Intuitionistic Linear Logic

Filip Smola

November 28, 2024

Contents

1 Intuitionistic Linear Logic

theory *ILL*
  imports
    Main
    HOL−Combinatorics.Permutations
begin

Note that in this theory we often use procedural proofs rather than structured ones. We find these to be more informative about how the basic rules of the logic are used when compared to collecting all the rules in one call of an automated method.

1.1 Deep Embedding of Propositions

We formalise ILL propositions as a datatype, parameterised by the type of propositional variables. The propositions are:

- Propositional variables
- Times of two terms, with unit **1**
- With of two terms, with unit $\top$
- Plus of two terms, with unit **0**
- Linear implication, with no unit
- Exponential of a term

datatype *'a ill-prop* =
  Prop 'a
| *Times 'a ill-prop 'a ill-prop* (**infixr** $\otimes$ *90*) | *One* (**1**)
| *With 'a ill-prop 'a ill-prop* (**infixr** & *90*) | *Top* ($\top$)
| *Plus 'a ill-prop 'a ill-prop* (**infixr** $\oplus$ *90*) | *Zero* (**0**)
| *LImp 'a ill-prop 'a ill-prop* (**infixr** $\triangleright$ *90*)
  — Note that Isabelle font does not include $\multimap$, so we use $\triangleright$ instead
| *Exp 'a ill-prop* (! *1000*)

1.2 Shallow Embedding of Deductions

See Bierman [1] or Kalvala and de Paiva [2] for an overview of valid sequents in ILL.

We first formalise ILL deductions as a relation between a list of propositions (anteceents) and a single proposition (consequent). This constitutes a shallow embedding of deductions (with a deep embedding to follow).

In using a list, as opposed to a multiset, we make the exchange rule explicit. Furthermore, we take as primitive a rule exchanging two propositions and later derive both the corresponding rule for lists of propositions as well as for multisets.

The specific formulation of rules we use here includes lists in more positions than is traditionally done when presenting ILL. This is inspired by the recommendations of Kalvala and de Paiva, intended to improve pattern matching and automation.

inductive *sequent* :: *'a ill-prop list* $\Rightarrow$ *'a ill-prop* $\Rightarrow$ *bool* (**infix** $\vdash$ *60*)
 where
 identity: $[a] \vdash a$
| *exchange*: $[\![G \mathbin{@} [a] \mathbin{@} [b] \mathbin{@} D \vdash c]\!] \Longrightarrow G \mathbin{@} [b] \mathbin{@} [a] \mathbin{@} D \vdash c$
| *cut*: $[\![G \vdash b; D \mathbin{@} [b] \mathbin{@} E \vdash c]\!] \Longrightarrow D \mathbin{@} G \mathbin{@} E \vdash c$
| *timesL*: $G \mathbin{@} [a] \mathbin{@} [b] \mathbin{@} D \vdash c \Longrightarrow G \mathbin{@} [a \otimes b] \mathbin{@} D \vdash c$

| *timesR*: $[\![G \vdash a;\ D \vdash b]\!] \Longrightarrow G\ @\ D \vdash a \otimes b$
| *oneL*: $G\ @\ D \vdash c \Longrightarrow G\ @\ [\mathbf{1}]\ @\ D \vdash c$
| *oneR*: $[] \vdash \mathbf{1}$
| *limpL*: $[\![G \vdash a;\ D\ @\ [b]\ @\ E \vdash c]\!] \Longrightarrow G\ @\ D\ @\ [a \rhd b]\ @\ E \vdash c$
| *limpR*: $G\ @\ [a]\ @\ D \vdash b \Longrightarrow G\ @\ D \vdash a \rhd b$
| *withL1*: $G\ @\ [a]\ @\ D \vdash c \Longrightarrow G\ @\ [a\ \&\ b]\ @\ D \vdash c$
| *withL2*: $G\ @\ [b]\ @\ D \vdash c \Longrightarrow G\ @\ [a\ \&\ b]\ @\ D \vdash c$
| *withR*: $[\![G \vdash a;\ G \vdash b]\!] \Longrightarrow G \vdash a\ \&\ b$
| *topR*: $G \vdash \top$
| *plusL*: $[\![G\ @\ [a]\ @\ D \vdash c;\ G\ @\ [b]\ @\ D \vdash c]\!] \Longrightarrow G\ @\ [a \oplus b]\ @\ D \vdash c$
| *plusR1*: $G \vdash a \Longrightarrow G \vdash a \oplus b$
| *plusR2*: $G \vdash b \Longrightarrow G \vdash a \oplus b$
| *zeroL*: $G\ @\ [\mathbf{0}]\ @\ D \vdash c$
| *weaken*: $G\ @\ D \vdash b \Longrightarrow G\ @\ [!a]\ @\ D \vdash b$
| *contract*: $G\ @\ [!a]\ @\ [!a]\ @\ D \vdash b \Longrightarrow G\ @\ [!a]\ @\ D \vdash b$
| *derelict*: $G\ @\ [a]\ @\ D \vdash b \Longrightarrow G\ @\ [!a]\ @\ D \vdash b$
| *promote*: $map\ Exp\ G \vdash a \Longrightarrow map\ Exp\ G \vdash\ !a$

lemmas [*simp*] = *sequent.identity*

1.3 Proposition Equivalence

Two propositions are equivalent when each can be derived from the other

definition *ill-eq* :: $'a$ *ill-prop* $\Rightarrow$ $'a$ *ill-prop* $\Rightarrow$ *bool* (**infix** $\dashv\vdash$ *60*)
where $a \dashv\vdash b = ([a] \vdash b \wedge [b] \vdash a)$

We show that this is an equivalence relation

lemma *ill-eq-refl* [*simp*]:
$a \dashv\vdash a$
⟨*proof*⟩

lemma *ill-eq-sym* [*sym*]:
$a \dashv\vdash b \Longrightarrow b \dashv\vdash a$
⟨*proof*⟩

lemma *ill-eq-tran* [*trans*]:
$[\![a \dashv\vdash b;\ b \dashv\vdash c]\!] \Longrightarrow a \dashv\vdash c$
⟨*proof*⟩

lemma *equivp ill-eq*
⟨*proof*⟩

lemma *ill-eqI* [*intro*]:
$[a] \vdash b \Longrightarrow [b] \vdash a \Longrightarrow a \dashv\vdash b$
⟨*proof*⟩

lemma *ill-eqE* [*elim*]:
$a \dashv\vdash b \Longrightarrow ([a] \vdash b \Longrightarrow [b] \vdash a \Longrightarrow R) \Longrightarrow R$
⟨*proof*⟩

lemma *ill-eq-lr*: $a \dashv\vdash b \Longrightarrow [a] \vdash b$
and *ill-eq-rl*: $a \dashv\vdash b \Longrightarrow [b] \vdash a$
$\langle proof \rangle$

1.4 Useful Rules

We can derive a number of useful rules from the defining ones, especially their specific instantiations.

Particularly useful is an instantiation of the Cut rule that makes it transitive, allowing us to use equational reasoning (**also** and **finally**) to build derivations using single propositions

lemma *simple-cut* [*trans*]:
$[\![G \vdash b;\ [b] \vdash c]\!] \Longrightarrow G \vdash c$
$\langle proof \rangle$

lemma
shows *sequent-Nil-left*: $[]\ @\ G \vdash c \Longrightarrow G \vdash c$
and *sequent-Nil-right*: $G\ @\ [] \vdash c \Longrightarrow G \vdash c$
$\langle proof \rangle$

lemma *simple-exchange*:
$[\![[a,\ b] \vdash c]\!] \Longrightarrow [b,\ a] \vdash c$
$\langle proof \rangle$

lemma *simple-timesL*:
$[\![[a]\ @\ [b] \vdash c]\!] \Longrightarrow [a \otimes b] \vdash c$
$\langle proof \rangle$

lemma *simple-withL1*: $[\![[a] \vdash c]\!] \Longrightarrow [a\ \&\ b] \vdash c$
and *simple-withL2*: $[\![[b] \vdash c]\!] \Longrightarrow [a\ \&\ b] \vdash c$
$\langle proof \rangle$

lemma *simple-plusL*:
$[\![[a] \vdash c;\ [b] \vdash c]\!] \Longrightarrow [a \oplus b] \vdash c$
$\langle proof \rangle$

lemma *simple-weaken*:
$[!a] \vdash \mathbf{1}$
$\langle proof \rangle$

lemma *simple-derelict*:
$[\![[a] \vdash b]\!] \Longrightarrow [!a] \vdash b$
$\langle proof \rangle$

lemmas *simple-promote* = *promote*[*of* [-], *unfolded list.map*]

lemma *promote-and-derelict*:

assumes $G \vdash c$
shows *map Exp* $G \vdash\ !c$
⟨*proof*⟩

lemmas *dereliction* = *simple-derelict*[*OF identity*]

lemma *simple-contract*:
$[\![[!a]\ @\ [!a] \vdash b]\!] \Longrightarrow [!a] \vdash b$
⟨*proof*⟩

lemma *duplicate*:
$[!a] \vdash\ !a \otimes\ !a$
⟨*proof*⟩

lemma *unary-promote*:
$[\![[!g] \vdash a]\!] \Longrightarrow [!g] \vdash\ !a$
⟨*proof*⟩

lemma *tensor*:
$[\![[a] \vdash b;\ [c] \vdash d]\!] \Longrightarrow [a \otimes c] \vdash b \otimes d$
⟨*proof*⟩

lemma *ill-eq-tensor*:
$a \dashv\vdash b \Longrightarrow x \dashv\vdash y \Longrightarrow a \otimes x \dashv\vdash b \otimes y$
⟨*proof*⟩

lemma *times-assoc*:
$[(a \otimes b) \otimes c] \vdash a \otimes (b \otimes c)$
⟨*proof*⟩

lemma *times-assoc'*:
$[a \otimes (b \otimes c)] \vdash (a \otimes b) \otimes c$
⟨*proof*⟩

lemma *simple-limpR*:
$[a] \vdash b \Longrightarrow [\mathbf{1}] \vdash a \rhd b$
⟨*proof*⟩

lemma *simple-limpR-exp*:
$[a] \vdash b \Longrightarrow [\mathbf{1}] \vdash\ !(a \rhd b)$
⟨*proof*⟩

lemma *limp-eval*:
$[a \otimes a \rhd b] \vdash b$
⟨*proof*⟩

lemma *timesR-intro*:
$[\![G \vdash a;\ D \vdash b;\ G\ @\ D = X]\!] \Longrightarrow X \vdash a \otimes b$
⟨*proof*⟩

lemma *explimp-eval*:
$[a \otimes !(a \rhd b)] \vdash b \otimes !(a \rhd b)$
$\langle proof \rangle$

lemma *plus-progress*:
$[\![[a] \vdash b; [c] \vdash d]\!] \Longrightarrow [a \oplus c] \vdash b \oplus d$
$\langle proof \rangle$

The following set of rules are based on Proposition 1 of Bierman [1]. Where there is a direct correspondence, we include a comment indicating the specific item in the proposition.

