EviL Isabelle/HOL Sessions

Matthew P. Wampler-Doty

January 21, 2010

Contents

1 A Minimal Logic Axiom Class

theory *MinAxClass*
imports *Main*
begin

This file introduces some proof theory for *minimal logic*, the implicational fragment of *intuitionistic logic*. The most important results of this file involve development of some elementary results in the sequent calculus, namely various forms of *deduction theorem*, *monotonicity* and finally *cut*. Presumably one could consider *minimal logic* an axiomatic extension of certain *substructural logics*, but this is admittedly beyond the scope of our project.

As an aside, this file represents a first real attempt to prove anything nontrivial employing *classes* and more advanced Isar proof patterns in Isabelle/HOL. It doesn't run particularly fast, the style is pretty inconsistent, many proofs could probably be simplified, and it is overall not very elegant in our opinion.

class *MinAx* =
  fixes *imp* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** $\rightarrow$ *25*)
  fixes *vdash* :: $'a \Rightarrow$ *bool* ($\vdash$ - [*20*] *20*)
  assumes *ax1*: $\vdash \varphi \rightarrow \psi \rightarrow \varphi$
  assumes *ax2*: $\vdash (\varphi \rightarrow \psi \rightarrow \chi) \rightarrow (\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi)$
  assumes *mp*: $\vdash \varphi \rightarrow \psi \Longrightarrow \vdash \varphi \Longrightarrow \vdash \psi$

Note that *mp* stands for *modus ponens*

We first show, very briefly, that this set of axioms is consistent, by giving an instance in which they are satisfied (in this case, we just use the basic logic of Isabelle/HOL)

instantiation *bool* :: *MinAx*
begin
definition *imp-bool-def* [*iff*]: *imp* = $(\lambda\ \varphi\ \psi.\ \varphi \longrightarrow \psi)$
definition *vdash-bool-def* [*iff*]: $(\vdash \varphi) = \varphi$

instance ⟨*proof*⟩
end

This result may seem trivial, but it is really is fundamental to all minimal logic; we shall use it over and over again.

lemma (**in** *MinAx*) *refl*: $\vdash \varphi \rightarrow \varphi$
⟨*proof*⟩

We next turn to providing some other basic results in minimal logic. Note that *hs* stands for *hypothetical syllogism*.

lemma (in *MinAx*) *weaken*: $\vdash \varphi \Longrightarrow \vdash \psi \to \varphi$
$\langle$*proof*$\rangle$

lemma (in *MinAx*) *hs*: $\vdash \varphi \to \psi \Longrightarrow \vdash \psi \to \chi \Longrightarrow \vdash \varphi \to \chi$
$\langle$*proof*$\rangle$

That concludes our discussion of basic minimal logic. We now turn to developing a rudimentary sequent calculus; the basis of our analysis will be a higher order operation, which translates lists into chains of implication.

primrec (in *MinAx*) *lift-imp* :: *'a list* ⇒ *'a* ⇒ *'a* **(infix** :→ *24*) **where**
  $([] :\to \varphi) = \varphi$
| $((\psi \ \# \ \psi s) :\to \varphi) = (\psi \to (\psi s :\to \varphi))$

As you can see, we use a primitive recursive function in the above definition of *op* :→; we can write this particular lambda abstraction with the shorthand *op* :→. Moreover, we can conceptually we think of this as *foldr op* → ψs φ, in fact this follows from a rather trivial induction:

lemma (in *MinAx*) $(\psi s :\to \varphi) = foldr\ (\%\ \psi\ \varphi.\ \psi \to \varphi)\ \psi s\ \varphi$
$\langle$*proof*$\rangle$

With *op* :→, we now turn to developing some elementary results in the sequent calculus. The first results we find simply correspond to the minimal logic metarules previously established, and also the axioms we have been given. Note that while results in the sequent calculus follow, we first prove stronger theorems in the object language, as this practice typically makes inductive results easier.

abbreviation (in *MinAx*) *lift-vdash* :: *'a list* ⇒ *'a* ⇒ *bool* **(infix** :⊢ *10*) **where**
  $(\Gamma :\vdash \varphi) \equiv (\vdash \Gamma :\to \varphi)$

lemma (in *MinAx*) *lift*: $\vdash \varphi \Longrightarrow \Gamma :\vdash \varphi$
  $\langle$*proof*$\rangle$

lemma (in *MinAx*) *lift-ax2*: $\vdash (\varphi s :\to (\psi \to \chi)) \to (\varphi s :\to \psi) \to (\varphi s :\to \chi)$
  $\langle$*proof*$\rangle$

lemma (in *MinAx*) *lift-mp*: $\Gamma :\vdash \varphi \to \psi \Longrightarrow \Gamma :\vdash \varphi \Longrightarrow \Gamma :\vdash \psi$
$\langle$*proof*$\rangle$

lemma (in *MinAx*) *lift-weaken*: $\Gamma :\vdash \varphi \Longrightarrow \Gamma :\vdash \psi \to \varphi$
$\langle$*proof*$\rangle$

lemma (**in** *MinAx*) *lift-ax1*: $\vdash \varphi \rightarrow (\psi s :\rightarrow \varphi)$
$\langle proof \rangle$

lemma (**in** *MinAx*) *lift-hs*: $\Gamma :\vdash \varphi \rightarrow \psi \Longrightarrow \Gamma :\vdash \psi \rightarrow \chi \Longrightarrow \Gamma :\vdash \varphi \rightarrow \chi$
$\langle proof \rangle$

This theorem is in basic minimal logic, but it is hard to prove without dipping shallowly into the sequent calculus. It will be a gateway to much more general theorems.

lemma (**in** *MinAx*) *flip*: $\vdash (\varphi \rightarrow \psi \rightarrow \chi) \rightarrow (\psi \rightarrow \varphi \rightarrow \chi)$
$\langle proof \rangle$

We next establish two analogues in using sequents

lemma (**in** *MinAx*) *lift-flip1*:
$\vdash (\psi \rightarrow (\psi s :\rightarrow \varphi)) \rightarrow (\psi s :\rightarrow (\psi \rightarrow \varphi))$
$\langle proof \rangle$

lemma (**in** *MinAx*) *lift-flip2*:
$\vdash (\psi s :\rightarrow (\psi \rightarrow \varphi)) \rightarrow (\psi \rightarrow (\psi s :\rightarrow \varphi))$
$\langle proof \rangle$

Next, we give another result in basic minimal logic; we again use some results in sequent calculus to ease proving this result

lemma (**in** *MinAx*) *imp-remove*: $\vdash (\chi \rightarrow \chi \rightarrow \varphi) \rightarrow \chi \rightarrow \varphi$
$\langle proof \rangle$

Our first major theorem in the sequent calculus in minimal logic. As we will see, this is the basis for just about all of the major results

lemma (**in** *MinAx*) *lift-removeAll*[*iff*]:
$\vdash (\psi s :\rightarrow \varphi) \rightarrow ((\mathit{removeAll}\ \chi\ \psi s) :\rightarrow (\chi \rightarrow \varphi))$
$\langle proof \rangle$

We can now prove two expressions of the deduction theorem, and we'll also prove the cut rule:

lemma (**in** *MinAx*) *disch*: $\Gamma :\vdash \varphi \Longrightarrow \mathit{removeAll}\ \psi\ \Gamma :\vdash \psi \rightarrow \varphi$
$\langle proof \rangle$

lemma (**in** *MinAx*) *undisch* [*iff*]: $(\Gamma :\vdash \psi \rightarrow \varphi) = (\psi \# \Gamma :\vdash \varphi)$
$\langle proof \rangle$

lemma (**in** *MinAx*) *cut*:
assumes *a*: $\psi \# \Gamma :\vdash \varphi$

    and b: Γ :⊢ ψ
     shows Γ :⊢ φ
⟨*proof*⟩

The following theorem, as we shall see, gives rise to *monotonicity*, arguably the fundamental theorem of minimal logic. We universally quantify everything to ease the inductive proof, which is somewhat technically challenging even when this trick is employed

lemma (**in** *MinAx*) *imp-mono*:
  $\forall\ \psi s\ \varphi.\ set\ \psi s \subseteq set\ \chi s \longrightarrow (\vdash (\psi s :\to \varphi) \to (\chi s :\to \varphi))$
⟨*proof*⟩

Finally, we can state *monotonicity*...

lemma (**in** *MinAx*) *lift-mono*: $set\ \Gamma \subseteq set\ \Psi \Longrightarrow \Gamma :\vdash \varphi \Longrightarrow \Psi :\vdash \varphi$
⟨*proof*⟩

lemma (**in** *MinAx*) *lift-eq*: $set\ \Gamma = set\ \Psi \Longrightarrow (\Gamma :\vdash \varphi) = (\Psi :\vdash \varphi)$
⟨*proof*⟩

This is now a trivial consequence of our *monotonicity* theorem.

lemma (**in** *MinAx*) *lift-elm*:
  $\varphi \in set\ \Gamma \Longrightarrow \Gamma :\vdash \varphi$
⟨*proof*⟩

A less trivial consequence is the general cut rule...

lemma (**in** *MinAx*) *super-cut*:
   assumes $\forall\ \varphi \in set\ \Delta.\ \Gamma :\vdash \varphi$
     and $\Delta\ @\ \Gamma :\vdash \psi$
    shows $\Gamma :\vdash \psi$
⟨*proof*⟩

end

2 A Classical Logic Axiom Class

theory *ClassAxClass*
imports *MinAxClass*
begin

class *ClassAx* = *MinAx* +
  fixes *bot* :: *'a* $(\bot)$
  assumes *ax3*: $\vdash ((\varphi \to \bot) \to (\psi \to \bot)) \to \psi \to \varphi$

instantiation *bool* :: *ClassAx*
begin
definition *bot-bool-def* [*iff*]: $\bot = False$

instance ⟨*proof*⟩
end

no-notation
Not (¬ - [*40*] *40*)

abbreviation (**in** *ClassAx*)
neg :: 'a ⇒ 'a (¬ - [*40*] *40*) **where**
$\neg\ \varphi \equiv (\varphi \to \bot)$

The following rule is sometimes called *negation elimination* in natural deduction… this is a good name, so we'll name this lemma after that rule.

lemma (**in** *ClassAx*) *neg-elim*: $\vdash \neg\ \varphi \to \varphi \to \psi$
⟨*proof*⟩

We next turn to proving two forms of double negation; the latter is evidently intuitionistically valid while the former is a favorite of classical logicians.

lemma (**in** *ClassAx*) *dblneg1*: $\vdash \neg\ \neg\ \varphi \to \varphi$
⟨*proof*⟩

lemma (**in** *ClassAx*) *dblneg2*: $\vdash \varphi \to \neg\ \neg\ \varphi$
⟨*proof*⟩

Finally, we prove a form of Hilbert's explosion principle, also known as *ex falso quodlibet*

lemma (**in** *ClassAx*) *expls*: $\vdash \bot \to \varphi$
⟨*proof*⟩

We now turn to introducing the shorthand for disjunction and conjunction:

no-notation
op | (**infixr** ∨ *30*)

abbreviation (**in** *ClassAx*)
disj :: 'a ⇒ 'a ⇒ 'a (**infixr** ∨ *30*) **where**
$\varphi \vee \psi \equiv \neg\ \varphi \to \psi$

For the time being, we don't care really about conjunction or bi-implication. We already have effectively proven $\varphi \vee \bot \to \varphi$; we now turn to proving commutativity.

