $R (\text{world-of } l) v \implies$   
 $\text{mlderesives } \Gamma R (\text{MBoxE } (\text{MWorld } v), A)$

**theorem** *propositional-modal-sound*:

**assumes** *modal-labels.lderives*  $\Gamma x$

**shows**  $\exists l. \text{world-of } l = w \wedge$

$\text{mlderesives } (\text{map } (\text{modalise-lfm } w) \Gamma) R (l, \text{modalise-fm } (\text{snd } x))$

**using** *assms*

**proof** (*induction arbitrary: w R rule: modal-labels.lderives.induct*)

```

case (LAssm x  $\Gamma$ )
then show ?case
  by (intro exI[of - MWorld w])
    (auto simp: modalise-lfm-def intro: mlderives.MAssm)
next
case (LBotE  $\Gamma$  l A)
then obtain k where k: world-of k = w
  mlderives (map (modalise-lfm w)  $\Gamma$ ) R (k, MBot)
  by auto
then show ?case
  by (intro exI[of - MAbort k]) (auto intro: mlderives.MBotE)
next
case (LImpI l A  $\Gamma$  m B)
then obtain k where k: world-of k = w
  mlderives (map (modalise-lfm w) ((l, A) #  $\Gamma$ )) R (k, modalise-fm B)
  by auto
then have body: mlderives ((MWorld w, modalise-fm A) #
  map (modalise-lfm w)  $\Gamma$ ) R (k, modalise-fm B)
  by (simp add: modalise-lfm-def)
then have mlderives (map (modalise-lfm w)  $\Gamma$ ) R
  (MLam (MWorld w) k, MImp (modalise-fm A) (modalise-fm B))
  using k(1) by (auto intro: mlderives.MImpI)
then show ?case
  by (intro exI[of - MLam (MWorld w) k]) (simp add: k(1))
next
case (LImpE  $\Gamma$  l A B m)
from LImpE.IH(1)[of w R] obtain k where k: world-of k = w
  mlderives (map (modalise-lfm w)  $\Gamma$ ) R (k, MImp (modalise-fm A) (modalise-fm
  B))
  by auto
from LImpE.IH(2)[of w R] obtain h where h: world-of h = w
  mlderives (map (modalise-lfm w)  $\Gamma$ ) R (h, modalise-fm A)
  by auto
then have mlderives (map (modalise-lfm w)  $\Gamma$ ) R (MApp k h, modalise-fm B)
  using k by (auto intro: mlderives.MImpE)
then show ?case
  by (intro exI[of - MApp k h]) (simp add: k h)
next
case (LConI  $\Gamma$  l A m B)
from LConI.IH(1)[of w R] obtain k where k: world-of k = w
  mlderives (map (modalise-lfm w)  $\Gamma$ ) R (k, modalise-fm A)
  by auto
from LConI.IH(2)[of w R] obtain h where h: world-of h = w
  mlderives (map (modalise-lfm w)  $\Gamma$ ) R (h, modalise-fm B)
  by auto
then have mlderives (map (modalise-lfm w)  $\Gamma$ ) R
  (MPair k h, MCon (modalise-fm A) (modalise-fm B))
  using k by (auto intro: mlderives.MConI)
then show ?case