lemma *swap*: — Item 1
$[a \otimes b] \vdash b \otimes a$
$\langle proof \rangle$

lemma *unit*: — Item 2
$[a \otimes \mathbf{1}] \vdash a$
$\langle proof \rangle$

lemma *unit'*: — Item 2
$[a] \vdash a \otimes \mathbf{1}$
$\langle proof \rangle$

lemma *with-swap*: — Item 3
$[a \ \& \ b] \vdash b \ \& \ a$
$\langle proof \rangle$

lemma *with-top*: — Item 4
$a \dashv\vdash a \ \& \ \top$
$\langle proof \rangle$

lemma *plus-swap*: — Item 5
$[a \oplus b] \vdash b \oplus a$
$\langle proof \rangle$

lemma *plus-zero*: — Item 6
$a \dashv\vdash a \oplus \mathbf{0}$
$\langle proof \rangle$

lemma *with-distrib*: — Item 7
$[a \otimes (b \ \& \ c)] \vdash (a \otimes b) \ \& \ (a \otimes c)$
$\langle proof \rangle$

lemma *plus-distrib*: — Item 8
$[a \otimes (b \oplus c)] \vdash (a \otimes b) \oplus (a \otimes c)$
$\langle proof \rangle$

lemma *plus-distrib'*: — Item 9

$[(a \otimes b) \oplus (a \otimes c)] \vdash a \otimes (b \oplus c)$
⟨*proof*⟩

lemma *times-exp*: — Item 10
$[!a \otimes !b] \vdash !(a \otimes b)$
⟨*proof*⟩

lemma *one-exp*: — Item 10
$\mathbf{1} \dashv\vdash !(\mathbf{1})$
⟨*proof*⟩

lemma — Item 11
$[!a] \vdash \mathbf{1} \mathbin{\&} a \mathbin{\&} (!a \otimes !a)$
⟨*proof*⟩

lemma — Item 12
$!a \otimes !b \dashv\vdash !(a \mathbin{\&} b)$
⟨*proof*⟩

lemma — Item 13
$\mathbf{1} \dashv\vdash !(\top)$
⟨*proof*⟩

1.5 Compacting Lists of Propositions

Compacting transforms a list of propositions into a single proposition using the ($\otimes$) operator, taking care to not expand the size when given a list with only one element. This operation allows us to link the meta-level antecedent concatenation with the object-level ($\otimes$) operator, turning a list of antecedents into a single proposition with the same power in proofs.

function *compact* :: *'a ill-prop list* $\Rightarrow$ *'a ill-prop*
where
$xs \neq [] \Longrightarrow compact\ (x \# xs) = x \otimes compact\ xs$
$|\ xs = [] \Longrightarrow compact\ (x \# xs) = x$
$|\ compact\ [] = \mathbf{1}$
⟨*proof*⟩
termination ⟨*proof*⟩

For code generation we use an if statement

lemma *compact-code* [*code*]:
$compact\ [] = \mathbf{1}$
$compact\ (x \# xs) = (if\ xs = []\ then\ x\ else\ x \otimes compact\ xs)$
⟨*proof*⟩

Two lists of propositions that compact to the same result must be equal if they do not include any ($\otimes$) or $\mathbf{1}$ elements. We show first that they must be equally long and then that they must be equal.

lemma *compact-eq-length*:

assumes $\bigwedge a.\ a \in set\ xs \Longrightarrow a \neq \mathbf{1}$
and $\bigwedge a.\ a \in set\ ys \Longrightarrow a \neq \mathbf{1}$
and $\bigwedge a\ u\ v.\ a \in set\ xs \Longrightarrow a \neq u \otimes v$
and $\bigwedge a\ u\ v.\ a \in set\ ys \Longrightarrow a \neq u \otimes v$
and $compact\ xs = compact\ ys$
shows $length\ xs = length\ ys$
$\langle proof \rangle$

lemma *compact-eq*:
assumes $\bigwedge a.\ a \in set\ xs \Longrightarrow a \neq \mathbf{1}$
and $\bigwedge a.\ a \in set\ ys \Longrightarrow a \neq \mathbf{1}$
and $\bigwedge a\ u\ v.\ a \in set\ xs \Longrightarrow a \neq u \otimes v$
and $\bigwedge a\ u\ v.\ a \in set\ ys \Longrightarrow a \neq u \otimes v$
and $compact\ xs = compact\ ys$
shows $xs = ys$
$\langle proof \rangle$

Compacting to **1** means the list of propositions was either empty or just that

lemma *compact-eq-oneE*:
assumes $compact\ xs = \mathbf{1}$
obtains $xs = []\ |\ xs = [\mathbf{1}]$
$\langle proof \rangle$

Compacting to $(\otimes)$ means the list of propositions was either just that or started with the left-hand proposition and the rest compacts to the right-hand proposition

lemma *compact-eq-timesE*:
assumes $compact\ xs = x \otimes y$
obtains $xs = [x \otimes y]\ |\ ys$ **where** $xs = x\ \#\ ys$ **and** $compact\ ys = y$
$\langle proof \rangle$

Compacting to anything but **1** or $(\otimes)$ means the list was just that

lemma *compact-eq-otherD*:
assumes $compact\ xs = a$
and $\bigwedge x\ y.\ a \neq x \otimes y$
and $a \neq \mathbf{1}$
shows $xs = [a]$
$\langle proof \rangle$

For any list of propositions, we can derive its compacted form from it

lemma *identity-list*:
$G \vdash (compact\ G)$
$\langle proof \rangle$

For any valid sequent, we can compact any sublist of its antecedents without invalidating it

lemma *compact-split-antecedents*:

assumes $X @ G @ Y \vdash c$
shows $n \leq \mathit{length}\ G \Longrightarrow X @ \mathit{take}\ (\mathit{length}\ G - n)\ G @ [\mathit{compact}\ (\mathit{drop}\ (\mathit{length}\ G - n)\ G)] @ Y \vdash c$
$\langle \mathit{proof} \rangle$

More generally, compacting a sublist of antecedents does not affect sequent validity

lemma *compact-antecedents*:
$(X @ [\mathit{compact}\ G] @ Y \vdash c) = (X @ G @ Y \vdash c)$
$\langle \mathit{proof} \rangle$

Times with a single proposition can be absorbed into compacting up to proposition equivalence

lemma *times-equivalent-cons*:
$a \otimes \mathit{compact}\ b \dashv\vdash \mathit{compact}\ (a \# b)$
$\langle \mathit{proof} \rangle$

Times of compacted lists is equivalent to compacting the appended lists

lemma *times-equivalent-append*:
$\mathit{compact}\ a \otimes \mathit{compact}\ b \dashv\vdash \mathit{compact}\ (a @ b)$
$\langle \mathit{proof} \rangle$

Any number of single-antecedent sequents can be compacted with the rule $[\![[?a] \vdash ?b; [?c] \vdash ?d]\!] \Longrightarrow [?a \otimes ?c] \vdash ?b \otimes ?d$

lemma *compact-sequent*:
$\forall x \in \mathit{set}\ \mathit{xs}.\ [f\ x] \vdash g\ x \Longrightarrow [\mathit{compact}\ (\mathit{map}\ f\ \mathit{xs})] \vdash \mathit{compact}\ (\mathit{map}\ g\ \mathit{xs})$
$\langle \mathit{proof} \rangle$

Any number of equivalences can be compacted together

lemma *compact-equivalent*:
$\forall x \in \mathit{set}\ \mathit{xs}.\ f\ x \dashv\vdash g\ x \Longrightarrow \mathit{compact}\ (\mathit{map}\ f\ \mathit{xs}) \dashv\vdash \mathit{compact}\ (\mathit{map}\ g\ \mathit{xs})$
$\langle \mathit{proof} \rangle$

1.6 Multiset Exchange

Recall that our ($\vdash$) definition uses explicit single-proposition exchange. We now derive a rule for exchanging lists of propositions and then a rule that uses multisets to disregard the antecedent order entirely.

We can exchange lists of propositions by stepping through *compact*

lemma *exchange-list*:
$G @ A @ B @ D \vdash c \Longrightarrow G @ B @ A @ D \vdash c$
$\langle \mathit{proof} \rangle$

lemma *simple-exchange-list*:
$[\![A @ B \vdash c]\!] \Longrightarrow B @ A \vdash c$
$\langle \mathit{proof} \rangle$

By applying the list exchange rule multiple times, the lists do not need to be adjacent

lemma *exchange-separated*:
  $G\ @\ A\ @\ X\ @\ B\ @\ D \vdash c \Longrightarrow G\ @\ B\ @\ X\ @\ A\ @\ D \vdash c$
  ⟨*proof*⟩

Single transposition in the antecedents does not invalidate a sequent

lemma *exchange-transpose*:
  assumes $G \vdash c$
    and $a \in \{..<\mathit{length}\ G\}$
    and $b \in \{..<\mathit{length}\ G\}$
   shows $\mathit{permute\text{-}list}\ (\mathit{transpose}\ a\ b)\ G \vdash c$
⟨*proof*⟩

More generally, by transposition being involutive, a single antecedent transposition does not affect sequent validity

lemma *exchange-permute-eq*:
  assumes $a \in \{..<\mathit{length}\ G\}$
    and $b \in \{..<\mathit{length}\ G\}$
   shows $\mathit{permute\text{-}list}\ (\mathit{transpose}\ a\ b)\ G \vdash c = G \vdash c$
  ⟨*proof*⟩

Validity of a sequent is not affected by replacing any antecedent sublist with a list that represents the same multiset. This is because lists representing equal multisets are connected by a permutation, which is a sequence of transpositions and as such does not affect validity.

lemma *exchange-mset*:
  $\mathit{mset}\ A = \mathit{mset}\ B \Longrightarrow G\ @\ A\ @\ D \vdash c = G\ @\ B\ @\ D \vdash c$
⟨*proof*⟩

1.7 Additional Lemmas

These rules are based on Figure 2 of Kalvala and de Paiva [2], labelled by them as "additional rules for proof search". We present them out of order because we use some in the proofs of the others, but annotate them with the original labels as comments.

lemma *ill-mp1*: — mp_1
  assumes $A\ @\ [b]\ @\ B\ @\ C \vdash c$
   shows $A\ @\ [a]\ @\ B\ @\ [a \rhd b]\ @\ C \vdash c$
⟨*proof*⟩

lemmas *simple-mp1* = *ill-mp1*[*of Nil - Nil Nil*, *simplified*, *OF identity*]

lemma — raa_1
  $G\ @\ [!b]\ @\ D\ @\ [!b \rhd \mathbf{0}]\ @\ E \vdash a$
  ⟨*proof*⟩

lemma *ill-mp2*: — mp_2
  assumes A @ $[b]$ @ B @ $C \vdash c$
    shows A @ $[a \rhd b]$ @ B @ $[a]$ @ $C \vdash c$
  ⟨*proof*⟩

lemmas *simple-mp2* = *ill-mp2*[*of Nil - Nil Nil*, *simplified*, *OF identity*]

lemma — raa_2
  G @ $[!b \rhd \mathbf{0}]$ @ D @ $[!b]$ @ $P \vdash A$
  ⟨*proof*⟩

lemma — $\otimes$-&
  assumes G @ $[(!a \rhd \mathbf{0})\ \&\ (!b \rhd \mathbf{0})]$ @ $D \vdash c$
    shows G @ $[!(!(a \oplus b) \rhd \mathbf{0})]$ @ $D \vdash c$
⟨*proof*⟩

lemma — &-*lemma*
  assumes G @ $[!a, !b]$ @ $D \vdash c$
    shows G @ $[!(a\ \&\ b)]$ @ $D \vdash c$
⟨*proof*⟩

lemma — $\multimap_L$-*lemma*
  assumes G @ $D \vdash a$
  shows G @ $[!(a \rhd b)]$ @ $D \vdash b$
  ⟨*proof*⟩

lemma — $\multimap_R$-*lemma*
  assumes $[a, !a]$ @ $G \vdash b$
  shows $G \vdash !a \rhd b$
  ⟨*proof*⟩

lemma — *a-not-a*
  assumes G @ $[!a \rhd \mathbf{0}]$ @ $D \vdash b$
  shows G @ $[!a \rhd (!a \rhd \mathbf{0})]$ @ $D \vdash b$
⟨*proof*⟩

end
theory *Proof*
  imports *ILL*
begin

1.8 Deep Embedding of Deductions

To directly manipulate ILL deductions themselves we deeply embed them as a datatype. This datatype has a constructor to represent each introduction rule of ($\vdash$), with the ILL propositions and further deductions those rules use as arguments. Additionally, it has a constructor to represent premises (sequents assumed to be valid) which allow us to represent contingent deductions.