For our own sense of clarity, within the proof we shall use the unabbreviated notation.

lemma (**in** *ClassAx*) *disj-comm*: $\vdash \varphi \vee \psi \rightarrow \psi \vee \varphi$
⟨*proof*⟩

We get to perhaps the most important result of this file now, *disjunction elimination*, which is sometimes known as the *constructive dilemma*.

lemma (**in** *ClassAx*) *disjE*:
$\vdash \varphi \vee \psi \rightarrow (\varphi \rightarrow \chi) \rightarrow (\psi \rightarrow \chi) \rightarrow \chi$
⟨*proof*⟩

lemma (**in** *ClassAx*) *cdil*:
  assumes *a*: $\Gamma :\vdash \varphi \vee \psi$
      and *b*: $\Gamma :\vdash \varphi \rightarrow \chi$
      and *c*: $\Gamma :\vdash \psi \rightarrow \chi$
    shows $\Gamma :\vdash \chi$
⟨*proof*⟩

end

3 A Theory for Manipulating Finite and Infinite Sets, Lists

theory *Set-to-List*
imports *Main Infinite-Set*
begin

This file sets forward two main results. The first is an elementary theory regarding the translation between sets and finite lists. The second is the embedding, via (relatively) injective functions, from finite lists to infinite lists.

We shall begin by giving our theory for converting finite sets to lists via a choice function.

lemma *finite-set-list-ex*:
  assumes *fin*: *finite* (A::$'a$ *set*)
    shows $\exists$ *ls*. *set ls* = A
  ⟨*proof*⟩

lemma *set-of-list-is-finite*:
  finite (*set* Γ)
⟨*proof*⟩

We now give the definition of the our function which converts sets into lists. We should note that since it is a choice function, it is only meaningful in cases in which a list exists. In fact, we will see that our function is meaningful in exactly those cases where our original set is finite. We end with noting that, despite being based on a choice function, it has a definite value for the empty set.

definition *list* :: *'a set* ⇒ *'a list* **where**
list A = (*SOME ls*. *set ls* = *A*)

lemma *set-list*: *finite A* ⟷ (*set* (*list A*) = *A*)
⟨*proof*⟩

lemma *empty-set-list*[*simp*]: *list* {} = []
⟨*proof*⟩

We now turn to showing that if A::'*a* ⇒ *bool* is finite and B::'*b* ⇒ *bool* is infinite, then there exists a function f::'*a* ⇒ '*b* which is injective on A and has its range in B

Rather than prove this from scratch, we will use some library theorems to assist us, namely *finite-imp-nat-seg-image-inj-on* and *infinite-countable-subset*.

However, we evidently need to prove an elementary lemma regarding the relative inverses of functions that are injective on some range.

```
lemma inj-on-inj-off:
  assumes one-one: inj-on f A
    shows ∃ g. inj-on g (f ‘ A)
              ∧ (∀ x ∈ A. x = (g o f) x)
              ∧ (∀ y ∈ (f ‘ A). (f o g) y = y)
⟨proof⟩

lemma fin-inj-on-infi:
  assumes fin-A: finite (A ::'a set)
     and infi-B: infinite (B :: 'b set)
  shows ∃ g::'a ⇒ 'b. inj-on g A ∧ range g ⊆ B
⟨proof⟩

end
```

4 Finitary Lindenbaum Constructions

theory *Little-Lindy*
imports *ClassAxClass Set-to-List*

begin

no-notation (**in** *ClassAx*)
op | (**infixr** ∨ *30*) **and**
Not (¬ - [*40*] *40*)

We first define *pseudo-negation*, which is essential to the finite Lindenbaum construction.

definition (**in** *ClassAx*) *pneg* :: *'a* ⇒ *'a* (∼ - [*40*] *40*) **where**
(∼ φ) = (*if* (∃ ψ. (¬ ψ) = φ) *then* (*SOME* ψ. (¬ ψ) = φ) *else* ¬ φ)

We now turn to proving *tertium non datur* for pseudo negation, as well as logical equivalence with negation.

lemma (**in** *ClassAx*) *pneg-tnd*: ⊢ ∼ φ ∨ φ
⟨*proof*⟩

lemma (**in** *ClassAx*) *pneg-negimpI*: ⊢ ¬ φ → ∼ φ
⟨*proof*⟩

lemma (**in** *ClassAx*) *pneg-negimpII*: ⊢ ∼ φ → ¬ φ
⟨*proof*⟩

The following lemma is critical to the consistency proof of the Lindenbaum construction.

lemma (**in** *ClassAx*) *cnst*:
assumes ♡: ~ (Γ :⊢ ψ)
shows ~ (φ # Γ :⊢ ψ) ∨ ~ ((∼ φ) # Γ :⊢ ψ)
⟨*proof*⟩

We now turn to giving a general, finitistic Lindenbaum construction. The basis for our method is the following observation: finite sets always correspond to some list. Wielding the axiom of choice, we choose a suitable representative list. We then define a primitive recursive function, named with type *'a* ⇒ *'a list* ⇒ *'a list* ⇒ *'a list*, which first takes a formula ψ::*'a*. It then takes a φ::*'a* off the top of the second argument Φ::*'a list* and adds it to the consistently first argument Γ::*'a list* if it may be consistently added without proving ψ, and adds ∼ φ otherwise. The procedure then recurses.

```
primrec (in ClassAx) lind :: 'a ⇒ 'a list ⇒ 'a list ⇒ 'a list where
   lind ψ Γ [] = Γ
 | lind ψ Γ (φ # Φ) = (let φ' = if ~(φ # Γ :⊢ ψ)
                                  then φ
                                  else ∼ φ
                        in (lind ψ (φ' # Γ) Φ))
```

We will show the two crucial properties of this construction: (1) either φ or $\sim \varphi$ are present in the final list for all $\varphi \in \Phi$ and (2) if Γ is does not prove ψ, then the resulting construction does not prove ψ.

We start by proving several basic lemmas, which help us understand the results of a lindenbaum construction. As usual, we frequently use universal quantification in the statement of lemmas to strengthen inductive hypotheses.

lemma **(in** *ClassAx*) *lind-is-mono*:
$\forall\,\Gamma.\ \textit{set}\ \Gamma \subseteq \textit{set}\ (\textit{lind}\ \psi\ \Gamma\ \Phi)$
⟨*proof*⟩

lemma **(in** *ClassAx*) *lind-is-max*:
$\forall\,\Gamma.\ \forall\,\varphi \in \textit{set}\ \Phi.\ \varphi \in \textit{set}\ (\textit{lind}\ \psi\ \Gamma\ \Phi) \vee (\sim \varphi) \in \textit{set}\ (\textit{lind}\ \psi\ \Gamma\ \Phi)$
⟨*proof*⟩

lemma **(in** *ClassAx*) *lind-is-bounded*:
  assumes *pneg-closed*: $(\forall\ \varphi \in \textit{set}\ \Phi.\ (\sim \varphi) \in \textit{set}\ \Phi)$
    shows $\forall\ \Gamma.\ \textit{set}\ (\textit{lind}\ \psi\ \Gamma\ \Phi) \subseteq \textit{set}\ \Gamma \cup \textit{set}\ \Phi$
⟨*proof*⟩

We now turn to perhaps the key lemma regarding Lindenbaum constructions: they preserve consistency!

lemma **(in** *ClassAx*) *lind-is-cnst*:
  $\forall\,\Gamma.\ \tilde{}\ (\Gamma :\vdash \psi) \longrightarrow \tilde{}\ (\textit{lind}\ \psi\ \Gamma\ \Phi :\vdash \psi)$
⟨*proof*⟩

We now give a predicate for atoms, which are maximally consistent sets relative to a finite set Φ. We shall prove that they contain a formula $\varphi \in \Phi$ if and only if they deduce that formula. While we are at it, we shall prove that in the same context, $(\varphi \notin \Gamma) = (\sim \varphi \in \Gamma)$

definition **(in** *ClassAx*)
Atoms :: *'a set* ⇒ *'a set* ⇒ *bool* (*At*) **where**
$\textit{At}\ \Phi\ \Gamma \equiv\ \ \Gamma \subseteq \Phi$
  $\wedge\ (\forall\,\varphi \in \Phi.\ \varphi \in \Gamma \vee (\sim \varphi) \in \Gamma)$
  $\wedge\ \tilde{}(\textit{list}\ \Gamma :\vdash \bot)$

lemma **(in** *ClassAx*) *coincidence*:
assumes *A*: *finite* Φ
  and *B*: $\Gamma \in \textit{At}(\Phi)$
  and *C*: $\varphi \in \Phi$
  and *D*: $P\ (\sim \varphi) = (\tilde{}\ P\ \varphi)$
shows $(\varphi \in \Gamma) = (\textit{list}\ \Gamma :\vdash \varphi)$
  and $P\ \varphi = (\textit{list}\ \Gamma :\vdash \varphi) \Longrightarrow P\ (\sim \varphi) = (\textit{list}\ \Gamma :\vdash \sim \varphi)$

⟨*proof*⟩

We finally turn to presenting the finitary Lindenbaum Lemma. It is in terms of atoms that we shall phrase the primary result we have been leading up to:

lemma (**in** *ClassAx*) *little-lindy*:
  assumes *A*: *finite* Φ
    and *B*: $\forall \varphi \in \Phi$. $(\sim \varphi) \in \Phi$
    and *C*: $\Gamma \subseteq \Phi$
    and *D*: ~(*list* Γ :⊢ ψ)
   shows $\exists\ \Gamma'$. *At* $\Phi\ \Gamma'$
        $\wedge\ \Gamma \subseteq \Gamma'$
        $\wedge$ ~(*list* Γ' :⊢ ψ)

⟨*proof*⟩

end

5 Classic Results in Classical Logic

theory *Classic*
imports *ClassAxClass Little-Lindy*
begin

We first give the grammar for Classical Logic, which is just a simple BNF:

$$\phi ::= p \mid \bot \mid \phi \to \psi$$

Here is the same grammar in Isabelle/HOL; note that its basically the same as the logician's shorthand.