```

```

    by (intro exI[of - MPair k h]) (simp add: k h)
next
  case (LConE1  $\Gamma$   $l$   $A$   $B$ )
  then obtain  $k$  where  $k$ : world-of  $k = w$ 
    mlderives (map (modalise-lfm  $w$ )  $\Gamma$ )  $R$  ( $k$ , MCon (modalise-fm  $A$ ) (modalise-fm
 $B$ ))
    by auto
  then show ?case
    by (intro exI[of - MFst k]) (auto intro: mlderives.MConE1)
next
  case (LConE2  $\Gamma$   $l$   $A$   $B$ )
  then obtain  $k$  where  $k$ : world-of  $k = w$ 
    mlderives (map (modalise-lfm  $w$ )  $\Gamma$ )  $R$  ( $k$ , MCon (modalise-fm  $A$ ) (modalise-fm
 $B$ ))
    by auto
  then show ?case
    by (intro exI[of - MSnd k]) (auto intro: mlderives.MConE2)
next
  case (LDisI1  $\Gamma$   $l$   $A$   $B$ )
  then obtain  $k$  where  $k$ : world-of  $k = w$ 
    mlderives (map (modalise-lfm  $w$ )  $\Gamma$ )  $R$  ( $k$ , modalise-fm  $A$ )
    by auto
  then show ?case
    by (intro exI[of - MInl k]) (auto intro: mlderives.MDisI1)
next
  case (LDisI2  $\Gamma$   $l$   $B$   $A$ )
  then obtain  $k$  where  $k$ : world-of  $k = w$ 
    mlderives (map (modalise-lfm  $w$ )  $\Gamma$ )  $R$  ( $k$ , modalise-fm  $B$ )
    by auto
  then show ?case
    by (intro exI[of - MInr k]) (auto intro: mlderives.MDisI2)
next
  case (LDisE  $\Gamma$   $l$   $A$   $B$   $m$   $n$   $C$   $p$   $q$ )
  from LDisE.IH(1)[of  $w$   $R$ ] obtain  $d$  where  $d$ :
    world-of  $d = w$ 
    mlderives (map (modalise-lfm  $w$ )  $\Gamma$ )  $R$  ( $d$ , MDis (modalise-fm  $A$ ) (modalise-fm
 $B$ ))
    by auto
  from LDisE.IH(2)[of  $w$   $R$ ] obtain  $left$  where  $left$ :
    world-of  $left = w$ 
    mlderives (map (modalise-lfm  $w$ ) (( $m$ ,  $A$ ) #  $\Gamma$ ))  $R$  ( $left$ , modalise-fm  $C$ )
    by auto
  from LDisE.IH(3)[of  $w$   $R$ ] obtain  $right$  where  $right$ :
    world-of  $right = w$ 
    mlderives (map (modalise-lfm  $w$ ) (( $p$ ,  $B$ ) #  $\Gamma$ ))  $R$  ( $right$ , modalise-fm  $C$ )
    by auto
  from  $left$  have  $left$ -deriv: mlderives ((MWorld  $w$ , modalise-fm  $A$ ) #
    map (modalise-lfm  $w$ )  $\Gamma$ )  $R$  ( $left$ , modalise-fm  $C$ )
    by (simp add: modalise-lfm-def)

```

```

from right have right-deriv: mlderes ((MWorld w, modalise-fm B) #
  map (modalise-lfm w)  $\Gamma$ ) R (right, modalise-fm C)
by (simp add: modalise-lfm-def)
have mlderes (map (modalise-lfm w)  $\Gamma$ ) R
  (MCases d (MWorld w) left (MWorld w) right, modalise-fm C)
using d left right left-deriv right-deriv by (auto intro: mlderes.MDisE)
then show ?case
by (intro exI[of - MCases d (MWorld w) left (MWorld w) right])
  (simp add: d left right)

```

qed

**theorem** *modal-labels-sound*:

```

assumes mlderes  $\Gamma$  R x
and  $\forall (l, A) \in \text{set } \Gamma. \text{modal-kripke-sat } V \text{ } R \text{ (world-of } l) \text{ } A$ 
shows modal-kripke-sat V R (world-of (fst x)) (snd x)
using assms
proof (induction arbitrary: V rule: mlderes.induct)
case (MAssm x  $\Gamma$  R)
then show ?case
by (cases x) auto
next
case (MBotE  $\Gamma$  R l A)
then show ?case
by auto
next
case (MImpI l m A  $\Gamma$  R B)
then show ?case
by auto
next
case (MImpE l m  $\Gamma$  R A B)
then show ?case
by auto
next
case (MConI l m  $\Gamma$  R A B)
then show ?case
by auto
next
case (MConE1  $\Gamma$  R l A B)
then show ?case
by auto
next
case (MConE2  $\Gamma$  R l A B)
then show ?case
by auto
next
case (MDisI1  $\Gamma$  R l A B)
then show ?case
by auto
next