The datatype is parameterised by two type variables:

- $'a$ represents the propositional variables for the contained ILL propositions, and
- $'l$ represents labels we associate with premises.

datatype ($'a$, $'l$) *ill-deduct* =
  Premise $'a$ *ill-prop list* $'a$ *ill-prop* $'l$
| *Identity* $'a$ *ill-prop*
| *Exchange* $'a$ *ill-prop list* $'a$ *ill-prop* $'a$ *ill-prop* $'a$ *ill-prop list* $'a$ *ill-prop*
  ($'a$, $'l$) *ill-deduct*
| *Cut* $'a$ *ill-prop list* $'a$ *ill-prop* $'a$ *ill-prop list* $'a$ *ill-prop list* $'a$ *ill-prop*
  ($'a$, $'l$) *ill-deduct* ($'a$, $'l$) *ill-deduct*
| *TimesL* $'a$ *ill-prop list* $'a$ *ill-prop* $'a$ *ill-prop* $'a$ *ill-prop list* $'a$ *ill-prop*
  ($'a$, $'l$) *ill-deduct*
| *TimesR* $'a$ *ill-prop list* $'a$ *ill-prop* $'a$ *ill-prop list* $'a$ *ill-prop* ($'a$, $'l$) *ill-deduct*
  ($'a$, $'l$) *ill-deduct*
| *OneL* $'a$ *ill-prop list* $'a$ *ill-prop list* $'a$ *ill-prop* ($'a$, $'l$) *ill-deduct*
| *OneR*
| *LimpL* $'a$ *ill-prop list* $'a$ *ill-prop* $'a$ *ill-prop list* $'a$ *ill-prop* $'a$ *ill-prop list*
  $'a$ *ill-prop* ($'a$, $'l$) *ill-deduct* ($'a$, $'l$) *ill-deduct*
| *LimpR* $'a$ *ill-prop list* $'a$ *ill-prop* $'a$ *ill-prop list* $'a$ *ill-prop* ($'a$, $'l$) *ill-deduct*
| *WithL1* $'a$ *ill-prop list* $'a$ *ill-prop* $'a$ *ill-prop* $'a$ *ill-prop list* $'a$ *ill-prop*
  ($'a$, $'l$) *ill-deduct*
| *WithL2* $'a$ *ill-prop list* $'a$ *ill-prop* $'a$ *ill-prop* $'a$ *ill-prop list* $'a$ *ill-prop*
  ($'a$, $'l$) *ill-deduct*
| *WithR* $'a$ *ill-prop list* $'a$ *ill-prop* $'a$ *ill-prop* ($'a$, $'l$) *ill-deduct* ($'a$, $'l$) *ill-deduct*
| *TopR* $'a$ *ill-prop list*
| *PlusL* $'a$ *ill-prop list* $'a$ *ill-prop* $'a$ *ill-prop* $'a$ *ill-prop list* $'a$ *ill-prop*
  ($'a$, $'l$) *ill-deduct* ($'a$, $'l$) *ill-deduct*
| *PlusR1* $'a$ *ill-prop list* $'a$ *ill-prop* $'a$ *ill-prop* ($'a$, $'l$) *ill-deduct*
| *PlusR2* $'a$ *ill-prop list* $'a$ *ill-prop* $'a$ *ill-prop* ($'a$, $'l$) *ill-deduct*
| *ZeroL* $'a$ *ill-prop list* $'a$ *ill-prop list* $'a$ *ill-prop*
| *Weaken* $'a$ *ill-prop list* $'a$ *ill-prop list* $'a$ *ill-prop* $'a$ *ill-prop* ($'a$, $'l$) *ill-deduct*
| *Contract* $'a$ *ill-prop list* $'a$ *ill-prop* $'a$ *ill-prop list* $'a$ *ill-prop* ($'a$, $'l$) *ill-deduct*
| *Derelict* $'a$ *ill-prop list* $'a$ *ill-prop* $'a$ *ill-prop list* $'a$ *ill-prop* ($'a$, $'l$) *ill-deduct*
| *Promote* $'a$ *ill-prop list* $'a$ *ill-prop* ($'a$, $'l$) *ill-deduct*

1.8.1 Semantics

With every deduction we associate the antecedents and consequent of its conclusion sequent

primrec *antecedents* :: ($'a$, $'l$) *ill-deduct* $\Rightarrow$ $'a$ *ill-prop list*
 where
  antecedents (*Premise* G c l) = G
| *antecedents* (*Identity* a) = $[a]$

| *antecedents* (*Exchange G a b D c P*) = *G* @ [*b*] @ [*a*] @ *D*
| *antecedents* (*Cut G b D E c P Q*) = *D* @ *G* @ *E*
| *antecedents* (*TimesL G a b D c P*) = *G* @ [$a \otimes b$] @ *D*
| *antecedents* (*TimesR G a D b P Q*) = *G* @ *D*
| *antecedents* (*OneL G D c P*) = *G* @ [**1**] @ *D*
| *antecedents* (*OneR*) = []
| *antecedents* (*LimpL G a D b E c P Q*) = *G* @ *D* @ [$a \rhd b$] @ *E*
| *antecedents* (*LimpR G a D b P*) = *G* @ *D*
| *antecedents* (*WithL1 G a b D c P*) = *G* @ [*a* & *b*] @ *D*
| *antecedents* (*WithL2 G a b D c P*) = *G* @ [*a* & *b*] @ *D*
| *antecedents* (*WithR G a b P Q*) = *G*
| *antecedents* (*TopR G*) = *G*
| *antecedents* (*PlusL G a b D c P Q*) = *G* @ [$a \oplus b$] @ *D*
| *antecedents* (*PlusR1 G a b P*) = *G*
| *antecedents* (*PlusR2 G a b P*) = *G*
| *antecedents* (*ZeroL G D c*) = *G* @ [**0**] @ *D*
| *antecedents* (*Weaken G D b a P*) = *G* @ [!*a*] @ *D*
| *antecedents* (*Contract G a D b P*) = *G* @ [!*a*] @ *D*
| *antecedents* (*Derelict G a D b P*) = *G* @ [!*a*] @ *D*
| *antecedents* (*Promote G a P*) = *map Exp G*

primrec *consequent* :: (*'a*, *'l*) *ill-deduct* ⇒ *'a ill-prop*
where
consequent (*Premise G c l*) = *c*
| *consequent* (*Identity a*) = *a*
| *consequent* (*Exchange G a b D c P*) = *c*
| *consequent* (*Cut G b D E c P Q*) = *c*
| *consequent* (*TimesL G a b D c P*) = *c*
| *consequent* (*TimesR G a D b P Q*) = $a \otimes b$
| *consequent* (*OneL G D c P*) = *c*
| *consequent* (*OneR*) = **1**
| *consequent* (*LimpL G a D b E c P Q*) = *c*
| *consequent* (*LimpR G a D b P*) = $a \rhd b$
| *consequent* (*WithL1 G a b D c P*) = *c*
| *consequent* (*WithL2 G a b D c P*) = *c*
| *consequent* (*WithR G a b P Q*) = *a* & *b*
| *consequent* (*TopR G*) = ⊤
| *consequent* (*PlusL G a b D c P Q*) = *c*
| *consequent* (*PlusR1 G a b P*) = $a \oplus b$
| *consequent* (*PlusR2 G a b P*) = $a \oplus b$
| *consequent* (*ZeroL G D c*) = *c*
| *consequent* (*Weaken G D b a P*) = *b*
| *consequent* (*Contract G a D b P*) = *b*
| *consequent* (*Derelict G a D b P*) = *b*
| *consequent* (*Promote G a P*) = !*a*

We define a sequent datatype for presenting deduction tree conclusions, deeply embedding (possibly invalid) sequents themselves.

Note: these are not used everywhere, separate antecedents and consequent tend to work better for proof automation. For instance, the full conclusion cannot be derived where only facts about antecedents are known.

datatype *'a ill-sequent* = *Sequent 'a ill-prop list 'a ill-prop*

Validity of deeply embedded sequents is defined by the shallow ($\vdash$) relation

primrec *ill-sequent-valid* :: *'a ill-sequent* ⇒ *bool*
  where *ill-sequent-valid* (*Sequent a c*) = $a \vdash c$

We set up a notation bundle to have infix $\vdash$ for stand for the sequent datatype and not the relation

bundle *deep-sequent*
begin
no-notation *sequent* (**infix** $\vdash$ *60*)
notation *Sequent* (**infix** $\vdash$ *60*)
end

context
  includes *deep-sequent*
begin

With deeply embedded sequents we can define the conclusion of every deduction

primrec *ill-conclusion* :: (*'a*, *'l*) *ill-deduct* ⇒ *'a ill-sequent*
  where
    ill-conclusion (*Premise G c l*) = $G \vdash c$
  | *ill-conclusion* (*Identity a*) = $[a] \vdash a$
  | *ill-conclusion* (*Exchange G a b D c P*) = G @ $[b]$ @ $[a]$ @ $D \vdash c$
  | *ill-conclusion* (*Cut G b D E c P Q*) = D @ G @ $E \vdash c$
  | *ill-conclusion* (*TimesL G a b D c P*) = G @ $[a \otimes b]$ @ $D \vdash c$
  | *ill-conclusion* (*TimesR G a D b P Q*) = G @ $D \vdash a \otimes b$
  | *ill-conclusion* (*OneL G D c P*) = G @ $[\mathbf{1}]$ @ $D \vdash c$
  | *ill-conclusion* (*OneR*) = $[] \vdash \mathbf{1}$
  | *ill-conclusion* (*LimpL G a D b E c P Q*) = G @ D @ $[a \rhd b]$ @ $E \vdash c$
  | *ill-conclusion* (*LimpR G a D b P*) = G @ $D \vdash a \rhd b$
  | *ill-conclusion* (*WithL1 G a b D c P*) = G @ $[a \,\&\, b]$ @ $D \vdash c$
  | *ill-conclusion* (*WithL2 G a b D c P*) = G @ $[a \,\&\, b]$ @ $D \vdash c$
  | *ill-conclusion* (*WithR G a b P Q*) = $G \vdash a \,\&\, b$
  | *ill-conclusion* (*TopR G*) = $G \vdash \top$
  | *ill-conclusion* (*PlusL G a b D c P Q*) = G @ $[a \oplus b]$ @ $D \vdash c$
  | *ill-conclusion* (*PlusR1 G a b P*) = $G \vdash a \oplus b$
  | *ill-conclusion* (*PlusR2 G a b P*) = $G \vdash a \oplus b$
  | *ill-conclusion* (*ZeroL G D c*) = G @ $[\mathbf{0}]$ @ $D \vdash c$
  | *ill-conclusion* (*Weaken G D b a P*) = G @ $[!a]$ @ $D \vdash b$
  | *ill-conclusion* (*Contract G a D b P*) = G @ $[!a]$ @ $D \vdash b$
  | *ill-conclusion* (*Derelict G a D b P*) = G @ $[!a]$ @ $D \vdash b$
  | *ill-conclusion* (*Promote G a P*) = *map Exp G* $\vdash !a$

This conclusion is the same as what *antecedents* and *consequent* express

lemma *ill-conclusionI* [*intro*!]:
assumes *antecedents* $P = G$
and *consequent* $P = c$
shows *ill-conclusion* $P = G \vdash c$
⟨*proof*⟩

lemma *ill-conclusionE* [*elim*!]:
assumes *ill-conclusion* $P = G \vdash c$
obtains *antecedents* $P = G$
and *consequent* $P = c$
⟨*proof*⟩

lemma *ill-conclusion-alt*:
$(\mathit{ill\text{-}conclusion}\ P = G \vdash c) = (\mathit{antecedents}\ P = G \wedge \mathit{consequent}\ P = c)$
⟨*proof*⟩

lemma *ill-conclusion-antecedents*: $\mathit{ill\text{-}conclusion}\ P = G \vdash c \Longrightarrow \mathit{antecedents}\ P = G$
and *ill-conclusion-consequent*: $\mathit{ill\text{-}conclusion}\ P = G \vdash c \Longrightarrow \mathit{consequent}\ P = c$
⟨*proof*⟩

Every deduction is well-formed if all deductions it relies on are well-formed and have the form required by the corresponding *sequent* rule.