Since we are constantly abusing our notation, we shall first turn off some old notation we had adopted in ClassAxClass, so we can reuse it here.

no-notation
  bot (⊥) **and**
  imp (**infixr** → *25*) **and**
  vdash (⊢ - [*20*] *20*) **and**
  lift-vdash (**infix** :⊢ *10*) **and**
  Not (¬ - [*40*] *40*) **and**
  neg (¬ - [*40*] *40*) **and**
  pneg (~ - [*40*] *40*)

datatype *'a cl-form* =
   CL-P 'a   (*P*#)
 | *CL-Bot*   (⊥)

```
  | CL-Imp 'a cl-form 'a cl-form  (infixr → 25)
```

We next go over the semantics of Classical Logic, which follow a textbook recursive definition.

```
fun cl-eval :: 'a set ⇒ 'a cl-form ⇒ bool (infix ⊨ 20) where
   (S ⊨ P# p) = (p ∈ S)
 | (- ⊨ ⊥) = False
 | (S ⊨ φ → ψ) = ((S ⊨ φ) ⟶ (S ⊨ ψ))

abbreviation
cl-neg :: 'a cl-form ⇒ 'a cl-form (¬ - [40] 40) where
¬ φ ≡ (φ → ⊥)
```

With semantics defined, we turn to defining the syntax of CL, our classical logic, which is the smallest set containing the three axioms of classical logic laid out in ClassAx, and closed under *modus ponens*

```
inductive-set CL :: 'a cl-form set where
  cl-ax1: (φ → ψ → φ) ∈ CL |
  cl-ax2: ((φ → ψ → χ) → (φ → ψ) → (φ → χ)) ∈ CL |
  cl-ax3: (((φ → ⊥) → ψ → ⊥) → ψ → φ) ∈ CL |
  cl-mp: ⟦ (φ → ψ) ∈ CL; φ ∈ CL ⟧ ⟹ ψ ∈ CL

abbreviation cl-vdash :: 'a cl-form ⇒ bool (⊢ - [20] 20) where
(⊢ φ) ≡ φ ∈ CL
```

As per tradition, soundness is trivial:

```
lemma cl-soundness: ⊢ φ ⟹ S ⊨ φ
  ⟨proof⟩
```

Furthermore, This trivially implies that that CL is consistent:

```
lemma cl-const: ~ (⊢ ⊥)
⟨proof⟩
```

The remainder of the current discussion shall be devoted to showing completeness. We first show that our logic is an instance of ClassAx:

```
interpretation cl-ClassAx: ClassAx op → cl-vdash ⊥
⟨proof⟩
```

Next, we define the *Fischer-Ladner* subformula operation, and prove some key lemmas regarding it.

```
primrec FL :: 'a cl-form ⇒ 'a cl-form set where
   FL (P# p) = {P# p, ¬ (P# p), ⊥, ¬ ⊥}
 | FL ⊥ = {⊥, ¬ ⊥}
```

| $\mathit{FL}\ (\varphi \to \psi) = \{\ \varphi \to \psi,\ \neg\ (\varphi \to \psi),$
$\qquad\qquad\varphi,\ \neg\ \varphi,\ \psi,\ \neg\ \psi\ ,\ \bot,\ \neg\ \bot\}$
$\qquad\quad\cup\ \mathit{FL}\ \varphi\ \cup\ \mathit{FL}\ \psi$

lemma *finite-FL*: *finite* $(\mathit{FL}\ \varphi)$
⟨*proof*⟩

lemma *imp-closed-FL*: $(\psi \to \chi) \in \mathit{FL}\ \varphi$
$\qquad\qquad\Longrightarrow \psi \in \mathit{FL}\ \varphi \wedge \chi \in \mathit{FL}\ \varphi$
⟨*proof*⟩

We note define *pseudo-negation* for our classical logic system. Note that we have previously defined *pneg* in developing our classical logic class. Indeed, what we shall define is demonstrated to be the same operation. However, the advantage of our presentation is that it is in fact constructive, which means that it is better for automated reasoning. The advantage of the previous definition is that it is abstract, and so can be used for very general reasoning. But it relies on choice and so apparently does not automate terribly well. . .

fun *dest-neg* :: *'a cl-form* ⇒ *'a cl-form*
where *dest-neg* $(\neg\ \varphi) = \varphi$

abbreviation *cl-pneg* :: *'a cl-form* ⇒ *'a cl-form* (~' - [*40*] *40*)
where
$\sim'\ \varphi \equiv$ (*if* $(\exists\ \psi.\ (\neg\ \psi) = \varphi)$
then (*dest-neg* φ)
else $\neg\ \varphi$)

notation
Classic.cl-ClassAx.pneg (~ - [*40*] *40*)

lemmas *pneg-def* = *Classic.cl-ClassAx.pneg-def*

lemma *cl-pneg-eq*: $(\sim'\ \varphi) = (\sim\ \varphi)$
⟨*proof*⟩

lemma *neg-pneg-sem-eq*: $(\tilde{}\ (S \models \varphi)) = (S \models \sim\ \varphi)$
⟨*proof*⟩

lemma *pneg-FL*: $\forall\,\psi \in \mathit{FL}(\varphi).\ (\sim\ \psi) \in \mathit{FL}(\varphi)$
⟨*proof*⟩

We now turn to showing how Atoms of *FL* Φ can be translated into models. We then show the *Henkin Truth Lemma* for holds for this translation. We will need to set up some more boilerplate to accomplish this (local abbreviations, local names for class theorems, and so on).

notation
Classic.cl-ClassAx.Atoms (*At*) **and**
Classic.cl-ClassAx.lift-imp (**infix** $:\rightarrow$ *24*)

abbreviation *cl-lift-vdash* :: *'a cl-form list* ⇒ *'a cl-form* ⇒ *bool* (**infix** $:\vdash$ *10*)
where
  $(\Gamma :\vdash \varphi) \equiv (\vdash \Gamma :\rightarrow \varphi)$

abbreviation *cl-mod* :: *'a cl-form set* ⇒ *'a set* (†-) **where**
  $\dagger\Gamma \equiv \{p.\ (P\#\ p) \in \Gamma\}$

lemmas
Atoms-def = *Classic.cl-ClassAx.Atoms-def* **and**
coincidence = *Classic.cl-ClassAx.coincidence* **and**
lift = *Classic.cl-ClassAx.lift* **and**
lift-mp = *Classic.cl-ClassAx.lift-mp* **and**
lift-weaken = *Classic.cl-ClassAx.lift-weaken* **and**
pneg-negimpII = *Classic.cl-ClassAx.pneg-negimpII* **and**
neg-elim = *Classic.cl-ClassAx.neg-elim*

lemma *henkin-truth*:
assumes *A*: $\Gamma \in At\ (FL\ \psi)$
  and *B*: $\varphi \in FL(\psi)$
shows $(\dagger\Gamma \vDash \varphi) = (list\ \Gamma :\vdash \varphi)$
  and $(\dagger\Gamma \vDash \sim \varphi) = (list\ \Gamma :\vdash \sim \varphi)$
⟨*proof*⟩

We now turn to our completeness theorem for classical logic

lemmas
little-lindy = *Classic.cl-ClassAx.little-lindy*

lemma *cl-completeness*:
  assumes *dnp*: $\tilde{}\ (\vdash \psi)$
  shows $\exists\ S.\ \tilde{}\ (S \vDash \psi)$
⟨*proof*⟩

lemma *cl-equiv*: $(\vdash \psi) = (\forall S.\ S \vDash \psi)$
⟨*proof*⟩

As an added bonus, since the semantics for classical logic are already essentially automated, we can use them to lazily prove hard things in the proof theory of classical logic automatically. . . as the following demonstrates

lemma *cl-proof* [*intro*!]: $\forall S.\ S \vDash \psi \Longrightarrow\ \vdash \psi$

⟨*proof*⟩

lemma $\vdash ((\psi \to \varphi) \to \psi) \to \psi$
  ⟨*proof*⟩

We'll next turn to setting up a system for importing our theorems from classical logic into the ClassAx class. This will prove extremely useful for our future exploits in formalizing modal logic (since this will mean we will have any classical tautology we can think of at our disposal in proofs).

As a technical note, we are generally agnostic over what proposition letters are in our treatment of classical logic - but here we make a definite interpretation, which is that they are propositions in whatever classical logic we are looking at.