```

```

    case (MDisI2  $\Gamma$  R l B A)
  then show ?case
    by auto
next
  case (MDisE l m n p q  $\Gamma$  R A B C)
  then show ?case
    by auto
next
  case (MBoxI l R  $\Gamma$  A)
  then show ?case
    by auto
next
  case (MBoxE  $\Gamma$  R l A v)
  then show ?case
    by auto
qed
end

```

```

theory Modal-K-Schema
  imports Modal-Labels-Example
begin

```

The modal instance derives the normal modal K axiom schema directly in the labelled calculus. The proof is uniform in the formulae, the accessibility relation, and the chosen world; necessitation is available only for formulae that are derivable from the empty context at every world and under every accessibility relation.

**theorem** *modal-K-schema:*

```

  fixes A B :: 'a mfm
  and R :: nat  $\Rightarrow$  nat  $\Rightarrow$  bool
  and w :: nat

```

**shows**

```

   $\exists$  l. world-of l = w  $\wedge$ 
    mlderives [] R (l, MImp (Box (MImp A B)) (MImp (Box A) (Box B)))

```

**proof** –

```

  let ?imp-assm = MWorld w
  let ?box-assm = MWorld w
  let ?box-label = MBoxI (MWorld w)
  let ?inner-label = MLam ?box-assm ?box-label
  let ?outer-label = MLam ?imp-assm ?inner-label
  let ?K = MImp (Box (MImp A B)) (MImp (Box A) (Box B))
  let ? $\Gamma$  = [(?box-assm, Box A), (?imp-assm, Box (MImp A B))]

```

```

  have box-deriv: mlderives ? $\Gamma$  R (?box-label, Box B)

```

**proof** (rule mlderives.MBoxI)

```

  fix v

```

```

  assume accessible: R (world-of (MWorld w)) v

```

```

let ?elim-label = MBoxE (MWorld v)
have imp-asm-deriv: mlderesives ? $\Gamma$  R (?imp-asm, Box (MImp A B))
  by (rule mlderesives.MAssm) simp
have imp-at-v: mlderesives ? $\Gamma$  R (?elim-label, MImp A B)
proof (rule mlderesives.MBoxE)
  show mlderesives ? $\Gamma$  R (?imp-asm, Box (MImp A B))
    by (rule imp-asm-deriv)
  show R (world-of ?imp-asm) v
    using accessible by simp
qed
have box-asm-deriv: mlderesives ? $\Gamma$  R (?box-asm, Box A)
  by (rule mlderesives.MAssm) simp
have A-at-v: mlderesives ? $\Gamma$  R (?elim-label, A)
proof (rule mlderesives.MBoxE)
  show mlderesives ? $\Gamma$  R (?box-asm, Box A)
    by (rule box-asm-deriv)
  show R (world-of ?box-asm) v
    using accessible by simp
qed
have B-at-v: mlderesives ? $\Gamma$  R (MApp ?elim-label ?elim-label, B)
proof (rule mlderesives.MImpE)
  show world-of ?elim-label = world-of ?elim-label
    by simp
  show mlderesives ? $\Gamma$  R (?elim-label, MImp A B)
    by (rule imp-at-v)
  show mlderesives ? $\Gamma$  R (?elim-label, A)
    by (rule A-at-v)
qed
show  $\exists m. \text{world-of } m = v \wedge \text{mlderesives } ?\Gamma R (m, B)$ 
  using B-at-v by (intro exI[of - MApp ?elim-label ?elim-label]) simp
qed

have inner-deriv:
  mlderesives [(?imp-asm, Box (MImp A B))] R
    (?inner-label, MImp (Box A) (Box B))
proof (rule mlderesives.MImpI)
  show world-of ?box-asm = world-of ?box-label
    by simp
  show mlderesives ((?box-asm, Box A) # [(?imp-asm, Box (MImp A B))]) R
    (?box-label, Box B)
    using box-deriv by simp
qed

have outer-deriv: mlderesives [] R (?outer-label, ?K)
proof (rule mlderesives.MImpI)
  show world-of ?imp-asm = world-of ?inner-label
    by simp
  show mlderesives ((?imp-asm, Box (MImp A B)) # []) R
    (?inner-label, MImp (Box A) (Box B))