primrec *ill-deduct-wf* :: $('a, 'l)$ *ill-deduct* $\Rightarrow$ *bool*
where
$\mathit{ill\text{-}deduct\text{-}wf}\ (\mathit{Premise}\ G\ c\ l) = \mathit{True}$
$|\ \mathit{ill\text{-}deduct\text{-}wf}\ (\mathit{Identity}\ a) = \mathit{True}$
$|\ \mathit{ill\text{-}deduct\text{-}wf}\ (\mathit{Exchange}\ G\ a\ b\ D\ c\ P) =$
$(\mathit{ill\text{-}deduct\text{-}wf}\ P \wedge \mathit{ill\text{-}conclusion}\ P = G\ @\ [a]\ @\ [b]\ @\ D \vdash c)$
$|\ \mathit{ill\text{-}deduct\text{-}wf}\ (\mathit{Cut}\ G\ b\ D\ E\ c\ P\ Q) =$
$(\ \mathit{ill\text{-}deduct\text{-}wf}\ P \wedge \mathit{ill\text{-}conclusion}\ P = G \vdash b\ \wedge$
$\mathit{ill\text{-}deduct\text{-}wf}\ Q \wedge \mathit{ill\text{-}conclusion}\ Q = D\ @\ [b]\ @\ E \vdash c)$
$|\ \mathit{ill\text{-}deduct\text{-}wf}\ (\mathit{TimesL}\ G\ a\ b\ D\ c\ P) =$
$(\mathit{ill\text{-}deduct\text{-}wf}\ P \wedge \mathit{ill\text{-}conclusion}\ P = G\ @\ [a]\ @\ [b]\ @\ D \vdash c)$
$|\ \mathit{ill\text{-}deduct\text{-}wf}\ (\mathit{TimesR}\ G\ a\ D\ b\ P\ Q) =$
$(\ \mathit{ill\text{-}deduct\text{-}wf}\ P \wedge \mathit{ill\text{-}conclusion}\ P = G \vdash a\ \wedge$
$\mathit{ill\text{-}deduct\text{-}wf}\ Q \wedge \mathit{ill\text{-}conclusion}\ Q = D \vdash b)$
$|\ \mathit{ill\text{-}deduct\text{-}wf}\ (\mathit{OneL}\ G\ D\ c\ P) =$
$(\mathit{ill\text{-}deduct\text{-}wf}\ P \wedge \mathit{ill\text{-}conclusion}\ P = G\ @\ D \vdash c)$
$|\ \mathit{ill\text{-}deduct\text{-}wf}\ (\mathit{OneR}) = \mathit{True}$
$|\ \mathit{ill\text{-}deduct\text{-}wf}\ (\mathit{LimpL}\ G\ a\ D\ b\ E\ c\ P\ Q) =$
$(\ \mathit{ill\text{-}deduct\text{-}wf}\ P \wedge \mathit{ill\text{-}conclusion}\ P = G \vdash a\ \wedge$
$\mathit{ill\text{-}deduct\text{-}wf}\ Q \wedge \mathit{ill\text{-}conclusion}\ Q = D\ @\ [b]\ @\ E \vdash c)$
$|\ \mathit{ill\text{-}deduct\text{-}wf}\ (\mathit{LimpR}\ G\ a\ D\ b\ P) =$
$(\mathit{ill\text{-}deduct\text{-}wf}\ P \wedge \mathit{ill\text{-}conclusion}\ P = G\ @\ [a]\ @\ D \vdash b)$
$|\ \mathit{ill\text{-}deduct\text{-}wf}\ (\mathit{WithL1}\ G\ a\ b\ D\ c\ P) =$
$(\mathit{ill\text{-}deduct\text{-}wf}\ P \wedge \mathit{ill\text{-}conclusion}\ P = G\ @\ [a]\ @\ D \vdash c)$
$|\ \mathit{ill\text{-}deduct\text{-}wf}\ (\mathit{WithL2}\ G\ a\ b\ D\ c\ P) =$
$(\mathit{ill\text{-}deduct\text{-}wf}\ P \wedge \mathit{ill\text{-}conclusion}\ P = G\ @\ [b]\ @\ D \vdash c)$

| *ill-deduct-wf* (*WithR* G a b P Q) =
  (*ill-deduct-wf* P ∧ *ill-conclusion* P = $G \vdash a$ ∧
   ill-deduct-wf Q ∧ *ill-conclusion* Q = $G \vdash b$)
| *ill-deduct-wf* (*TopR* G) = *True*
| *ill-deduct-wf* (*PlusL* G a b D c P Q) =
  (*ill-deduct-wf* P ∧ *ill-conclusion* P = G @ [a] @ $D \vdash c$ ∧
   ill-deduct-wf Q ∧ *ill-conclusion* Q = G @ [b] @ $D \vdash c$)
| *ill-deduct-wf* (*PlusR1* G a b P) =
  (*ill-deduct-wf* P ∧ *ill-conclusion* P = $G \vdash a$)
| *ill-deduct-wf* (*PlusR2* G a b P) =
  (*ill-deduct-wf* P ∧ *ill-conclusion* P = $G \vdash b$)
| *ill-deduct-wf* (*ZeroL* G D c) = *True*
| *ill-deduct-wf* (*Weaken* G D b a P) =
  (*ill-deduct-wf* P ∧ *ill-conclusion* P = G @ $D \vdash b$)
| *ill-deduct-wf* (*Contract* G a D b P) =
  (*ill-deduct-wf* P ∧ *ill-conclusion* P = G @ [!a] @ [!a] @ $D \vdash b$)
| *ill-deduct-wf* (*Derelict* G a D b P) =
  (*ill-deduct-wf* P ∧ *ill-conclusion* P = G @ [a] @ $D \vdash b$)
| *ill-deduct-wf* (*Promote* G a P) =
  (*ill-deduct-wf* P ∧ *ill-conclusion* P = *map Exp* $G \vdash a$)

In some proofs phasing well-formedness in terms of *antecedents* and *consequent* is more useful.

lemmas *ill-deduct-wf-alt* = *ill-deduct-wf.simps*[*unfolded ill-conclusion-alt*]

end

Premises of a deduction can be gathered recursively. Because every element of the result is an instance of *Premise*, we represent them with the relevant three parameters (antecedents, consequent, label).

primrec *ill-deduct-premises*
  :: ('a, 'l) *ill-deduct* ⇒ ('a *ill-prop list* × 'a *ill-prop* × 'l) *list*
 where
  ill-deduct-premises (*Premise* G c l) = [(G, c, l)]
| *ill-deduct-premises* (*Identity* a) = []
| *ill-deduct-premises* (*Exchange* G a b D c P) = *ill-deduct-premises* P
| *ill-deduct-premises* (*Cut* G b D E c P Q) =
  (*ill-deduct-premises* P @ *ill-deduct-premises* Q)
| *ill-deduct-premises* (*TimesL* G a b D c P) = *ill-deduct-premises* P
| *ill-deduct-premises* (*TimesR* G a D b P Q) =
  (*ill-deduct-premises* P @ *ill-deduct-premises* Q)
| *ill-deduct-premises* (*OneL* G D c P) = *ill-deduct-premises* P
| *ill-deduct-premises* (*OneR*) = []
| *ill-deduct-premises* (*LimpL* G a D b E c P Q) =
  (*ill-deduct-premises* P @ *ill-deduct-premises* Q)
| *ill-deduct-premises* (*LimpR* G a D b P) = *ill-deduct-premises* P
| *ill-deduct-premises* (*WithL1* G a b D c P) = *ill-deduct-premises* P
| *ill-deduct-premises* (*WithL2* G a b D c P) = *ill-deduct-premises* P
| *ill-deduct-premises* (*WithR* G a b P Q) =

$(ill\text{-}deduct\text{-}premises\ P\ @\ ill\text{-}deduct\text{-}premises\ Q)$
| $ill\text{-}deduct\text{-}premises\ (TopR\ G) = []$
| $ill\text{-}deduct\text{-}premises\ (PlusL\ G\ a\ b\ D\ c\ P\ Q) =$
$(ill\text{-}deduct\text{-}premises\ P\ @\ ill\text{-}deduct\text{-}premises\ Q)$
| $ill\text{-}deduct\text{-}premises\ (PlusR1\ G\ a\ b\ P) = ill\text{-}deduct\text{-}premises\ P$
| $ill\text{-}deduct\text{-}premises\ (PlusR2\ G\ a\ b\ P) = ill\text{-}deduct\text{-}premises\ P$
| $ill\text{-}deduct\text{-}premises\ (ZeroL\ G\ D\ c) = []$
| $ill\text{-}deduct\text{-}premises\ (Weaken\ G\ D\ b\ a\ P) = ill\text{-}deduct\text{-}premises\ P$
| $ill\text{-}deduct\text{-}premises\ (Contract\ G\ a\ D\ b\ P) = ill\text{-}deduct\text{-}premises\ P$
| $ill\text{-}deduct\text{-}premises\ (Derelict\ G\ a\ D\ b\ P) = ill\text{-}deduct\text{-}premises\ P$
| $ill\text{-}deduct\text{-}premises\ (Promote\ G\ a\ P) = ill\text{-}deduct\text{-}premises\ P$

1.8.2 Soundness

Deeply embedded deductions are sound with respect to $(\vdash)$ in the sense that the conclusion of any well-formed deduction is a valid sequent if all of its premises are assumed to be valid sequents. This is proven easily, because our definitions stem from the $(\vdash)$ relation.

lemma *ill-deduct-sound*:
assumes $ill\text{-}deduct\text{-}wf\ P$
and $\bigwedge a\ c\ l.\ (a,\ c,\ l) \in set\ (ill\text{-}deduct\text{-}premises\ P) \Longrightarrow ill\text{-}sequent\text{-}valid\ (Sequent\ a\ c)$
shows $ill\text{-}sequent\text{-}valid\ (ill\text{-}conclusion\ P)$
$\langle proof \rangle$

1.8.3 Completeness

Deeply embedded deductions are complete with respect to $(\vdash)$ in the sense that for any valid sequent there exists a well-formed deduction with no premises that has it as its conclusion. This is proven easily, because the deduction nodes map directly onto the rules of the $(\vdash)$ relation.

lemma *ill-deduct-complete*:
assumes $G \vdash c$
shows $\exists P.\ ill\text{-}conclusion\ P = Sequent\ G\ c \wedge ill\text{-}deduct\text{-}wf\ P \wedge ill\text{-}deduct\text{-}premises\ P = []$
$\langle proof \rangle$

1.8.4 Derived Deductions

We define a number of useful deduction patterns as (potentially recursive) functions. In each case we verify the well-formedness, conclusion and premises.

Swap order in a times proposition: $[a \otimes b] \vdash b \otimes a$:

fun $ill\text{-}deduct\text{-}swap :: 'a\ ill\text{-}prop \Rightarrow 'a\ ill\text{-}prop \Rightarrow ('a,\ 'l)\ ill\text{-}deduct$
where $ill\text{-}deduct\text{-}swap\ a\ b =$
$TimesL\ []\ a\ b\ []\ (b \otimes a)$

(*Exchange* [] b a [] $(b \otimes a)$
(*TimesR* [b] b [a] a (*Identity* b) (*Identity* a)))

lemma *ill-deduct-swap* [*simp*]:
ill-deduct-wf (*ill-deduct-swap* a b)
ill-conclusion (*ill-deduct-swap* a b) = *Sequent* $[a \otimes b]$ $(b \otimes a)$
ill-deduct-premises (*ill-deduct-swap* a b) = []
$\langle proof \rangle$

Simplified cut rule: $[\![G \vdash b;\ [b] \vdash c]\!] \Longrightarrow G \vdash c$:

fun *ill-deduct-simple-cut* :: ($'a$, $'l$) *ill-deduct* ⇒ ($'a$, $'l$) *ill-deduct* ⇒ ($'a$, $'l$) *ill-deduct*
where *ill-deduct-simple-cut* P Q = *Cut* (*antecedents* P) (*consequent* P) [] [] (*consequent* Q) P Q

lemma *ill-deduct-simple-cut* [*simp*]:
$[\![$[*consequent* P] = *antecedents* Q; *ill-deduct-wf* P; *ill-deduct-wf* $Q]\!] \Longrightarrow$
ill-deduct-wf (*ill-deduct-simple-cut* P Q)
[*consequent* P] = *antecedents* $Q \Longrightarrow$
ill-conclusion (*ill-deduct-simple-cut* P Q) = *Sequent* (*antecedents* P) (*consequent* Q)
ill-deduct-premises (*ill-deduct-simple-cut* P Q) = *ill-deduct-premises* P @ *ill-deduct-premises* Q
$\langle proof \rangle$