Before we proceed much further, we'll clean up our notation a bit and undo some of our previous abuse (so that we may presumably resume abusing notation in future theories).

no-notation
  cl-vdash (⊢ - [*20*] *20*) **and**
  Classic.cl-ClassAx.Atoms (*At*) **and**
  Classic.cl-ClassAx.lift-imp (**infix** :→ *24*) **and**
  cl-lift-vdash (**infix** :⊢ *10*) **and**
  Classic.cl-ClassAx.pneg (~ - [*40*] *40*) **and**
  cl-pneg (~′ - [*40*] *40*) **and**
  cl-mod (†-)

notation
  bot (⊥) **and**
  imp (**infixr** → *25*) **and**
  vdash (⊢ - [*20*] *20*) **and**
  cl-vdash ($\vdash_{CL}$ - [*20*] *20*) **and**
  lift-vdash (**infix** :⊢ *10*) **and**
  neg (¬ - [*40*] *40*) **and**
  pneg (~ - [*40*] *40*)

primrec (**in** *ClassAx*) *cltr* :: ′*a cl-form* ⇒ ′*a* **where**
    cltr (*P*# *a*) = *a*
  | *cltr* ⊥ = ⊥
  | *cltr* ($\varphi \to \psi$) = ((*cltr* φ) → (*cltr* ψ))

lemma (**in** *ClassAx*) *cl-translate*: $\varphi \in CL \Longrightarrow \vdash$ *cltr* φ
⟨*proof*⟩

end

6 EviL Grammar and Semantics

theory *EviL-Semantics*
imports *Classic*
begin

We now give the grammar and semantics for EviL. We shall be employing two different kinds of semantics for EviL - EviL world sets / EviL world pairs and also conventional Kripke semantics.

```
no-notation
  bot (⊥) and
  imp (infixr → 25)  and
  vdash (⊢ - [20] 20) and
  lift-vdash (infix :⊢ 10) and
  Not  (¬ - [40] 40) and
  neg (¬ - [40] 40) and
  pneg (~ - [40] 40) and
  CL-P (P#) and
  CL-Bot (⊥) and
  CL-Imp (infixr → 25)
```

The datatype below defines a language of a modal logic with a possibly infinite number of agents, which we represent with a *'b*. Informally, we might write this using the following BNF grammar (with some Isabelle style type annotations):

$$\phi \;::=\; \alpha \mid \bot \mid \phi \to \psi \mid \Box_X \phi \mid \odot_X \mid \boxplus_X \phi \mid \boxminus_X \phi$$

```
datatype ('a,'b) evil-form =
    E-P 'a                                   (P# -)
  | E-Bot                                    (⊥)
  | E-PP 'b                                  (⊙)
  | E-Imp ('a,'b) evil-form ('a,'b) evil-form (infixr → 25)
  | E-B 'b ('a,'b) evil-form                 (□)
  | E-BB 'b ('a,'b) evil-form                 ([−])
  | E-BBI 'b ('a,'b) evil-form                ([+])
```

types ('a,'b) *evil-world* = 'a *set* ∗ ('b ⇒ ('a *cl-form set*))

We now turn to giving the recursive, compositional EviL semantic evaluation function. EviL can be understood to rest on the semantics for classical propositional logic we have previously given. This gives EviL a sort of Russian doll semantics, in way.

```
fun evil-eval :: ('a,'b) evil-world set
                  ⇒ ('a,'b) evil-world
                   ⇒ ('a,'b) evil-form
                    ⇒ bool (-,- |⊨ - 50) where
    (-,(a,-) |⊨ P# p) = (p ∈ a)
  | (-,- |⊨ ⊥) = False
  | (Ω,(a,A) |⊨ φ → ψ) =
      ((Ω,(a,A) |⊨ φ) ⟶ (Ω,(a,A) |⊨ ψ))
  | (Ω,(a,A) |⊨ □ X φ) =
      (∀ (b,B) ∈ Ω. (∀ χ ∈ A(X). b ⊨ χ)
                    ⟶  Ω,(b,B) |⊨ φ)
  | (Ω,(a,A) |⊨ ⊙ X) = (∀ χ ∈ A(X). a ⊨ χ)
  | (Ω,(a,A) |⊨ [−] X φ) =
      (∀ (b,B) ∈ Ω. a = b
        ⟶ B(X) ⊆ A(X)
         ⟶  Ω,(b,B) |⊨ φ)
  | (Ω,(a,A) |⊨ [+] X φ) =
      (∀ (b,B) ∈ Ω. a = b
        ⟶ B(X) ⊇ A(X)
         ⟶  Ω,(b,B) |⊨ φ)
```

Here are the Kripke semantics for EviL, which shall be critical for Henkin truth lemmas, Lindenbaum model construction and other model theoretic concerns.

```
record ('w,'a,'b) evil-kripke =
  W :: 'w set
  V :: 'w ⇒ 'a ⇒ bool
  PP :: 'b ⇒ 'w set
  RB :: 'b ⇒ ('w * 'w) set
  RBB :: 'b ⇒ ('w * 'w) set
  RBBI :: 'b ⇒ ('w * 'w) set

fun evil-modal-eval :: ('w,'a,'b) evil-kripke
                        ⇒ 'w
                         ⇒ ('a,'b) evil-form
                          ⇒ bool (-,- |⊢ - 50) where
    (M,w |⊢ P# p) = (p ∈ V(M)(w))
  | (-,- |⊢ ⊥) = False
  | (M,w |⊢ φ → ψ) =
        ((M,w |⊢ φ) ⟶ (M,w |⊢ ψ))
  | (M,w |⊢  □ X φ) =
        (∀ v ∈ W(M). (w,v) ∈ RB(M)(X)
                 ⟶  M,v |⊢ φ)
  | (M,w |⊢ ⊙ X) = (w ∈ PP(M)(X))
  | (M,w |⊢ [−] X φ) =
        (∀ v ∈ W(M). (w,v) ∈ RBB(M)(X)
                 ⟶  M,v |⊢ φ)
  | (M,w |⊢ [+] X φ) =
        (∀ v ∈ W(M). (w,v) ∈ RBBI(M)(X)
                 ⟶  M,v |⊢ φ)
```

end

7 EviL Axiomatics

theory *EviL-Logic*
imports *EviL-Semantics*
begin

In this file, we turn to the task of providing axiomatics for a Hilbert system giving the logic of EviL. We shall follow the treatment in Classic.thy, and instantiate EviL as a Classical Logic. Since we'll continue the business of abusing notation, we first set our notation appropriately.

```
no-notation
  bot (⊥) and
  imp (infixr → 25)  and
  vdash (⊢ - [20] 20) and
  lift-vdash (infix :⊢ 10) and
  lift-imp (infix :→ 24) and
  Not  (¬ - [40] 40) and
  neg (¬ - [40] 40) and
  Classic.cl-neg (¬ - [40] 40) and
  pneg (~ - [40] 40) and
  cl-pneg (~' - [40] 40) and
  CL-P (P#) and
  CL-Bot (⊥) and
  CL-Imp (infixr → 25)
```

```
abbreviation
evil-neg :: ('a,'b) evil-form ⇒ ('a,'b) evil-form (¬ - [40] 40) where
¬ φ ≡ (φ → ⊥)
```

```
abbreviation
evil-D :: 'b ⇒ ('a,'b) evil-form ⇒ ('a,'b) evil-form (◇) where
◇ X φ ≡ ¬ (□ X (¬ φ))
```

abbreviation
evil-DD :: $'b \Rightarrow ('a,'b)$ *evil-form* $\Rightarrow ('a,'b)$ *evil-form* $(\langle+\rangle)$ **where**
$\langle+\rangle\ X\ \varphi \equiv \neg\ ([+]\ X\ (\neg\ \varphi))$

abbreviation
evil-DDI :: $'b \Rightarrow ('a,'b)$ *evil-form* $\Rightarrow ('a,'b)$ *evil-form* $(\langle-\rangle)$ **where**
$\langle-\rangle\ X\ \varphi \equiv \neg\ ([-]\ X\ (\neg\ \varphi))$

Here are the axioms of EviL; since these principles have their basis in philosophy, we offer philosophical readings of each.

inductive-set *EviL* :: $('a,'b)$ *evil-form set* **where**
— If something is true, nothing can change this
evil-ax1: $(\varphi \to \psi \to \varphi) \in$ *EviL* |

— If φ and ψ jointly imply χ,
— and φ implies ψ,
— then φ alone is sufficient too show χ
evil-ax2: $((\varphi \to \psi \to \chi) \to (\varphi \to \psi) \to (\varphi \to \chi)) \in$ *EviL* |

— If the failure of φ ensures the failure of ψ,
— then ψ's success ensures φ's success.
evil-ax3: $((\neg\ \varphi\ \ \to \neg\ \psi) \to \psi \to \varphi) \in$ *EviL* |

— If under any further evidence X considers, φ holds,
— then φ holds simpliciter,
— since considering no additional evidence is trivially considering further evidence

evil-ax4: $([+]\ X\ \varphi \to \varphi) \in$ *EviL* |

— If under any further evidence X considers, φ holds,
— then φ also holds whenever X considers further further evidence.
evil-ax5: $(([+]\ X\ \varphi) \to ([+]\ X\ ([+]\ X\ \varphi))) \in$ *EviL* |

— Changing one's mind does not effect matters of fact
evil-ax6: $(P\#\ p \to [+]\ X\ (P\#\ p)) \in$ *EviL* |
evil-ax7: $(P\#\ p \to [-]\ X\ (P\#\ p)) \in$ *EviL* |

— The more evidence X discards,
— the freer her imagination becomes.
evil-ax8: $((\Diamond\ X\ \varphi) \to [-]\ X\ (\Diamond\ X\ \varphi)) \in$ *EviL* |

— If X believes φ,
— she believes it despite what anyone thinks
evil-ax9: $((\Box\ X\ \varphi) \to \Box\ X\ ([+]\ Y\ \varphi)) \in$ *EviL* |
evil-ax10: $((\Box\ X\ \varphi) \to \Box\ X\ ([-]\ Y\ \varphi)) \in$ *EviL* |

— If X's evidence is sound,
— then what she believes is true
evil-ax11: $(\odot\ X \to (\Box\ X\ \varphi) \to \varphi) \in$ *EviL* |

— If X's evidence is sound,
— then any subset of it she can consider must be sound too
evil-ax12: $(\odot\ X \to [-]\ X\ (\odot\ X)) \in$ *EviL* |

— If φ is true,
— then no matter what further evidence X considers,
— she can forget it and φ will still be true
evil-ax13: $(\varphi \to [+]\ X\ (\langle - \rangle\ X\ \varphi)) \in$ *EviL* |

— If φ is true,
— then no matter what evidence X dispenses with,
— if X remembers correctly then φ will still be true
evil-ax14: $(\varphi \to [-]\ X\ (\langle + \rangle\ \ X\ \varphi)) \in$ *EviL* |