```

```

    using inner-deriv by simp
  qed

  show ?thesis
    using outer-deriv by (intro exI[of - ?outer-label]) simp
  qed

theorem modal-necessitation:
  fixes A :: 'a mfm
  and R :: nat ⇒ nat ⇒ bool
  and w :: nat
  assumes provable-everywhere:
    ∧v R'. ∃l. world-of l = v ∧ mlderesives [] R' (l, A)
  shows
    ∃l. world-of l = w ∧ mlderesives [] R (l, Box A)
proof -
  have box-deriv: mlderesives [] R (MBoxI (MWorld w), Box A)
  proof (rule mlderesives.MBoxI)
    fix v
    assume R (world-of (MWorld w)) v
    from provable-everywhere[of v R]
    show ∃m. world-of m = v ∧ mlderesives [] R (m, A) .
  qed
  show ?thesis
    using box-deriv by (intro exI[of - MBoxI (MWorld w)]) simp
  qed

corollary modal-K-kripke-valid:
  fixes A B :: 'a mfm
  shows ∀ V R w. modal-kripke-sat V R w
    (MImp (Box (MImp A B)) (MImp (Box A) (Box B)))
proof (intro allI)
  fix V R w
  let ?K = MImp (Box (MImp A B)) (MImp (Box A) (Box B))
  obtain l where l-world: world-of l = w
  and deriv: mlderesives [] R (l, ?K)
  using modal-K-schema[where A=A and B=B and R=R and w=w] by blast
  have modal-kripke-sat V R (world-of (fst (l, ?K))) (snd (l, ?K))
  proof (rule modal-labels-sound)
    show mlderesives [] R (l, ?K)
    by (rule deriv)
    show ∀(l, A) ∈ set []. modal-kripke-sat V R (world-of l) A
    by simp
  qed
  with l-world show modal-kripke-sat V R w ?K
  by simp
  qed

corollary modal-K-example:

```

```

shows  $\exists l. \text{world-of } l = 0 \wedge \text{mderives } [] R$ 
      ( $l, \text{MImp } (\text{Box } (\text{MImp } (\text{MAtom } (0::\text{nat})) (\text{MAtom } 1)))$ 
        ( $\text{MImp } (\text{Box } (\text{MAtom } 0)) (\text{Box } (\text{MAtom } 1)))$ )
using modal-K-schema[
  where  $A = \text{MAtom } (0::\text{nat})$  and  $B = \text{MAtom } (1::\text{nat})$  and  $R = R$  and  $w = 0$ ]
by simp

```

**end**

**theory** *Examples*

**imports** *Provenance-Labels Modal-Labels-Example*

**begin**

This theory collects small derivations that exercise the labelled framework: provenance-labelled implicational combinators, labelled modus ponens, and a possible-world derivation of the modal K axiom.

**lemma** *K-axiom-labelled:*

*provenance.lderives* [] ( $\{\}, \text{Imp } A (\text{Imp } B A)$ )

**proof** –

**have** *a-from-ab:* *provenance.lderives* [ $\{\!1\}, B$ ], ( $\{\!0\}, A$ )] ( $\{\!0\}, A$ )

**by** (*rule provenance.LAssm*) *simp*

**have** *b-imp-a:* *provenance.lderives* [ $\{\!0\}, A$ ]] ( $\{\!0\}, \text{Imp } B A$ )

**proof** –

**have** *provenance.lderives* [ $\{\!0\}, A$ ]] ( $(\lambda S T. T - S) \{\!1\} \{\!0\}, \text{Imp } B A$ )

**using** *a-from-ab* **by** (*rule provenance.LImpI*)

**then show** *?thesis*

**by** *auto*

**qed**

**have** *provenance.lderives* [] ( $(\lambda S T. T - S) \{\!0\} \{\!0\}, \text{Imp } A (\text{Imp } B A)$ )

**using** *b-imp-a* **by** (*rule provenance.LImpI*)

**then show** *?thesis*

**by** *auto*

**qed**

**lemma** *S-axiom-labelled:*

*provenance.lderives* []