Combine two deductions with times: $[\![[a] \vdash b;\ [c] \vdash d]\!] \Longrightarrow [a \otimes c] \vdash b \otimes d$:

fun *ill-deduct-tensor* :: ($'a$, $'l$) *ill-deduct* ⇒ ($'a$, $'l$) *ill-deduct* ⇒ ($'a$, $'l$) *ill-deduct*
where *ill-deduct-tensor* p q =
TimesL [] (*hd* (*antecedents* p)) (*hd* (*antecedents* q)) [] (*consequent* p ⊗ *consequent* q)
(*TimesR* (*antecedents* p) (*consequent* p) (*antecedents* q) (*consequent* q) p q)

lemma *ill-deduct-tensor* [*simp*]:
$[\![$*antecedents* $P = [a]$; *antecedents* $Q = [c]$; *ill-deduct-wf* P; *ill-deduct-wf* $Q]\!] \Longrightarrow$
ill-deduct-wf (*ill-deduct-tensor* P Q)
$[\![$*antecedents* $P = [a]$; *antecedents* $Q = [c]]\!] \Longrightarrow$
ill-conclusion (*ill-deduct-tensor* P Q) = *Sequent* $[a \otimes c]$ (*consequent* P ⊗ *consequent* Q)
ill-deduct-premises (*ill-deduct-tensor* P Q) = *ill-deduct-premises* P @ *ill-deduct-premises* Q
$\langle proof \rangle$

Associate times proposition to right: $[(a \otimes b) \otimes c] \vdash a \otimes b \otimes c$:

fun *ill-deduct-assoc* :: $'a$ *ill-prop* ⇒ $'a$ *ill-prop* ⇒ $'a$ *ill-prop* ⇒ ($'a$, $'l$) *ill-deduct*
where *ill-deduct-assoc* a b c =
TimesL [] $(a \otimes b)$ c [] $(a \otimes (b \otimes c))$
(*Exchange* [] c $(a \otimes b)$ [] $(a \otimes (b \otimes c))$
(*TimesL* [c] a b [] $(a \otimes (b \otimes c))$
(*Exchange* [] a c [b] $(a \otimes (b \otimes c))$
(*TimesR* [a] a [c, b] $(b \otimes c)$

(*Identity* a)
(*Exchange* [] b c [] ($b \otimes c$)
  (*TimesR* [b] b [c] c
    (*Identity* b)
    (*Identity* c)))))))

lemma *ill-deduct-assoc* [*simp*]:
ill-deduct-wf (*ill-deduct-assoc* a b c)
ill-conclusion (*ill-deduct-assoc* a b c) = *Sequent* [($a \otimes b$) $\otimes$ c] ($a \otimes$ ($b \otimes c$))
ill-deduct-premises (*ill-deduct-assoc* a b c) = []
⟨*proof*⟩

Associate times proposition to left: [$a \otimes b \otimes c$] $\vdash$ ($a \otimes b$) $\otimes$ c:

fun *ill-deduct-assoc'* :: *'a ill-prop* ⇒ *'a ill-prop* ⇒ *'a ill-prop* ⇒ (*'a*, *'l*) *ill-deduct*
where *ill-deduct-assoc'* a b c =
TimesL [] a ($b \otimes c$) [] (($a \otimes b$) $\otimes$ c)
(*TimesL* [a] b c [] (($a \otimes b$) $\otimes$ c)
  (*TimesR* [a, b] ($a \otimes b$) [c] c
    (*TimesR* [a] a [b] b
      (*Identity* a)
      (*Identity* b))
    (*Identity* c)))

lemma *ill-deduct-assoc'* [*simp*]:
ill-deduct-wf (*ill-deduct-assoc'* a b c)
ill-conclusion (*ill-deduct-assoc'* a b c) = *Sequent* [$a \otimes$ ($b \otimes c$)] (($a \otimes b$) $\otimes$ c)
ill-deduct-premises (*ill-deduct-assoc'* a b c) = []
⟨*proof*⟩

Eliminate times unit a proposition: [$a \otimes \mathbf{1}$] $\vdash$ a:

fun *ill-deduct-unit* :: *'a ill-prop* ⇒ (*'a*, *'l*) *ill-deduct*
where *ill-deduct-unit* a = *TimesL* [] a ($\mathbf{1}$) [] a (*OneL* [a] [] a (*Identity* a))

lemma *ill-deduct-unit* [*simp*]:
ill-deduct-wf (*ill-deduct-unit* a)
ill-conclusion (*ill-deduct-unit* a) = *Sequent* [$a \otimes \mathbf{1}$] a
ill-deduct-premises (*ill-deduct-unit* a) = []
⟨*proof*⟩

Introduce times unit into a proposition [a] $\vdash$ $a \otimes \mathbf{1}$:

fun *ill-deduct-unit'* :: *'a ill-prop* ⇒ (*'a*, *'l*) *ill-deduct*
where *ill-deduct-unit'* a = *TimesR* [a] a [] ($\mathbf{1}$) (*Identity* a) *OneR*

lemma *ill-deduct-unit'* [*simp*]:
ill-deduct-wf (*ill-deduct-unit'* a)
ill-conclusion (*ill-deduct-unit'* a) = *Sequent* [a] ($a \otimes \mathbf{1}$)
ill-deduct-premises (*ill-deduct-unit'* a) = []
⟨*proof*⟩

Simplified weakening: $[!\ a] \vdash \mathbf{1}$:

fun *ill-deduct-simple-weaken* :: *'a ill-prop* ⇒ (*'a*, *'l*) *ill-deduct*
  where *ill-deduct-simple-weaken a* = *Weaken* [] [] (**1**) *a OneR*

lemma *ill-deduct-simple-weaken* [*simp*]:
  ill-deduct-wf (*ill-deduct-simple-weaken a*)
  ill-conclusion (*ill-deduct-simple-weaken a*) = *Sequent* [!*a*] **1**
  ill-deduct-premises (*ill-deduct-simple-weaken a*) = []
  ⟨*proof*⟩

Simplified dereliction: $[!\ a] \vdash a$:

fun *ill-deduct-dereliction* :: *'a ill-prop* ⇒ (*'a*, *'l*) *ill-deduct*
  where *ill-deduct-dereliction a* = *Derelict* [] *a* [] *a* (*Identity a*)

lemma *ill-deduct-dereliction* [*simp*]:
  ill-deduct-wf (*ill-deduct-dereliction a*)
  ill-conclusion (*ill-deduct-dereliction a*) = *Sequent* [!*a*] *a*
  ill-deduct-premises (*ill-deduct-dereliction a*) = []
  ⟨*proof*⟩

Duplicate exponentiated proposition: $[!\ a] \vdash !\ a \otimes !\ a$:

fun *ill-deduct-duplicate* :: *'a ill-prop* ⇒ (*'a*, *'l*) *ill-deduct*
  where *ill-deduct-duplicate a* =
    Contract [] *a* [] (!*a* ⊗ !*a*) (*TimesR* [!*a*] (!*a*) [!*a*] (!*a*) (*Identity* (!*a*)) (*Identity* (!*a*)))

lemma *ill-deduct-duplicate* [*simp*]:
  ill-deduct-wf (*ill-deduct-duplicate a*)
  ill-conclusion (*ill-deduct-duplicate a*) = *Sequent* [!*a*] (!*a* ⊗ !*a*)
  ill-deduct-premises (*ill-deduct-duplicate a*) = []
  ⟨*proof*⟩

Simplified plus elimination: $[\![[a] \vdash c;\ [b] \vdash c]\!] \Longrightarrow [a \oplus b] \vdash c$:

fun *ill-deduct-simple-plusL* :: (*'a*, *'l*) *ill-deduct* ⇒ (*'a*, *'l*) *ill-deduct* ⇒ (*'a*, *'l*) *ill-deduct*
  where *ill-deduct-simple-plusL p q* =
    PlusL [] (*hd* (*antecedents p*)) (*hd* (*antecedents q*)) [] (*consequent p*) *p q*

lemma *ill-deduct-simple-plusL* [*simp*]:
  ⟦ *antecedents P* = [*a*]; *antecedents Q* = [*b*]; *ill-deduct-wf P*
  ; *ill-deduct-wf Q*; *consequent P* = *consequent Q*⟧ ⟹
    ill-deduct-wf (*ill-deduct-simple-plusL P Q*)
  ⟦*antecedents P* = [*a*]; *antecedents Q* = [*b*]⟧ ⟹
    ill-conclusion (*ill-deduct-simple-plusL P Q*) = *Sequent* [*a* ⊕ *b*] (*consequent P*)
  ill-deduct-premises (*ill-deduct-simple-plusL P Q*)
  = *ill-deduct-premises P* @ *ill-deduct-premises Q*
  ⟨*proof*⟩

Simplified left plus introduction: $[a] \vdash a \oplus b$:

fun *ill-deduct-plusR1* :: $'a$ *ill-prop* $\Rightarrow$ $'a$ *ill-prop* $\Rightarrow$ $('a, 'l)$ *ill-deduct*
  where *ill-deduct-plusR1 a b* = *PlusR1* [*a*] *a b* (*Identity a*)

lemma *ill-deduct-plusR1* [*simp*]:
  ill-deduct-wf (*ill-deduct-plusR1 a b*)
  ill-conclusion (*ill-deduct-plusR1 a b*) = *Sequent* [*a*] ($a \oplus b$)
  ill-deduct-premises (*ill-deduct-plusR1 a b*) = []
  ⟨*proof*⟩

Simplified right plus introduction: $[b] \vdash a \oplus b$:

fun *ill-deduct-plusR2* :: $'a$ *ill-prop* $\Rightarrow$ $'a$ *ill-prop* $\Rightarrow$ $('a, 'l)$ *ill-deduct*
  where *ill-deduct-plusR2 a b* = *PlusR2* [*b*] *a b* (*Identity b*)

lemma *ill-deduct-plusR2* [*simp*]:
  ill-deduct-wf (*ill-deduct-plusR2 a b*)
  ill-conclusion (*ill-deduct-plusR2 a b*) = *Sequent* [*b*] ($a \oplus b$)
  ill-deduct-premises (*ill-deduct-plusR2 a b*) = []
  ⟨*proof*⟩

Simplified linear implication introduction: $[a] \vdash b \Longrightarrow [\mathbf{1}] \vdash a \rhd b$:

fun *ill-deduct-simple-limpR* :: $('a, 'l)$ *ill-deduct* $\Rightarrow$ $('a, 'l)$ *ill-deduct*
  where *ill-deduct-simple-limpR p* =
    LimpR [] (*hd* (*antecedents p*)) [**1**] (*consequent p*)
    (*OneL* [*hd* (*antecedents p*)] [] (*consequent p*) *p*)

lemma *ill-deduct-simple-limpR* [*simp*]:
  ⟦*antecedents P* = [*a*]; *consequent P* = *b*; *ill-deduct-wf P*⟧ $\Longrightarrow$
    ill-deduct-wf (*ill-deduct-simple-limpR P*)
  ⟦*antecedents P* = [*a*]; *consequent P* = *b*⟧ $\Longrightarrow$
    ill-conclusion (*ill-deduct-simple-limpR P*) = *Sequent* [**1**] ($a \rhd b$)
   ill-deduct-premises (*ill-deduct-simple-limpR P*)
  = *ill-deduct-premises P*
  ⟨*proof*⟩

Simplified introduction of exponentiated impliciation: $[a] \vdash b \Longrightarrow [\mathbf{1}] \vdash\ !\ (a \rhd b)$:

fun *ill-deduct-simple-limpR-exp* :: $('a, 'l)$ *ill-deduct* $\Rightarrow$ $('a, 'l)$ *ill-deduct*
  where *ill-deduct-simple-limpR-exp p* =
    OneL [] [] (!((*hd* (*antecedents p*)) $\rhd$ (*consequent p*)))
    (*Promote* [] ((*hd* (*antecedents p*)) $\rhd$ (*consequent p*))
      (*ill-deduct-simple-cut*
        OneR
        (*ill-deduct-simple-limpR p*)))

lemma *ill-deduct-simple-limpR-exp* [*simp*]:
  ⟦*antecedents P* = [*a*]; *consequent P* = *b*; *ill-deduct-wf P*⟧ $\Longrightarrow$
    ill-deduct-wf (*ill-deduct-simple-limpR-exp P*)
  ⟦*antecedents P* = [*a*]; *consequent P* = *b*⟧ $\Longrightarrow$
    ill-conclusion (*ill-deduct-simple-limpR-exp P*) = *Sequent* [**1**] (!($a \rhd b$))

ill-deduct-premises (*ill-deduct-simple-limpR-exp* P) = *ill-deduct-premises* P
⟨*proof*⟩