— If X believes φ implies ψ and φ
— on the basis of her evidence, she can come to believe ψ
— on this same basis of her evidence.
evil-ax15: $((\Box\ X\ (\varphi \to \psi)) \to (\Box\ X\ \varphi) \to \Box\ X\ \psi) \in$ *EviL* |

— If no matter what evidence X tries to forget,
— φ implies ψ, and also φ holds,
— then no matter what evidence she disregards it must be that ψ.
evil-ax16: $(([-]\ X\ (\varphi \to \psi)) \to ([-]\ X\ \varphi) \to [-]\ X\ \psi) \in$ *EviL* |

— If no matter what further evidence X considers,
— φ implies ψ, and also φ holds,
— then no matter what further evidence she consider it must be that ψ.
evil-ax17: $(([+]\ X\ (\varphi \to \psi)) \to ([+]\ X\ \varphi) \to [+]\ X\ \psi) \in$ *EviL* |

— If something is always true, then an agent can come to believe this
evil-B-nec: $\varphi \in$ *EviL* $\Longrightarrow (\Box\ X\ \varphi) \in$ *EviL* |

— If something is always true,
— then it's true no matter what an agent tries to forget
evil-BB-nec: $\varphi \in$ *EviL* $\Longrightarrow ([-]\ X\ \varphi) \in$ *EviL* |

— If something is always true,
— then it's true regardless of what more an agent might choose to believe
evil-BBI-nec: $\varphi \in$ *EviL* $\Longrightarrow ([+]\ X\ \varphi) \in$ *EviL* |

— Modus ponens
evil-mp: $[\![\ (\varphi \to \psi) \in \mathit{EviL};\ \varphi \in \mathit{EviL}\ ]\!] \Longrightarrow \psi \in \mathit{EviL}$

abbreviation *evil-vdash* :: $('a,'b)$ *evil-form* $\Rightarrow$ *bool* $(\vdash$ - $[20]\ 20)$ **where**
$(\vdash \varphi) \equiv \varphi \in \mathit{EviL}$

It's natural to want to prove soundness after introducing all of these axioms. The proof is completely mechanical:

theorem *evil-soundness*: $\vdash \varphi \Longrightarrow \forall\ (a,A) \in \Omega.\ \Omega,(a,A) \models \varphi$
  ⟨*proof*⟩

theorem *evil-consistency*: $\tilde{}\ (\vdash \bot)$
⟨*proof*⟩

We now turn to developing some basic proof theory for EviL. We start by showing that it is an extesion of classical logic; it is in fact a conservative extension (we assert this without proof). So we shall establish that it is an instance of ClassAx.

interpretation *evil-ClassAx*: *ClassAx op* $\to$ *evil-vdash* $\bot$
⟨*proof*⟩

In the subsequent discussion, we'll have need to prove a lot of theorems in classical propositional logic; our basic approach will be to appeal to completeness and apply automation to accomplish this. So we now reintroduce syntax for classical logic.

notation
CL-P $(P\#_{CL})$ **and**
CL-Bot $(\bot_{CL})$ **and**
cl-neg $(\neg_{CL})$ **and**
CL-Imp (**infixr** $\to_{CL}$ *25*)

Our first application of this approach will be to prove a rewrite rule for EviL; we shall have intend to appeal to rewriting further on in our proof

primrec *evil-sub* ::
 $[('a,'b)$ *evil-form*, $('a,'b)$ *evil-form*, $('a,'b)$ *evil-form*$]$
   $\Rightarrow ('a,'b)$ *evil-form* $(\text{-}[\text{-}'/\text{-}]\ [300,\ 0,\ 0]\ 300)$ **where**
  $(P\#\ a)[\varphi/\psi] = (\mathit{if}\ ((P\#\ a) = \varphi)\ \mathit{then}\ \psi\ \mathit{else}\ (P\#\ a))$
 $\mid \bot[\varphi/\psi] = (\mathit{if}\ (\bot = \varphi)\ \mathit{then}\ \psi\ \mathit{else}\ \bot)$
 $\mid (\odot\ X)[\varphi/\psi] = (\mathit{if}\ ((\odot\ X) = \varphi)\ \mathit{then}\ \psi\ \mathit{else}\ (\odot\ X))$
 $\mid (\delta \to \kappa)[\varphi/\psi] = (\mathit{if}\ ((\delta \to \kappa) = \varphi)\ \mathit{then}\ \psi$
      $\mathit{else}\ (\delta[\varphi/\psi] \to \kappa[\varphi/\psi]))$
 $\mid (\Box\ X\ \kappa)[\varphi/\psi] = (\mathit{if}\ ((\Box\ X\ \kappa) = \varphi)\ \mathit{then}\ \psi$
      $\mathit{else}\ (\Box\ X\ (\kappa[\varphi/\psi])))$

  | $([-]\ X\ \kappa)[\varphi/\psi] = (\mathit{if}\ (([-]\ X\ \kappa) = \varphi)\ \mathit{then}\ \psi$
    $\mathit{else}\ ([-]\ X\ (\kappa[\varphi/\psi])))$
  | $([+]\ X\ \kappa)[\varphi/\psi] = (\mathit{if}\ (([+]\ X\ \kappa) = \varphi)\ \mathit{then}\ \psi$
    $\mathit{else}\ ([+]\ X\ (\kappa[\varphi/\psi])))$

abbreviation *evil-iff* ::
  $[('a,'b)\ \mathit{evil\text{-}form},\ ('a,'b)\ \mathit{evil\text{-}form}]$
   $\Rightarrow ('a,'b)\ \mathit{evil\text{-}form}$ (**infixr** $\leftrightarrow$ *25*) **where**
$(\varphi \leftrightarrow \psi) \equiv ((\varphi \to \neg\ \psi) \to \neg(\neg\ \varphi \to \psi))$

abbreviation *cl-iff* ::
  $['a\ \mathit{cl\text{-}form},\ 'a\ \mathit{cl\text{-}form}] \Rightarrow\ 'a\ \mathit{cl\text{-}form}$ (**infixr** $\leftrightarrow_{CL}$ *25*) **where**
$(\varphi \leftrightarrow_{CL} \psi) \equiv ((\varphi \to_{CL} \neg_{CL}\ \psi) \to_{CL} \neg_{CL}\ (\neg_{CL}\ \varphi \to_{CL} \psi))$

As the following shows, most elementary theorems about logical equivalence reflect tautologies from classical propositional logic; having automated semantics and completeness makes this work rather straightforward.

lemma *evil-eq-refl*: $\vdash \varphi \leftrightarrow \varphi$
⟨*proof*⟩

lemma *evil-eq-symm* [*sym*]: $\vdash \varphi \leftrightarrow \psi \Longrightarrow \vdash \psi \leftrightarrow \varphi$
⟨*proof*⟩

lemma *evil-eq-trans*:
  $\vdash \varphi \leftrightarrow \psi \Longrightarrow \vdash \psi \leftrightarrow \chi \Longrightarrow \vdash \varphi \leftrightarrow \chi$
⟨*proof*⟩

One should note that the above three lemmas establish that *op* $\leftrightarrow$ is an equivalence relation, which is of course an elementary result in basic logic.

lemma *evil-eq-weaken*: $\vdash \varphi \leftrightarrow \psi \Longrightarrow \vdash \varphi \to \psi$
⟨*proof*⟩

lemma *evil-eq-mp*: $\vdash \varphi \leftrightarrow \psi \Longrightarrow \vdash \varphi \Longrightarrow \vdash \psi$
⟨*proof*⟩

lemma *evil-eq-intro*: $\vdash \varphi \to \psi \Longrightarrow \vdash \psi \to \varphi \Longrightarrow \vdash \varphi \leftrightarrow \psi$
⟨*proof*⟩

lemma *evil-contrapose*:
  $\vdash \varphi \to \psi \Longrightarrow \vdash \neg\ \psi \to \neg\ \varphi$
⟨*proof*⟩

notation
  evil-ClassAx.lift-imp (**infix** :→ *24*)

abbreviation *evil-lift-vdash* ::
  $('a,'b)$ *evil-form list*
   $\Rightarrow ('a,'b)$ *evil-form* $\Rightarrow$ *bool* (**infix** $:\vdash$ *10*) **where**
 $(\Gamma :\vdash \varphi) \equiv (\vdash \Gamma :\rightarrow \varphi)$

lemma *evil-B-map*: $\vdash \varphi \rightarrow \psi \Longrightarrow \vdash \Box\ X\ \varphi \rightarrow \Box\ X\ \psi$
⟨*proof*⟩

lemma *evil-lift-ax15*:
  assumes *notnil*: $\varphi s \neq []$
   shows $\vdash \Box\ X\ (\varphi s :\rightarrow \psi)$
        $\rightarrow ((map\ (\lambda\ \varphi.\ \Box\ X\ \varphi)\ \varphi s) :\rightarrow \Box\ X\ \psi)$
⟨*proof*⟩

lemma *evil-B-lift-map*:
 assumes *seq*: $\varphi s :\vdash \psi$
  shows $(map\ (\lambda\ \varphi.\ \Box\ X\ \varphi)\ \varphi s) :\vdash \Box\ X\ \psi$
⟨*proof*⟩

lemma *evil-DB-map*: $\vdash \varphi \rightarrow \psi \Longrightarrow \vdash \Diamond\ X\ \varphi \rightarrow \Diamond\ X\ \psi$
⟨*proof*⟩

lemma *evil-BB-map*: $\vdash \varphi \rightarrow \psi \Longrightarrow \vdash [-]\ X\ \varphi \rightarrow [-]\ X\ \psi$
⟨*proof*⟩

lemma *evil-lift-ax16*:
  assumes *notnil*: $\varphi s \neq []$
   shows $\vdash [-]\ X\ (\varphi s :\rightarrow \psi)$
        $\rightarrow ((map\ (\lambda\ \varphi.\ [-]\ X\ \varphi)\ \varphi s) :\rightarrow [-]\ X\ \psi)$
⟨*proof*⟩