( $\{\}, \text{Imp } (\text{Imp } A (\text{Imp } B C)) (\text{Imp } (\text{Imp } A B) (\text{Imp } A C))$ )

**proof** –

**let**  $?F = \text{Imp } A (\text{Imp } B C)$

**let**  $?G = \text{Imp } A B$

**let**  $?ctx = [(\{\!2\}, A), (\{\!1\}, ?G), (\{\!0\}, ?F)]$

**have** *f:* *provenance.lderives* *?ctx* ( $\{\!0\}, ?F$ )

**by** (*rule provenance.LAssm*) *simp*

**have** *g:* *provenance.lderives* *?ctx* ( $\{\!1\}, ?G$ )

**by** (*rule provenance.LAssm*) *simp*

**have** *a:* *provenance.lderives* *?ctx* ( $\{\!2\}, A$ )

**by** (*rule provenance.LAssm*) *simp*

**have** *f-applied:* *provenance.lderives* *?ctx* ( $\{\!0, 2\}, \text{Imp } B C$ )

```

proof –
  have raw: provenance.lderives ?ctx (( $\lambda S T. S \cup T$ ) {0} {2}, Imp B C)
    using f a by (rule provenance.LImpE)
  have (( $\lambda S T. S \cup T$ ) ({0}::prov) ({2}::prov)) = {0, 2}
    by auto
  with raw show ?thesis
    by simp
qed
have b: provenance.lderives ?ctx ({1, 2}, B)
proof –
  have raw: provenance.lderives ?ctx (( $\lambda S T. S \cup T$ ) {1} {2}, B)
    using g a by (rule provenance.LImpE)
  have (( $\lambda S T. S \cup T$ ) ({1}::prov) ({2}::prov)) = {1, 2}
    by auto
  with raw show ?thesis
    by simp
qed
have c: provenance.lderives ?ctx ({0, 1, 2}, C)
proof –
  have raw: provenance.lderives ?ctx (( $\lambda S T. S \cup T$ ) {0, 2} {1, 2}, C)
    using f-applied b by (rule provenance.LImpE)
  have (( $\lambda S T. S \cup T$ ) ({0, 2}::prov) ({1, 2}::prov)) = {0, 1, 2}
    by auto
  with raw show ?thesis
    by simp
qed
have imp-a-c: provenance.lderives [({1}, ?G), ({0}, ?F)] ({0, 1}, Imp A C)
proof –
  have raw: provenance.lderives [({1}, ?G), ({0}, ?F)]
    (( $\lambda S T. T - S$ ) {2} {0, 1, 2}, Imp A C)
    using c by (rule provenance.LImpI)
  have (( $\lambda S T. T - S$ ) ({2}::prov) ({0, 1, 2}::prov)) = {0, 1}
    by auto
  with raw show ?thesis
    by simp
qed
have imp-g-a-c: provenance.lderives [({0}, ?F)] ({0}, Imp ?G (Imp A C))
proof –
  have raw: provenance.lderives [({0}, ?F)]
    (( $\lambda S T. T - S$ ) {1} {0, 1}, Imp ?G (Imp A C))
    using imp-a-c by (rule provenance.LImpI)
  have (( $\lambda S T. T - S$ ) ({1}::prov) ({0, 1}::prov)) = {0}
    by auto
  with raw show ?thesis
    by simp
qed
have raw: provenance.lderives []
  (( $\lambda S T. T - S$ ) {0} {0},
   Imp ?F (Imp ?G (Imp A C)))

```

**using** *imp-g-a-c* **by** (*rule provenance.LImpI*)  
**have**  $((\lambda S T. T - S) (\{0\}::\text{prov}) (\{0\}::\text{prov})) = \{\}$   
**by** *auto*  
**with raw show** *?thesis*  
**by** *simp*  
**qed**