Linear implication elimination with times: $[a \otimes a \rhd b] \vdash b$:

fun *ill-deduct-limp-eval* :: $'a$ *ill-prop* ⇒ $'a$ *ill-prop* ⇒ ($'a$, $'l$) *ill-deduct*
where *ill-deduct-limp-eval* a b =
TimesL [] a ($a \rhd b$) [] b (*LimpL* [a] a [] b [] b (*Identity* a) (*Identity* b))

lemma *ill-deduct-limp-eval* [*simp*]:
ill-deduct-wf (*ill-deduct-limp-eval* a b)
ill-conclusion (*ill-deduct-limp-eval* a b) = *Sequent* $[a \otimes a \rhd b]$ b
ill-deduct-premises (*ill-deduct-limp-eval* a b) = []
⟨*proof*⟩

Exponential implication elimination with times: $[a \otimes !\ (a \rhd b)] \vdash b \otimes !\ (a \rhd b)$:

fun *ill-deduct-explimp-eval* :: $'a$ *ill-prop* ⇒ $'a$ *ill-prop* ⇒ ($'a$, $'l$) *ill-deduct*
where *ill-deduct-explimp-eval* a b =
TimesL [] a ($!(a \rhd b)$) [] ($b \otimes !(a \rhd b)$) (
Contract [a] ($a \rhd b$) [] ($b \otimes !(a \rhd b)$) (
TimesR [a, $!(a \rhd b)$] b [$!(a \rhd b)$] ($!(a \rhd b)$)
(*Derelict* [a] ($a \rhd b$) [] b (
LimpL [a] a [] b [] b
(*Identity* a)
(*Identity* b)))
(*Identity* ($!(a \rhd b)$)))))

lemma *ill-deduct-explimp-eval* [*simp*]:
ill-deduct-wf (*ill-deduct-explimp-eval* a b)
ill-conclusion (*ill-deduct-explimp-eval* a b) = *Sequent* $[a \otimes !(a \rhd b)]$ ($b \otimes !(a \rhd b)$)
ill-deduct-premises (*ill-deduct-explimp-eval* a b) = []
⟨*proof*⟩

Distributing times over plus: $[a \otimes b \oplus c] \vdash (a \otimes b) \oplus a \otimes c$:

fun *ill-deduct-distrib-plus* :: $'a$ *ill-prop* ⇒ $'a$ *ill-prop* ⇒ $'a$ *ill-prop* ⇒ ($'a$, $'l$) *ill-deduct*
where *ill-deduct-distrib-plus* a b c =
TimesL [] a ($b \oplus c$) [] (($a \otimes b$) ⊕ ($a \otimes c$))
(*PlusL* [a] b c [] (($a \otimes b$) ⊕ ($a \otimes c$))
(*PlusR1* [a, b] ($a \otimes b$) ($a \otimes c$)
(*TimesR* [a] a [b] b
(*Identity* a)
(*Identity* b)))
(*PlusR2* [a, c] ($a \otimes b$) ($a \otimes c$)
(*TimesR* [a] a [c] c
(*Identity* a)
(*Identity* c))))

lemma *ill-deduct-distrib-plus* [*simp*]:
ill-deduct-wf (*ill-deduct-distrib-plus a b c*)
ill-conclusion (*ill-deduct-distrib-plus a b c*) = *Sequent* $[a \otimes (b \oplus c)]$ $((a \otimes b) \oplus (a \otimes c))$
ill-deduct-premises (*ill-deduct-distrib-plus a b c*) = []
⟨*proof*⟩

Distributing times out of plus: $[(a \otimes b) \oplus a \otimes c] \vdash a \otimes b \oplus c$:

fun *ill-deduct-distrib-plus′* :: *′a ill-prop* ⇒ *′a ill-prop* ⇒ *′a ill-prop* ⇒ (*′a*, *′l*) *ill-deduct*
where *ill-deduct-distrib-plus′ a b c* =
PlusL [] $(a \otimes b)$ $(a \otimes c)$ [] $(a \otimes (b \oplus c))$
(*ill-deduct-tensor*
(*Identity a*)
(*ill-deduct-plusR1 b c*))
(*ill-deduct-tensor*
(*Identity a*)
(*ill-deduct-plusR2 b c*))

lemma *ill-deduct-distrib-plus′* [*simp*]:
ill-deduct-wf (*ill-deduct-distrib-plus′ a b c*)
ill-conclusion (*ill-deduct-distrib-plus′ a b c*) = *Sequent* $[(a \otimes b) \oplus (a \otimes c)]$ $(a \otimes (b \oplus c))$
ill-deduct-premises (*ill-deduct-distrib-plus′ a b c*) = []
⟨*proof*⟩

Combining two deductions with plus: $[\![[a] \vdash b; [c] \vdash d]\!] \Longrightarrow [a \oplus c] \vdash b \oplus d$:

fun *ill-deduct-plus-progress* :: (*′a*, *′l*) *ill-deduct* ⇒ (*′a*, *′l*) *ill-deduct* ⇒ (*′a*, *′l*) *ill-deduct*
where *ill-deduct-plus-progress p q* =
ill-deduct-simple-plusL
(*ill-deduct-simple-cut p* (*ill-deduct-plusR1* (*consequent p*) (*consequent q*)))
(*ill-deduct-simple-cut q* (*ill-deduct-plusR2* (*consequent p*) (*consequent q*)))

lemma *ill-deduct-plus-progress* [*simp*]:
⟦*antecedents P* = [*a*]; *antecedents Q* = [*c*]; *ill-deduct-wf P*; *ill-deduct-wf Q*⟧ ⟹
ill-deduct-wf (*ill-deduct-plus-progress P Q*)
⟦*antecedents P* = [*a*]; *antecedents Q* = [*c*]⟧ ⟹
ill-conclusion (*ill-deduct-plus-progress P Q*) = *Sequent* $[a \oplus c]$ (*consequent P* ⊕ *consequent Q*)
ill-deduct-premises (*ill-deduct-plus-progress P Q*)
= *ill-deduct-premises P* @ *ill-deduct-premises Q*
⟨*proof*⟩

Simplified with introduction: $[\![[a] \vdash b; [a] \vdash c]\!] \Longrightarrow [a] \vdash b \ \& \ c$:

fun *ill-deduct-with* :: (*′a*, *′l*) *ill-deduct* ⇒ (*′a*, *′l*) *ill-deduct* ⇒ (*′a*, *′l*) *ill-deduct*
where *ill-deduct-with p q* = *WithR* [*hd* (*antecedents p*)] (*consequent p*) (*consequent q*) *p q*

lemma *ill-deduct-with* [*simp*]:
$[\![$ *antecedents* $P = [a]$; *antecedents* $Q = [a]$; *consequent* $P = b$
; *consequent* $Q = c$; *ill-deduct-wf* P; *ill-deduct-wf* $Q]\!] \Longrightarrow$
ill-deduct-wf (*ill-deduct-with* P Q)
$[\![$*antecedents* $P = [a]$; *antecedents* $Q = [a]$; *consequent* $P = b$; *consequent* $Q = c]\!]$
$\Longrightarrow$
ill-conclusion (*ill-deduct-with* P Q) = *Sequent* $[a]$ (*consequent* P & *consequent*
Q)
ill-deduct-premises (*ill-deduct-with* P Q) = *ill-deduct-premises* P @ *ill-deduct-premises*
Q
⟨*proof*⟩

Simplified with left projection: $[a \ \& \ b] \vdash a$:

fun *ill-deduct-projectL* :: *'a ill-prop* ⇒ *'a ill-prop* ⇒ (*'a*, *'l*) *ill-deduct*
where *ill-deduct-projectL* a b = *WithL1* [] a b [] a (*Identity* a)

lemma *ill-deduct-projectL* [*simp*]:
ill-deduct-wf (*ill-deduct-projectL* a b)
ill-conclusion (*ill-deduct-projectL* a b) = *Sequent* $[a \ \& \ b]$ a
ill-deduct-premises (*ill-deduct-projectL* a b) = []
⟨*proof*⟩

Simplified with right projection: $[a \ \& \ b] \vdash b$:

fun *ill-deduct-projectR* :: *'a ill-prop* ⇒ *'a ill-prop* ⇒ (*'a*, *'l*) *ill-deduct*
where *ill-deduct-projectR* a b = *WithL2* [] a b [] b (*Identity* b)

lemma *ill-deduct-projectR* [*simp*]:
ill-deduct-wf (*ill-deduct-projectR* a b)
ill-conclusion (*ill-deduct-projectR* a b) = *Sequent* $[a \ \& \ b]$ b
ill-deduct-premises (*ill-deduct-projectR* a b) = []
⟨*proof*⟩

Distributing times over with: $[a \otimes b \ \& \ c] \vdash (a \otimes b) \ \& \ a \otimes c$:

fun *ill-deduct-distrib-with* :: *'a ill-prop* ⇒ *'a ill-prop* ⇒ *'a ill-prop* ⇒ (*'a*, *'l*)
ill-deduct
where *ill-deduct-distrib-with* a b c =
WithR $[a \otimes (b \ \& \ c)]$ $(a \otimes b)$ $(a \otimes c)$
(*ill-deduct-tensor*
(*Identity* a)
(*ill-deduct-projectL* b c))
(*ill-deduct-tensor*
(*Identity* a)
(*ill-deduct-projectR* b c))

lemma *ill-deduct-distrib-with* [*simp*]:
ill-deduct-wf (*ill-deduct-distrib-with* a b c)
ill-conclusion (*ill-deduct-distrib-with* a b c) = *Sequent* $[a \otimes (b \ \& \ c)]$ $((a \otimes b)$ &
$(a \otimes c))$
ill-deduct-premises (*ill-deduct-distrib-with* a b c) = []

$\langle proof \rangle$

Weakening a list of propositions: G @ $D \vdash b \Longrightarrow G$ @ map ! xs @ $D \vdash b$:

fun *ill-deduct-weaken-list*
  :: $'a$ *ill-prop list* $\Rightarrow$ $'a$ *ill-prop list* $\Rightarrow$ $'a$ *ill-prop list* $\Rightarrow$ $('a, 'l)$ *ill-deduct*
  $\Rightarrow$ $('a, 'l)$ *ill-deduct*
 where
  ill-deduct-weaken-list G D [] *P* = *P*
 | *ill-deduct-weaken-list G D* (*x*#*xs*) *P* =
   Weaken G (*map Exp xs* @ *D*) (*consequent P*) *x* (*ill-deduct-weaken-list G D xs P*)

lemma *ill-deduct-weaken-list* [*simp*]:
 ⟦*antecedents P* = *G* @ *D*; *ill-deduct-wf P*⟧ $\Longrightarrow$ *ill-deduct-wf* (*ill-deduct-weaken-list G D xs P*)
 antecedents P = *G* @ *D* ∨ *xs* ≠ [] $\Longrightarrow$
  antecedents (*ill-deduct-weaken-list G D xs P*) = *G* @ (*map Exp xs*) @ *D*
 consequent (*ill-deduct-weaken-list G D xs P*) = *consequent P*
 ill-deduct-premises (*ill-deduct-weaken-list G D xs P*) = *ill-deduct-premises P*
$\langle proof \rangle$