lemma *evil-BB-lift-map*:
 assumes *seq*: $\varphi s :\vdash \psi$
  shows $(map\ (\lambda\ \varphi.\ [-]\ X\ \varphi)\ \varphi s) :\vdash [-]\ X\ \psi$
⟨*proof*⟩

lemma *evil-DBB-map*: $\vdash \varphi \rightarrow \psi \Longrightarrow \vdash \langle - \rangle\ X\ \varphi \rightarrow \langle - \rangle\ X\ \psi$
⟨*proof*⟩

lemma *evil-BBI-map*: $\vdash \varphi \rightarrow \psi \Longrightarrow \vdash [+]\ X\ \varphi \rightarrow [+]\ X\ \psi$
⟨*proof*⟩

lemma *evil-lift-ax17*:
  assumes *notnil*: $\varphi s \neq []$

shows $\vdash$ [+] X (φs :→ ψ)
→ ((*map* (λ φ. [+] X φ) φs) :→ [+] X ψ)
⟨*proof*⟩

lemma *evil-BBI-lift-map*:
assumes *seq*: φs :⊢ ψ
shows (*map* (λ φ. [+] X φ) φs) :⊢ [+] X ψ
⟨*proof*⟩

lemma *evil-DBBI-map*: $\vdash \varphi \to \psi \Longrightarrow \vdash \langle + \rangle\ X\ \varphi \to \langle + \rangle\ X\ \psi$
⟨*proof*⟩

lemma *evil-sub*:
assumes *eq*: $\vdash \varphi \leftrightarrow \psi$
shows $\vdash \chi \leftrightarrow \chi[\varphi/\psi]$
⟨*proof*⟩

The substitution theorem above, while popular in the literature, is not rigorous. Since it relies on pattern matching, authors play faster and looser with it than other tasks.

However, we can show that substitution never changes proper subformulae of the thing being substituted. In every instance of substitution we shall employ, this fact is what suffices to make substitution really applicable.

— A little function which gives the proper subforumulae
primrec *evil-psubforms*
:: ('*a*,'*b*) *evil-form* ⇒ ('*a*,'*b*) *evil-form set* (⇓)
where
⇓($P\#$ p) = {}
| ⇓(⊥) = {}
| ⇓(⊙ X) = {}
| ⇓($\varphi \to \psi$) = {φ, ψ} ∪ ⇓(φ) ∪ ⇓(ψ)
| ⇓(□ X φ) = {φ} ∪ ⇓(φ)
| ⇓([−] X φ) = {φ} ∪ ⇓(φ)
| ⇓([+] X φ) = {φ} ∪ ⇓(φ)

— Here's a series of obvious inequalities we shall reuse
lemma *evil-limp-neq*[*intro*]: ∀ χ. ($\psi \to \chi$) ≠ ψ
⟨*proof*⟩

lemma *evil-rimp-neq*[*intro*]: ∀ ψ. ($\psi \to \chi$) ≠ χ
⟨*proof*⟩

lemma *evil-B-neq*[*intro*]: (□ X φ) ≠ φ
⟨*proof*⟩

lemma *evil-BB-neq*[*intro*]: $([-]\ X\ \varphi) \neq \varphi$
⟨*proof*⟩

lemma *evil-BBI-neq*[*intro*]: $([+]\ X\ \varphi) \neq \varphi$
⟨*proof*⟩

lemma *evil-not-neq*[*intro*]: $(\neg\ \varphi) \neq \varphi$
⟨*proof*⟩
lemma *evil-psform-limp-elim*[*intro*]:
$(\delta \to \kappa) \in \Downarrow \psi \Longrightarrow \delta \in \Downarrow \psi$
⟨*proof*⟩

lemma *evil-psform-rimp-elim*[*intro*]:
$(\delta \to \kappa) \in \Downarrow \psi \Longrightarrow \kappa \in \Downarrow \psi$
⟨*proof*⟩

lemma *evil-psform-B-elim*[*intro*]:
$\square\ X\ \psi \in \Downarrow \varphi \Longrightarrow \psi \in \Downarrow \varphi$
⟨*proof*⟩

lemma *evil-psform-BB-elim*[*intro*]:
$[-]\ X\ \psi \in \Downarrow \varphi \Longrightarrow \psi \in \Downarrow \varphi$
⟨*proof*⟩

lemma *evil-psform-BBI-elim*[*intro*]:
$[+]\ X\ \psi \in \Downarrow \varphi \Longrightarrow \psi \in \Downarrow \varphi$
⟨*proof*⟩
lemma *evil-psform-nin* [*intro*!]: $\varphi \notin \Downarrow \varphi$
⟨*proof*⟩

lemma *sub-neq* [*intro*!]:
assumes *sf*: $\psi \in \Downarrow \varphi$
shows $\psi \neq \varphi$
⟨*proof*⟩

lemma *sub-nosub* [*intro*]:
assumes *psub*: $\psi \in \Downarrow \varphi$
shows $\psi[\varphi/\chi] = \psi$
⟨*proof*⟩

lemma *evil-dneg-eq*: $\vdash \neg\ (\neg\ \varphi) \leftrightarrow \varphi$
⟨*proof*⟩

After showing all of the above, we have what we need to formalize our reasoning about EviL; specifically, we prove versions of axioms 13 and 14, an analogue of axiom 8 for [+] X, and analogues of axioms 4 and 5 for [−] X.

lemma *evil-dax13*: $\vdash \langle + \rangle\ X\ ([-]\ X\ \varphi) \rightarrow \varphi$
⟨*proof*⟩

lemma *evil-dax14*: $\vdash \langle - \rangle\ X\ ([+]\ X\ \varphi) \rightarrow \varphi$
⟨*proof*⟩

lemma *evil-BBIax8*: $\vdash (\Box\ X\ \varphi) \rightarrow [+]\ X\ (\Box\ X\ \varphi)$
⟨*proof*⟩

lemma *evil-BBax4*: $\vdash [-]\ X\ \varphi \rightarrow \varphi$
— If φ holds no matter what X tries to forget,
— then it must be that φ holds simpliciter
⟨*proof*⟩

lemma *evil-BBdax5*: $\vdash \langle - \rangle\ X\ (\langle - \rangle\ X\ \varphi) \rightarrow \langle - \rangle\ X\ \varphi$
— If φ is true no matter what X
— tries to forget, then it's true no matter
— what further evidence she disregards
⟨*proof*⟩

lemma *evil-BBax5*: $\vdash [-]\ X\ \varphi \rightarrow [-]\ X\ ([-]\ X\ \varphi)$
— If φ is true no matter what X
— tries to forget, then it's true no matter
— what further evidence she disregards
⟨*proof*⟩

end

8 Locales for EviL Properties

theory *EviL-Properties*
imports *EviL-Semantics*
begin

In this file we define two locales on EviL Kripke models, which we will be critical for proving the *column lemmas* and ultimately the *translation lemma.*

The first locale will assume properties which we shall prove our Lindenbaum construction satisfies.

locale *partly-EviL* =

```
  fixes M :: ('w,'a,'b) evil-kripke
  assumes prop0: RBB(M)(X) ⊆ (W(M) <*> W(M))
      and prop1: finite (W(M))
      and prop2: refl-on (W(M)) (RBB(M)(X))
      and prop3: trans (RBB(M)(X))
      and prop4: RBBI(M)(X) = (RBB(M)(X))^-1
      and prop5: (w,v) ∈ RBB(M)(X) ⟹ V(M)(w) = V(M)(v)
      and prop6: [[(w,v) ∈ RBB(M)(X); (w,u) ∈ RB(M)(X)]]
                 ⟹ (v,u) ∈ RB(M)(X)
      and prop7: (w,v) ∈ RBB(M)(X)
                 ⟹ ((u,w) ∈ RB(M)(Y)) = ((u,v) ∈ RB(M)(Y))
      and prop8: w ∈ PP(M)(X) ⟹ (w,w) ∈ RB(M)(X)
```

Our second locale strengthens the final 8th property of the first locale to a full biconditional; the *EviL bisimulation lemma* will establish that any *partly EviL* Kripke model is bisimular to a *completely EviL* Kripke model.

```
locale completely-EviL = partly-EviL +
  assumes prop9: (w ∈ PP(M)(X)) = ((w,w) ∈ RB(M)(X))

end
```

9 The EviL Truth (Lemma)

```
theory EviL-Truth
imports EviL-Logic
begin
```

In our previous treatment, we introduced the semantics, proof theory, soundness and completeness for classical logic in one file; addressing the issues related to the canonical model construction for classical logic along with everything else. Since the logic we are developing here is much richer, we have opted to devote this file to the truth lemma for the subformula model we have constructed.

```
no-notation
  bot (⊥) and
  imp (infixr → 25)  and
  vdash (⊢ - [20] 20) and
  lift-vdash (infix :⊢ 10) and
  Not  (¬ - [40] 40) and
  neg (¬ - [40] 40) and
  Classic.cl-neg (¬ - [40] 40) and
  pneg (~ - [40] 40) and
  cl-pneg (~' - [40] 40) and
```

CL-P ($P\#$) **and**
CL-Bot ($\bot$) **and**
CL-Imp (**infixr** $\rightarrow$ *25*)

We first introduce *pseudo* operators. Namely, we'll follow our previous treatment of pseudo-negation (that is, *Not*) that we did in `Classic.thy`, but we shall also introduce new psuedo-operations corresponding to $[-]$ and $[+]$. To do this, we first prove some basic logical equivilances, which are consequences of the above.

lemma *evil-BBI-eq*: $\vdash [+]\ X\ ([+]\ X\ \varphi) \leftrightarrow [+]\ X\ \varphi$
— Further further beliefs are the same as further beliefs
⟨*proof*⟩

lemma *evil-BB-eq*: $\vdash [-]\ X\ ([-]\ X\ \varphi) \leftrightarrow [-]\ X\ \varphi$
— To discard beliefs and then discard beliefs again
— is the same as discarding beliefs only once
⟨*proof*⟩

lemma *evil-eq-neg*: $\vdash\ \ \varphi \leftrightarrow \psi \Longrightarrow \vdash \neg\ \varphi \leftrightarrow \neg\ \psi$
⟨*proof*⟩

lemma *evil-DD-eq*: $\vdash \langle - \rangle\ X\ (\langle - \rangle\ X\ \varphi) \leftrightarrow \langle - \rangle\ X\ \varphi$
⟨*proof*⟩

lemma *evil-DDI-eq*: $\vdash \langle + \rangle\ X\ (\langle + \rangle\ X\ \varphi) \leftrightarrow \langle + \rangle\ X\ \varphi$
⟨*proof*⟩