**lemma** *modus-ponens-labelled*:  
*provenance.lderives*  $[(\{0\}, \text{Imp } A B), (\{1\}, A)] (\{0, 1\}, B)$   
**proof** –  
**have** *imp*: *provenance.lderives*  $[(\{0\}, \text{Imp } A B), (\{1\}, A)] (\{0\}, \text{Imp } A B)$   
**by** (*rule provenance.LAssm*) *simp*  
**have** *arg*: *provenance.lderives*  $[(\{0\}, \text{Imp } A B), (\{1\}, A)] (\{1\}, A)$   
**by** (*rule provenance.LAssm*) *simp*  
**have** *raw*: *provenance.lderives*  $[(\{0\}, \text{Imp } A B), (\{1\}, A)]$   
 $((\lambda S T. S \cup T) \{0\} \{1\}, B)$   
**using** *imp arg* **by** (*rule provenance.LImpE*)  
**have**  $((\lambda S T. S \cup T) (\{0\}::\text{prov}) (\{1\}::\text{prov})) = \{0, 1\}$   
**by** *auto*  
**with raw show** *?thesis*  
**by** *simp*  
**qed**

**lemma** *modal-K-axiom*:  
*mlderes*  $\square R$   
 $(MLam (MWorld w) (MLam (MWorld w) (MBoxI (MWorld w))),$   
 $MImp (Box (MImp A B)) (MImp (Box A) (Box B)))$   
**proof** (*rule mlderes.MImpI*)  
**show** *world-of*  $(MWorld w) = \text{world-of } (MLam (MWorld w) (MBoxI (MWorld w)))$   
**by** *simp*  
**show** *mlderes*  $[(MWorld w, Box (MImp A B))] R$   
 $(MLam (MWorld w) (MBoxI (MWorld w)), MImp (Box A) (Box B))$   
**proof** (*rule mlderes.MImpI*)  
**show** *world-of*  $(MWorld w) = \text{world-of } (MBoxI (MWorld w))$   
**by** *simp*  
**show** *mlderes*  $[(MWorld w, Box A), (MWorld w, Box (MImp A B))] R$   
 $(MBoxI (MWorld w), Box B)$   
**proof** (*rule mlderes.MBoxI*)  
**fix** *v*  
**assume** *access*:  $R (\text{world-of } (MWorld w)) v$   
**have** *imp*: *mlderes*  $[(MWorld w, Box A), (MWorld w, Box (MImp A B))] R$   
 $(MBoxE (MWorld v), MImp A B)$   
**by** (*meson MAssm MBoxE access list.set-intros(1,2)*)  
**have** *arg*: *mlderes*  $[(MWorld w, Box A), (MWorld w, Box (MImp A B))] R$   
 $(MBoxE (MWorld v), A)$   
**by** (*meson MAssm MBoxE access list.set-intros(1)*)  
**have** *result*: *mlderes*  $[(MWorld w, Box A), (MWorld w, Box (MImp A B))] R$   
*R*

```

      (MApp (MBoxE (MWorld v)) (MBoxE (MWorld v)), B)
    by (rule mlderives.MImpE[OF - imp arg]) simp
  show  $\exists m. \text{world-of } m = v \wedge$ 
      mlderives [(MWorld w, Box A), (MWorld w, Box (MImp A B))] R (m, B)
    using result
  by (intro exI[of - MApp (MBoxE (MWorld v)) (MBoxE (MWorld v))]) simp
qed
qed
qed
end

```

## References

- [1] D. Basin, S. Matthews, and L. Viganò. Labelled propositional modal logics: theory and practice. *Journal of Logic and Computation*, 7(6):685–717, 1997.
- [2] D. M. Gabbay. *Labelled Deductive Systems, Volume 1*. Oxford University Press, 1996.
- [3] A. K. Simpson. *The Proof Theory and Semantics of Intuitionistic Modal Logic*. PhD thesis, University of Edinburgh, 1994.
- [4] L. Viganò. *Labelled Non-Classical Logics*. Kluwer Academic Publishers, 2000.