Exponentiating a deduction: $G \vdash b \Longrightarrow map$! $G \vdash$! b

fun *ill-deduct-exp-helper* :: *nat* $\Rightarrow$ $('a, 'l)$ *ill-deduct* $\Rightarrow$ $('a, 'l)$ *ill-deduct*
 — Helper function to apply *Derelict* to first n antecedents
 where
  ill-deduct-exp-helper 0 P = *P*
 | *ill-deduct-exp-helper* (*Suc n*) *P* =
   Derelict
    (*map Exp* (*take n* (*antecedents P*)))
    (*nth* (*antecedents P*) *n*)
    (*drop* (*Suc n*) (*antecedents P*))
    (*consequent P*)
    (*ill-deduct-exp-helper n P*)

lemma *ill-deduct-exp-helper*:
 $n \leq$ *length* (*antecedents P*) $\Longrightarrow$
   antecedents (*ill-deduct-exp-helper n P*)
  = *map Exp* (*take n* (*antecedents P*)) @ *drop n* (*antecedents P*)
 consequent (*ill-deduct-exp-helper n P*) = *consequent P*
 $n \leq$ *length* (*antecedents P*) $\Longrightarrow$ *ill-deduct-wf* (*ill-deduct-exp-helper n P*) = *ill-deduct-wf P*
 ill-deduct-premises (*ill-deduct-exp-helper n P*) = *ill-deduct-premises P*
$\langle proof \rangle$

fun *ill-deduct-exp* :: $('a, 'l)$ *ill-deduct* $\Rightarrow$ $('a, 'l)$ *ill-deduct*
 where *ill-deduct-exp P* =
  Promote (*antecedents P*) (*consequent P*) (*ill-deduct-exp-helper* (*length* (*antecedents P*)) *P*)

lemma *ill-deduct-exp* [*simp*]:
ill-conclusion (*ill-deduct-exp P*) = *Sequent* (*map Exp* (*antecedents P*)) (!(*consequent P*))
ill-deduct-wf (*ill-deduct-exp P*) = *ill-deduct-wf P*
ill-deduct-premises (*ill-deduct-exp P*) = *ill-deduct-premises P*
⟨*proof*⟩

1.8.5 Compacting Equivalences

Compacting cons equivalence: $a \otimes$ *compact* $b \dashv\vdash$ *compact* $(a \# b)$:

primrec *ill-deduct-times-to-compact-cons* :: *'a ill-prop* ⇒ *'a ill-prop list* ⇒ (*'a*, *'l*) *ill-deduct*
— [$a \otimes$ *compact b*] ⊢ *compact* ($a \# b$)
where
ill-deduct-times-to-compact-cons a [] = *ill-deduct-unit a*
| *ill-deduct-times-to-compact-cons a* (*b#bs*) = *Identity* ($a \otimes$ *compact* (*b#bs*))

lemma *ill-deduct-times-to-compact-cons* [*simp*]:
ill-deduct-wf (*ill-deduct-times-to-compact-cons a b*)
ill-conclusion (*ill-deduct-times-to-compact-cons a b*)
= *Sequent* [$a \otimes$ *compact b*] (*compact* ($a \# b$))
ill-deduct-premises (*ill-deduct-times-to-compact-cons a b*) = []
⟨*proof*⟩

primrec *ill-deduct-compact-cons-to-times* :: *'a ill-prop* ⇒ *'a ill-prop list* ⇒ (*'a*, *'l*) *ill-deduct*
— [*compact* ($a \# b$)] ⊢ $a \otimes$ *compact b*
where
ill-deduct-compact-cons-to-times a [] = *ill-deduct-unit' a*
| *ill-deduct-compact-cons-to-times a* (*b#bs*) = *Identity* ($a \otimes$ *compact* (*b#bs*))

lemma *ill-deduct-compact-cons-to-times* [*simp*]:
ill-deduct-wf (*ill-deduct-compact-cons-to-times a b*)
ill-conclusion (*ill-deduct-compact-cons-to-times a b*)
= *Sequent* [*compact* ($a \# b$)] ($a \otimes$ *compact b*)
ill-deduct-premises (*ill-deduct-compact-cons-to-times a b*) = []
⟨*proof*⟩

Compacting append equivalence: *compact* $a \otimes$ *compact* $b \dashv\vdash$ *compact* (a @ b):

primrec *ill-deduct-times-to-compact-append*
:: *'a ill-prop list* ⇒ *'a ill-prop list* ⇒ (*'a*, *'l*) *ill-deduct*
— [*compact a* ⊗ *compact b*] ⊢ *compact* (*a* @ *b*)
where
ill-deduct-times-to-compact-append [] *b* =
ill-deduct-simple-cut (*ill-deduct-swap* (**1**) (*compact b*)) (*ill-deduct-unit* (*compact b*))
| *ill-deduct-times-to-compact-append* (*a#as*) *b* =

```
    ill-deduct-simple-cut
    ( ill-deduct-simple-cut
      ( ill-deduct-simple-cut
        ( ill-deduct-tensor
          ( ill-deduct-compact-cons-to-times a as)
          ( Identity (compact b)))
        ( ill-deduct-assoc a (compact as) (compact b)))
      ( ill-deduct-tensor
        ( Identity a)
        ( ill-deduct-times-to-compact-append as b)))
    ( ill-deduct-times-to-compact-cons a (as @ b))
```

lemma *ill-deduct-times-to-compact-append* [*simp*]:

```
  ill-deduct-wf (ill-deduct-times-to-compact-append a b :: ('a, 'l) ill-deduct)
   ill-conclusion (ill-deduct-times-to-compact-append a b :: ('a, 'l) ill-deduct)
  = Sequent [compact a ⊗ compact b] (compact (a @ b))
  ill-deduct-premises (ill-deduct-times-to-compact-append a b) = []
  ⟨proof⟩
```

primrec *ill-deduct-compact-append-to-times*

```
    :: 'a ill-prop list ⇒ 'a ill-prop list ⇒ ('a, 'l) ill-deduct
  — [compact (a @ b)] ⊢ compact a ⊗ compact b
  where
    ill-deduct-compact-append-to-times [] b =
      ill-deduct-simple-cut
        ( ill-deduct-unit' (compact b))
        ( ill-deduct-swap (compact b) (1))
  | ill-deduct-compact-append-to-times (a#as) b =
      ill-deduct-simple-cut
      ( ill-deduct-compact-cons-to-times a (as @ b))
      ( ill-deduct-simple-cut
        ( ill-deduct-tensor
          ( Identity a)
          ( ill-deduct-compact-append-to-times as b))
        ( ill-deduct-simple-cut
          ( ill-deduct-assoc' a (compact as) (compact b))
          ( ill-deduct-tensor
            ( ill-deduct-times-to-compact-cons a as)
            ( Identity (compact b)))))
```

lemma *ill-deduct-compact-append-to-times* [*simp*]:

```
  ill-deduct-wf (ill-deduct-compact-append-to-times a b :: ('a, 'l) ill-deduct)
   ill-conclusion (ill-deduct-compact-append-to-times a b :: ('a, 'l) ill-deduct)
  = Sequent [compact (a @ b)] (compact a ⊗ compact b)
  ill-deduct-premises (ill-deduct-compact-append-to-times a b) = []
  ⟨proof⟩
```

Combine a list of deductions with times using *ill-deduct-tensor*, representing a generalised version of the following theorem of the shallow embedding:

$\forall x \in set\ ?xs.\ [?f\ x] \vdash\ ?g\ x \Longrightarrow [compact\ (map\ ?f\ ?xs)] \vdash compact\ (map\ ?g\ ?xs)$

primrec *ill-deduct-tensor-list* :: ('a, 'l) *ill-deduct list* ⇒ ('a, 'l) *ill-deduct*
where
ill-deduct-tensor-list [] = *Identity* (**1**)
| *ill-deduct-tensor-list* (*x#xs*) =
(*if xs* = [] *then x else ill-deduct-tensor x* (*ill-deduct-tensor-list xs*))

lemma *ill-deduct-tensor-list* [*simp*]:
fixes *xs* :: ('a, 'l) *ill-deduct list*
assumes $\bigwedge x.\ x \in set\ xs \Longrightarrow \exists\, a.\ antecedents\ x = [a]$
shows *ill-conclusion* (*ill-deduct-tensor-list xs*)
= *Sequent* [*compact* (*map* (*hd* ∘ *antecedents*) *xs*)] (*compact* (*map consequent xs*))
and ($\bigwedge x.\ x \in set\ xs \Longrightarrow$ *ill-deduct-wf x*) ⟹ *ill-deduct-wf* (*ill-deduct-tensor-list xs*)
and *ill-deduct-premises* (*ill-deduct-tensor-list xs*) = *concat* (*map ill-deduct-premises xs*)
⟨*proof*⟩

1.8.6 Premise Substitution

Premise substitution replaces certain premises in a deduction with other deductions. The target premises are specified with a predicate on the three arguments of the *Premise* constructor: antecedents, consequent and label. The replacement for each is specified as a function of those three arguments. In this way, the substitution can replace a whole class of premises in a single pass.

primrec *ill-deduct-subst* ::
('a *ill-prop list* ⇒ 'a *ill-prop* ⇒ 'l ⇒ *bool*) ⇒
('a *ill-prop list* ⇒ 'a *ill-prop* ⇒ 'l ⇒ ('a, 'l) *ill-deduct*) ⇒
('a, 'l) *ill-deduct* ⇒ ('a, 'l) *ill-deduct*
where
ill-deduct-subst p f (*Premise G c l*) = (*if p G c l then f G c l else Premise G c l*)
| *ill-deduct-subst p f* (*Identity a*) = *Identity a*
| *ill-deduct-subst p f* (*Exchange G a b D c P*) = *Exchange G a b D c* (*ill-deduct-subst p f P*)
| *ill-deduct-subst p f* (*Cut G b D E c P Q*) =
Cut G b D E c (*ill-deduct-subst p f P*) (*ill-deduct-subst p f Q*)
| *ill-deduct-subst p f* (*TimesL G a b D c P*) = *TimesL G a b D c* (*ill-deduct-subst p f P*)
| *ill-deduct-subst p f* (*TimesR G a D b P Q*) =
TimesR G a D b (*ill-deduct-subst p f P*) (*ill-deduct-subst p f Q*)
| *ill-deduct-subst p f* (*OneL G D c P*) = *OneL G D c* (*ill-deduct-subst p f P*)
| *ill-deduct-subst p f* (*OneR*) = *OneR*
| *ill-deduct-subst p f* (*LimpL G a D b E c P Q*) =
LimpL G a D b E c (*ill-deduct-subst p f P*) (*ill-deduct-subst p f Q*)

| *ill-deduct-subst p f* (*LimpR G a D b P*) = *LimpR G a D b* (*ill-deduct-subst p f P*)
| *ill-deduct-subst p f* (*WithL1 G a b D c P*) = *WithL1 G a b D c* (*ill-deduct-subst p f P*)
| *ill-deduct-subst p f* (*WithL2 G a b D c P*) = *WithL2 G a b D c* (*ill-deduct-subst p f P*)
| *ill-deduct-subst p f* (*WithR G a b P Q*) =
WithR G a b (*ill-deduct-subst p f P*) (*ill-deduct-subst p f Q*)
| *ill-deduct-subst p f* (*TopR G*) = *TopR G*
| *ill-deduct-subst p f* (*PlusL G a b D c P Q*) =
PlusL G a b D c (*ill-deduct-subst p f P*) (*ill-deduct-subst p f Q*)
| *ill-deduct-subst p f* (*PlusR1 G a b P*) = *PlusR1 G a b* (*ill-deduct-subst p f P*)
| *ill-deduct-subst p f* (*PlusR2 G a b P*) = *PlusR2 G a b* (*ill-deduct-subst p f P*)
| *ill-deduct-subst p f* (*ZeroL G D c*) = *ZeroL G D c*
| *ill-deduct-subst p f* (*Weaken G D b a P*) = *Weaken G D b a* (*ill-deduct-subst p f P*)
| *ill-deduct-subst p f* (*Contract G a D b P*) = *Contract G a D b* (*ill-deduct-subst p f P*)
| *ill-deduct-subst p f* (*Derelict G a D b P*) = *Derelict G a D b* (*ill-deduct-subst p f P*)
| *ill-deduct-subst p f* (*Promote G a P*) = *Promote G a* (*ill-deduct-subst p f P*)

If the target premise is not present, then substitution does nothing

lemma *ill-deduct-subst-no-target*:
($\bigwedge$*G c l.* (*G*, *c*, *l*) ∈ *set* (*ill-deduct-premises x*) ⟹ ¬ *p G c l*) ⟹ *ill-deduct-subst p f x* = *x*
⟨*proof*⟩

If a deduction has no premise, then substitution does nothing

lemma *ill-deduct-subst-no-prems*:
ill-deduct-premises x = [] ⟹ *ill-deduct-subst p f x* = *x*
⟨*proof*⟩