Here are our psuedo box operators; the lemmas we shall prove reflect the lemmas associated with pseudo-negation.

definition *evil-pBB* :: $'b \Rightarrow ('a,'b)$ *evil-form*
$\Rightarrow ('a,'b)$ *evil-form* $([-]')$
where
$[-]'\ X\ \varphi \equiv ($*if* $(\exists\ \psi.\ ([-]\ X\ \psi) = \varphi)$
then φ
else $[-]\ X\ \varphi)$

definition *evil-pBBI* :: $'b \Rightarrow ('a,'b)$ *evil-form*
$\Rightarrow ('a,'b)$ *evil-form* $([+]')$
where
$[+]'\ X\ \varphi \equiv ($*if* $(\exists\ \psi.\ ([+]\ X\ \psi) = \varphi)$
then φ
else $[+]\ X\ \varphi)$

abbreviation *evil-pDD* :: $'b \Rightarrow ('a,'b)$ *evil-form*

$\Rightarrow$ $('a,'b)$ *evil-form* $(\langle - \rangle')$
where
$\langle - \rangle'\ X\ \varphi \equiv \neg\ ([-]'\ X\ (\neg\ \varphi))$

abbreviation *evil-pDDI* :: $'b \Rightarrow ('a,'b)$ *evil-form*
$\Rightarrow ('a,'b)$ *evil-form* $(\langle + \rangle')$
where
$\langle + \rangle'\ X\ \varphi \equiv \neg\ ([+]'\ X\ (\neg\ \varphi))$

declare *evil-pBB-def* [*simp*]
and *evil-pBBI-def* [*simp*]

To start, we shall prove some basic syntactic theorems regarding our new operators.

lemma *pBB-eq* [*simp*]: $[-]'\ X\ ([-]'\ X\ \varphi) = [-]'\ X\ \varphi$ ⟨*proof*⟩
lemma *pBBI-eq* [*simp*]: $[+]'\ X\ ([+]'\ X\ \varphi) = [+]'\ X\ \varphi$ ⟨*proof*⟩

lemma *pBB-BB-subform-sub*: $\Downarrow ([-]'\ X\ \varphi) \subseteq \Downarrow ([-]\ X\ \varphi)$
⟨*proof*⟩

lemma *pBBI-BBI-subform-sub*: $\Downarrow ([+]'\ X\ \varphi) \subseteq \Downarrow ([+]\ X\ \varphi)$
⟨*proof*⟩

We have here now two utterly analogous proofs, illustrating our psuedo-operations are algebraically indistinguishable to the logic of EviL.

lemma *evil-BB-pBB-eq*: $\vdash [-]'\ X\ \varphi \leftrightarrow [-]\ X\ \varphi$
⟨*proof*⟩

lemma *evil-BBI-pBBI-eq*: $\vdash [+]'\ X\ \varphi \leftrightarrow [+]\ X\ \varphi$
⟨*proof*⟩

lemma *evil-eq-contrapose*:
$\vdash \varphi \leftrightarrow \psi \Longrightarrow \vdash \neg\ \varphi \leftrightarrow \neg\ \psi$
⟨*proof*⟩

lemma *evil-DD-pDD-eq*: $\vdash \langle - \rangle'\ X\ \varphi \leftrightarrow \langle - \rangle\ X\ \varphi$
⟨*proof*⟩

lemma *evil-DDI-pDDI-eq*: $\vdash \langle + \rangle'\ X\ \varphi \leftrightarrow \langle + \rangle\ X\ \varphi$
⟨*proof*⟩

Some consequences of the above are that every axiom involving $[-]$ and $[+]$ has a variation involving the pseudo-boxes.

This constitutes a metalemma of sorts.

lemma *evil-pax4*: $\vdash [+]'\ X\ \varphi \rightarrow \varphi$
⟨*proof*⟩

lemma *evil-pBBax4*: $\vdash [-]'\ X\ \varphi \rightarrow \varphi$
⟨*proof*⟩

lemma *evil-pax5*: $\vdash [+]'\ X\ \varphi \rightarrow [+]'\ X\ ([+]'\ X\ \varphi)$
⟨*proof*⟩

lemma *evil-pBBax5*: $\vdash [-]'\ X\ \varphi \rightarrow [-]'\ X\ ([-]'\ X\ \varphi)$
⟨*proof*⟩

lemma *evil-pax6*: $\vdash P\#\ p \rightarrow [+]'\ X\ (P\#\ p)$
⟨*proof*⟩

lemma *evil-pax7*: $\vdash P\#\ p \rightarrow [-]'\ X\ (P\#\ p)$
⟨*proof*⟩

lemma *evil-pax8*: $\vdash \Diamond\ X\ \varphi \rightarrow [-]'\ X\ (\Diamond\ X\ \varphi)$
⟨*proof*⟩

lemma *evil-pBBIax8*: $\vdash \Box\ X\ \varphi \rightarrow [+]'\ X\ (\Box\ X\ \varphi)$
⟨*proof*⟩

lemma *evil-pax9*: $\vdash \Box\ X\ \varphi \rightarrow \Box\ X\ ([+]'\ Y\ \varphi)$
⟨*proof*⟩

lemma *evil-pax10*: $\vdash \Box\ X\ \varphi \rightarrow \Box\ X\ ([-]'\ Y\ \varphi)$
⟨*proof*⟩

lemma *evil-pax12*: $\vdash \odot\ X \rightarrow [-]'\ X\ (\odot\ X)$
⟨*proof*⟩

lemma *evil-pax13*: $\vdash \varphi \rightarrow [+]'\ X\ (\langle - \rangle'\ X\ \varphi)$
⟨*proof*⟩

lemma *evil-pax14*: $\vdash \varphi \rightarrow [-]'\ X\ (\langle + \rangle'\ X\ \varphi)$
⟨*proof*⟩

lemma *evil-pax16*: $\vdash [-]'\ X\ (\varphi \rightarrow \psi) \rightarrow [-]'\ X\ \varphi \rightarrow [-]'\ X\ \psi$
⟨*proof*⟩

lemma *evil-pax17*: $\vdash [+]'\ X\ (\varphi \rightarrow \psi) \rightarrow [+]'\ X\ \varphi \rightarrow [+]'\ X\ \psi$
⟨*proof*⟩

```
lemma evil-pBB-nec:
      assumes prv: ⊢ φ
        shows ⊢ [−]′ X φ
⟨proof⟩

lemma evil-pBBI-nec:
      assumes prv: ⊢ φ
        shows ⊢ [+]′ X φ
⟨proof⟩

lemma evil-lift-pax16:
   assumes notnil: φs ≠ []
     shows ⊢ [−]′ X (φs :→ ψ)
            → ((map ([−]′ X) φs) :→ [−]′ X ψ)
⟨proof⟩

lemma evil-pBB-lift-map:
 assumes seq: φs :⊢ ψ
   shows (map ([−]′ X) φs) :⊢ [−]′ X ψ
⟨proof⟩

lemma evil-lift-pax17:
   assumes notnil: φs ≠ []
     shows ⊢ [+]′ X (φs :→ ψ)
            → ((map ([+]′ X) φs) :→ [+]′ X ψ)
⟨proof⟩

lemma evil-pBBI-lift-map:
 assumes seq: φs :⊢ ψ
   shows (map ([+]′ X) φs) :⊢ [+]′ X ψ
⟨proof⟩
```

What follows is mostly repeat code from `Classic.thy`; however, we also show logical results which are analogous to the above.

One change is that our destructor is total now; we shall find a crazy occasion to reuse it in a lemma.

```
primrec evil-dest:: (′a,′b) evil-form
    ⇒ (′a,′b) evil-form (√)
  where  √ (P# p) = P# p
       | √ ⊥ = ⊥
       | √ (⊙ X) = ⊙ X
       | √ (φ → ψ) = φ
       | √ (□ X φ) = φ
       | √ ([−] X φ) = φ
```

| $\surd$ ([+] X φ) = φ

abbreviation *evil-pneg* :: ($'a,'b$) *evil-form*
  $\Rightarrow$ ($'a,'b$) *evil-form* ($\sim'$ - [*40*] *40*)
where
$\sim'$ φ ≡ (*if* (∃ ψ. (¬ ψ) = φ)
  then ($\surd$ φ)
  else ¬ φ)

notation
evil-ClassAx.pneg ($\sim$ - [*40*] *40*)

lemmas *pneg-def* = *evil-ClassAx.pneg-def*

— The new pseudo-negation is constructive(?) so always simplify to it
lemma *pneg-eq* [*simp*]: ($\sim$ φ) = ($\sim'$ φ)
⟨*proof*⟩
lemma *evil-pBB-pneg-eq* [*simp*]: ($\sim$ [−]$'$ X φ) = (¬ [−]$'$ X φ)
⟨*proof*⟩

lemma *evil-pBBI-pneg-eq* [*simp*]: ($\sim$ [+]$'$ X φ) = (¬ [+]$'$ X φ)
⟨*proof*⟩

lemma *evil-B-pneg-eq* [*simp*]: ($\sim$ □ X φ) = (¬ □ X φ)
⟨*proof*⟩

lemma *evil-pBB-pneg-eq2* [*simp*]: ($\sim$ ¬ [−]$'$ X φ) = ([−]$'$ X φ)
⟨*proof*⟩

lemma *evil-pBBI-pneg-eq2* [*simp*]: ($\sim$ ¬ [+]$'$ X φ) = ([+]$'$ X φ)
⟨*proof*⟩

lemma *evil-B-pneg-eq2* [*simp*]: ($\sim$ ¬ □ X φ) = (□ X φ)
⟨*proof*⟩

lemma *evil-Bot-pneg-eq* [*simp*]: ($\sim$ ⊥) = (¬ ⊥) ⟨*proof*⟩

lemma *evil-Bot-pneg-eq2* [*simp*]: ($\sim$ (¬ ⊥)) = ⊥ ⟨*proof*⟩
lemma *evil-pneg-eq*: ⊢ $\sim$ φ ↔ ¬ φ
⟨*proof*⟩

lemma *evil-pdneg-eq*: ⊢ ¬ $\sim$ φ ↔ φ
⟨*proof*⟩

lemma *neg-pneg-sem-eq* [*simp*]: (M,w |⊢ $\sim$ φ) = (~ (M,w |⊢ φ))

⟨proof⟩

With these preliminaries out of the way we turn to tackling issues related to the Fisher-Ladner closure. Observe that our semantics specify an unknown number of agents; this could potentially be an issue. However, we know for a given formula it can only mention a finite number of agents; hence the Fisher-Ladner subformula construction need only mention these agents.