If we substitute the target, then the substitution does nothing

lemma *ill-deduct-subst-of-target* [*simp*]:
f = *Premise* ⟹ *ill-deduct-subst p f x* = *x*
⟨*proof*⟩

Substitution matching the target's antecedents preserves overall deduction antecedents

lemma *ill-deduct-subst-antecedents* [*simp*]:
assumes ($\bigwedge$*G c l. p G c l* ⟹ *antecedents* (*f G c l*) = *G*)
shows *antecedents* (*ill-deduct-subst p f x*) = *antecedents x*
⟨*proof*⟩

Substitution matching the target's consequent preserves overall deduction consequent

lemma *ill-deduct-subst-consequent* [*simp*]:

assumes $\bigwedge G\ c\ l.\ p\ G\ c\ l \Longrightarrow$ *consequent* $(f\ G\ c\ l) = c$
shows *consequent* (*ill-deduct-subst* $p\ f\ x$) $=$ *consequent* x
⟨*proof*⟩

Substitution matching target's antecedent, consequent and well-formedness preserves overall well-formedness

lemma *ill-deduct-subst-wf* [*simp*]:
assumes $\bigwedge G\ c\ l.\ p\ G\ c\ l \Longrightarrow$ *antecedents* $(f\ G\ c\ l) = G$
and $\bigwedge G\ c\ l.\ p\ G\ c\ l \Longrightarrow$ *consequent* $(f\ G\ c\ l) = c$
and $\bigwedge G\ c\ l.\ p\ G\ c\ l \Longrightarrow$ *ill-deduct-wf* $(f\ G\ c\ l)$
shows *ill-deduct-wf* $x =$ *ill-deduct-wf* (*ill-deduct-subst* $p\ f\ x$)
⟨*proof*⟩

Premises after substitution are those that didn't satisfy the predicate and anything that was introduced by the function applied on satisfying premises' parameters.

lemma *ill-deduct-subst-ill-deduct-premises*:
ill-deduct-premises (*ill-deduct-subst* $p\ f\ x$)
$=$ *concat* (*map* ($\lambda(G,\ c,\ l)$.
if $p\ G\ c\ l$ *then ill-deduct-premises* $(f\ G\ c\ l)$ *else* $[(G,\ c,\ l)]$)
(*ill-deduct-premises* x))
⟨*proof*⟩

This substitution commutes with many operations on deductions

lemma
assumes $\bigwedge G\ c\ l.\ p\ G\ c\ l \Longrightarrow$ *antecedents* $(f\ G\ c\ l) = G$
and $\bigwedge G\ c\ l.\ p\ G\ c\ l \Longrightarrow$ *consequent* $(f\ G\ c\ l) = c$
shows *ill-deduct-subst-simple-cut* [*simp*]:
ill-deduct-subst $p\ f$ (*ill-deduct-simple-cut* $X\ Y$)
$=$ *ill-deduct-simple-cut* (*ill-deduct-subst* $p\ f\ X$) (*ill-deduct-subst* $p\ f\ Y$)
and *ill-deduct-subst′-tensor* [*simp*]:
ill-deduct-subst $p\ f$ (*ill-deduct-tensor* $X\ Y$) $=$
ill-deduct-tensor (*ill-deduct-subst* $p\ f\ X$) (*ill-deduct-subst* $p\ f\ Y$)
and *ill-deduct-subst-simple-plusL* [*simp*]:
ill-deduct-subst $p\ f$ (*ill-deduct-simple-plusL* $X\ Y$) $=$
ill-deduct-simple-plusL (*ill-deduct-subst* $p\ f\ X$) (*ill-deduct-subst* $p\ f\ Y$)
and *ill-deduct-subst-with* [*simp*]:
ill-deduct-subst $p\ f$ (*ill-deduct-with* $X\ Y$) $=$
ill-deduct-with (*ill-deduct-subst* $p\ f\ X$) (*ill-deduct-subst* $p\ f\ Y$)
and *ill-deduct-subst-simple-limpR* [*simp*]:
ill-deduct-subst $p\ f$ (*ill-deduct-simple-limpR* X) $=$
ill-deduct-simple-limpR (*ill-deduct-subst* $p\ f\ X$)
and *ill-deduct-subst-simple-limpR-exp* [*simp*]:
ill-deduct-subst $p\ f$ (*ill-deduct-simple-limpR-exp* X) $=$
ill-deduct-simple-limpR-exp (*ill-deduct-subst* $p\ f\ X$)
⟨*proof*⟩

1.8.7 List-Based Exchange

To expand the applicability of the exchange rule to lists of propositions, we first need to establish that the well-formedness of a deduction is not affected by compacting a sublist of the antecedents of its conclusions. This corresponds to the following equality in the shallow embedding of deductions: *?X* @ [*compact ?G*] @ *?Y* ⊢ *?c* = *?X* @ *?G* @ *?Y* ⊢ *?c*.

For one direction of the equality we need to use *TimesL* to recursively add one proposition at a time into the compacted part of the antecedents. Note that, just like *compact*, the recursion terminates in the singleton case.

primrec *ill-deduct-compact-antecedents-split*
  :: *nat* ⇒ *'a ill-prop list* ⇒ *'a ill-prop list* ⇒ *'a ill-prop list* ⇒ (*'a*, *'l*) *ill-deduct*
  ⇒ (*'a*, *'l*) *ill-deduct*
 where
  ill-deduct-compact-antecedents-split 0 X G Y P = *OneL* (*X* @ *G*) *Y* (*consequent P*) *P*
 | *ill-deduct-compact-antecedents-split* (*Suc n*) *X G Y P* = (*if n* = *0 then P else*
   TimesL
    (*X* @ *take* (*length G* − (*Suc n*)) *G*)
    (*hd* (*drop* (*length G* − (*Suc n*)) *G*))
    (*compact* (*drop* (*length G* − *n*) *G*))
    Y
    (*consequent P*)
    (*ill-deduct-compact-antecedents-split n X G Y P*))

lemma *ill-deduct-compact-antecedents-split* [*simp*]:
 assumes *n* ≤ *length G*
  shows *antecedents P* = *X* @ *G* @ *Y* ⟹
    antecedents (*ill-deduct-compact-antecedents-split n X G Y P*)
   = *X* @ *take* (*length G* − *n*) *G* @ [*compact* (*drop* (*length G* − *n*) *G*)] @ *Y*
  and *consequent* (*ill-deduct-compact-antecedents-split n X G Y P*) = *consequent P*
  and ⟦*antecedents P* = *X* @ *G* @ *Y*; *ill-deduct-wf P*⟧ ⟹
    ill-deduct-wf (*ill-deduct-compact-antecedents-split n X G Y P*)
  and *ill-deduct-premises* (*ill-deduct-compact-antecedents-split n X G Y P*)
   = *ill-deduct-premises P*
⟨*proof*⟩

Implication in the uncompacted-to-compacted direction

fun *ill-deduct-antecedents-to-times*
  :: *'a ill-prop list* ⇒ *'a ill-prop list* ⇒ *'a ill-prop list* ⇒ (*'a*, *'l*) *ill-deduct*
  ⇒ (*'a*, *'l*) *ill-deduct*
 — *X* @ *G* @ *Y* ⊢ *c* ⟹ *X* @ [*compact G*] @ *Y* ⊢ *c*
 where *ill-deduct-antecedents-to-times X G Y P* =
  ill-deduct-compact-antecedents-split (*length G*) *X G Y P*

lemma *ill-deduct-antecedents-to-times* [*simp*]:

antecedents $P = X$ @ G @ $Y \Longrightarrow$
antecedents (*ill-deduct-antecedents-to-times* X G Y P) $= X$ @ [*compact* G] @ Y
consequent (*ill-deduct-antecedents-to-times* X G Y P) $=$ *consequent* P
⟦*antecedents* $P = X$ @ G @ Y; *ill-deduct-wf* P⟧ $\Longrightarrow$
ill-deduct-wf (*ill-deduct-antecedents-to-times* X G Y P)
ill-deduct-premises (*ill-deduct-antecedents-to-times* X G Y P) $=$ *ill-deduct-premises* P
⟨*proof*⟩

For the other direction we only need to derive the compacted propositions from the original list. This corresponds to the following valid sequent in the shallow embedding of deductions: *?G* ⊢ *compact ?G*.

fun *ill-deduct-identity-compact* :: *'a ill-prop list* ⇒ (*'a*, *'l*) *ill-deduct*
where
ill-deduct-identity-compact [] = *OneR*
| *ill-deduct-identity-compact* [*x*] = *Identity x*
| *ill-deduct-identity-compact* (*x*#*xs*) =
TimesR [*x*] *x xs* (*compact xs*) (*Identity x*) (*ill-deduct-identity-compact xs*)

lemma *ill-deduct-identity-compact* [*simp*]:
ill-conclusion (*ill-deduct-identity-compact G*) = *Sequent G* (*compact G*)
ill-deduct-wf (*ill-deduct-identity-compact G*)
ill-deduct-premises (*ill-deduct-identity-compact G*) = []
⟨*proof*⟩

Implication in the compacted-to-uncompacted direction

fun *ill-deduct-antecedents-from-times*
:: *'a ill-prop list* ⇒ *'a ill-prop list* ⇒ *'a ill-prop list* ⇒ (*'a*, *'l*) *ill-deduct*
⇒ (*'a*, *'l*) *ill-deduct*
— X @ [*compact* G] @ $Y \vdash c \Longrightarrow X$ @ G @ $Y \vdash c$
where *ill-deduct-antecedents-from-times* X G Y P =
Cut G (*compact* G) X Y (*consequent* P) (*ill-deduct-identity-compact* G) P

lemma *ill-deduct-antecedents-from-times* [*simp*]:
ill-conclusion (*ill-deduct-antecedents-from-times* X G Y P) =
Sequent (X @ G @ Y) (*consequent* P)
⟦*antecedents* $P = X$ @ [*compact* G] @ Y; *ill-deduct-wf* P⟧ $\Longrightarrow$
ill-deduct-wf (*ill-deduct-antecedents-from-times* X G Y P)
ill-deduct-premises (*ill-deduct-antecedents-from-times* X G Y P)
= *ill-deduct-premises* P
⟨*proof*⟩

Finally, we establish the deep embedding of list-based exchange. This corresponds to the following theorem in the shallow embedding of deductions: *?G* @ *?A* @ *?B* @ *?D* ⊢ *?c* $\Longrightarrow$ *?G* @ *?B* @ *?A* @ *?D* ⊢ *?c*.

fun *ill-deduct-exchange-list*
:: *'a ill-prop list* ⇒ *'a ill-prop list* ⇒ *'a ill-prop list* ⇒ *'a ill-prop list* ⇒ *'a ill-prop*

$\Rightarrow$ ($'a$, $'l$) *ill-deduct* $\Rightarrow$ ($'a$, $'l$) *ill-deduct*
where *ill-deduct-exchange-list* G A B D c P =
ill-deduct-antecedents-from-times G B (A @ D)
(*ill-deduct-antecedents-from-times* (G @ [*compact* B]) A D
(*Exchange* G (*compact* A) (*compact* B) D c
(*ill-deduct-antecedents-to-times* (G @ [*compact* A]) B D
(*ill-deduct-antecedents-to-times* G A (B @ D) P))))

lemma *ill-deduct-exchange-list* [*simp*]:
ill-conclusion (*ill-deduct-exchange-list* G A B D c P) = *Sequent* (G @ B @ A @ D) c
$[\![$*ill-deduct-wf* P; *antecedents* P = G @ A @ B @ D; *consequent* P = $c]\!] \Longrightarrow$
ill-deduct-wf (*ill-deduct-exchange-list* G A B D c P)
ill-deduct-premises (*ill-deduct-exchange-list* G A B D c P) = *ill-deduct-premises* P
$\langle$*proof*$\rangle$

end

References

[1] G. M. Bierman. On intuitionistic linear logic. Technical Report UCAM-CL-TR-346, University of Cambridge, Computer Laboratory, Aug. 1994.

[2] S. Kalvala and V. De Paiva. Mechanizing linear logic in Isabelle. In *In 10th International Congress of Logic, Philosophy and Methodology of Science*, volume 24. Citeseer, 1995.