To accomplish this, we first introduce an operation:

```
primrec dudes
  :: ('a,'b) evil-form ⇒ 'b set (δ)
where
    δ (P# p) = {}
  | δ ⊥ = {}
  | δ (⊙ X) = {X}
  | δ (φ → ψ) = (δ φ) ∪ (δ ψ)
  | δ (□ X φ) = {X} ∪ (δ φ)
  | δ ([−] X φ) = {X} ∪ (δ φ)
  | δ ([+] X φ) = {X} ∪ (δ φ)
```

lemma *finite-dudes*: *finite* (δ φ)
  ⟨proof⟩

The function δ, gathers a list of the people mentioned by the formula it takes as an argument. We shall use it as follows: our Fisher-Ladner closure will be programmed to carry around a little state which correspond to the δ mentioned by the top level formula. These people are the only people we care about in our universe.

We now turn to giving the subformula set; as is evident, it's very large. Moreover, unlike the Fisher-Ladner closure, it's not a *closure*, but this is irrelevant for our purposes anyway.

```
primrec evil-FL :: 'b set
                ⇒ ('a,'b) evil-form
                ⇒ ('a,'b) evil-form set (Σ) where
  evil-FL-P:
    Σ Δ (P# p) =   {P# p, ¬ (P# p), ⊥, ¬ ⊥}
                ∪ {[−]' X (P# p) | X. X ∈ Δ}
                ∪ {[+]' X (P# p) | X. X ∈ Δ}
                ∪ {¬ [−]' X (P# p) | X. X ∈ Δ}
                ∪ {¬ [+]' X (P# p) | X. X ∈ Δ}
| evil-FL-PP:
    Σ Δ (⊙ X) = {⊙ X, ¬ (⊙ X), ⊥, ¬ ⊥,
                 [−]' X (⊙ X), ¬ [−]' X (⊙ X)}
| evil-FL-Bot:
  Σ Δ ⊥ = {⊥, ¬ ⊥}
| evil-FL-Imp:
  Σ Δ (φ → ψ) = { φ → ψ, ¬ (φ → ψ),
                   φ, ψ, ¬ φ, ¬ ψ, ⊥, ¬ ⊥}
                 ∪ (Σ Δ φ) ∪ (Σ Δ ψ)
| evil-FL-B:
   Σ Δ (□ X φ) = { □ X φ, ¬ □ X φ,
                   [+]′ X (□ X φ), ¬ [+]′ X (□ X φ),
                    φ, ⊥, ¬ ⊥}
                 ∪ {□ X ([−]′ Y φ) | Y. Y ∈ Δ}
                 ∪ {¬ □ X ([−]′ Y φ) | Y. Y ∈ Δ}
                 ∪ {□ X ([+]′ Y φ) | Y. Y ∈ Δ}
                 ∪ {¬ □ X ([+]′ Y φ) | Y. Y ∈ Δ}
                 ∪ {[−]′ Y φ | Y. Y ∈ Δ}
                 ∪ {¬ [−]′ Y φ | Y. Y ∈ Δ}
                 ∪ {[+]′ Y φ | Y. Y ∈ Δ}
                 ∪ {¬ [+]′ Y φ | Y. Y ∈ Δ}
                 ∪ (Σ Δ φ)
| evil-FL-BB:
   Σ Δ ([+] X φ) = { [+] X φ, ¬ [+] X φ,
                      φ, ¬ φ, ⊥, ¬ ⊥}
                   ∪ (Σ Δ φ)
| evil-FL-BBI:
   Σ Δ ([−] X φ) = { [−] X φ, ¬ [−] X φ,
                      φ, ¬ φ, ⊥, ¬ ⊥}
                   ∪ (Σ Δ φ)
```

lemma *finite-evil-FL*:
    assumes *fin-dudes*: *finite* Δ
      shows *finite* (Σ Δ φ)
        — like the letters of a
        — fraternity of EviL...
⟨*proof*⟩

lemma *evil-FL-refl*: φ ∈ Σ Δ φ
⟨*proof*⟩

lemma *pneg-evil-FL*: ∀ ψ ∈ (Σ Δ φ). (∼ ψ) ∈ (Σ Δ φ)
⟨*proof*⟩

lemma *evil-FL-subform-refl*: ⇓ φ ⊆ Σ Δ φ
⟨*proof*⟩

lemma *evil-FL-subforms*: ∀ ψ ∈ Σ Δ φ. ⇓ ψ ⊆ Σ Δ φ
⟨*proof*⟩

lemma *evil-FL-BB-to-pBB*: $\forall\ \psi\ X.\ \ [-]\ X\ \psi \in \Sigma\ \Delta\ \varphi$
$\qquad\qquad\qquad\qquad\qquad \longrightarrow\ [-]'\ X\ \psi \in \Sigma\ \Delta\ \varphi$
⟨*proof*⟩

lemma *evil-FL-BBI-to-pBBI*: $\forall\ \psi\ X.\ \ [+]\ X\ \psi \in \Sigma\ \Delta\ \varphi$
$\qquad\qquad\qquad\qquad\qquad \longrightarrow\ [+]'\ X\ \psi \in \Sigma\ \Delta\ \varphi$
⟨*proof*⟩

With all of the above out of our way, we are ready to provide the subformula canonical model for a given formula φ. However, note that this model will only be *partly-evil*. We shall help ourself to the $\angle$ symbol for this construction; as far as we can tell from the literature, the meaning of $\angle$ appears to have been forgotten by logicians as it is never employed.

notation
evil-ClassAx.Atoms (*Atoms*) **and**
evil-ClassAx.lift-imp (**infix** :→ *24*)

definition *pevil-canonical-model* ::
 $('a,'b)$ *evil-form*
 $\Rightarrow$ $(('a,'b)$ *evil-form set*$,'a,'b)$ *evil-kripke* $(\angle)$
where
$\angle\ \varphi \equiv$
 (| W = *Atoms* $(\Sigma\ (\delta\ \varphi)\ \varphi)$,
 $V = (\lambda\ w\ p.\ (P\#\ p) \in w)$,
 $PP = (\lambda\ X.\ \{w.(\odot\ X) \in w\})$,
 $RB = (\lambda\ X.$
 $\{(w,v).\ \{w,v\} \subseteq$ *Atoms* $(\Sigma\ (\delta\ \varphi)\ \varphi)\ \wedge$
 $\{\psi.\ (\Box\ X\ \psi) \in w\} \subseteq v\})$,

 $RBB = (\lambda\ X.$
 $\{(w,v).\ \{w,v\} \subseteq$ *Atoms* $(\Sigma\ (\delta\ \varphi)\ \varphi)\ \wedge$
 $\{\psi.\ ([-]'\ X\ \psi) \in w\} \subseteq v\ \wedge$
 $\{([-]'\ X\ \psi)\ |\ \psi.\ ([-]'\ X\ \psi) \in w\} \subseteq v\ \wedge$
 $\{\psi.\ ([+]'\ X\ \psi) \in v\} \subseteq w\ \wedge$
 $\{([+]'\ X\ \psi)\ |\ \psi.\ ([+]'\ X\ \psi) \in v\} \subseteq w\})$,
 $RBBI = (\lambda\ X.$
 $\{(w,v).\ \{w,v\} \subseteq$ *Atoms* $(\Sigma\ (\delta\ \varphi)\ \varphi)\ \wedge$
 $\{\psi.\ ([+]'\ X\ \psi) \in w\} \subseteq v\ \wedge$
 $\{([+]'\ X\ \psi)\ |\ \psi.\ ([-]'\ X\ \psi) \in w\} \subseteq v\ \wedge$
 $\{\psi.\ ([-]'\ X\ \psi) \in v\} \subseteq w\ \wedge$
 $\{([-]'\ X\ \psi)\ |\ \psi.\ ([-]'\ X\ \psi) \in v\} \subseteq w\})$
 |)

declare *pevil-canonical-model-def* [*simp*]

To prove the truth lemma for $\angle\ \varphi$ we shall prove the inductive steps for the boxes seperately.

However, we first prove a variety of lemmas regarding basic propertiers of atoms.

— I will admit that my ealier formulation of Atoms is awkward
— This new lemma declares a simplification I will want

```
lemma evil-Atoms-simp[simp]:
  (Γ ∈ Atoms Φ) ≡
    (Γ ⊆ Φ ∧
    (∀ φ∈Φ. φ ∈ Γ ∨ (~ φ) ∈ Γ) ∧
    ~ (list Γ :⊢ ⊥))
⟨proof⟩

declare evil-pBB-def [simp del]
   and evil-pBBI-def [simp del]
```

Apparently we have to prove several lemmas relating to Atoms in order to be able to proceed.

```
lemma evil-mem-prv:
    assumes finite Φ
        and Γ ∈ Atoms Φ
        and φ ∈ Γ
      shows list Γ :⊢ φ
⟨proof⟩

lemma evil-mem-prv2:
    assumes finite Φ
        and Γ ∈ Atoms Φ
        and φ ∈ Φ
        and list Γ :⊢ φ
      shows φ ∈ Γ
⟨proof⟩

lemma evil-pneg-nded:
   assumes finite Φ
       and Γ ∈ Atoms Φ
       and list Γ :⊢ φ
     shows ~(list Γ :⊢ ~ φ)
⟨proof⟩

lemma evil-Atom-mem-intro:
   assumes finite Φ
```

```
        and $\Gamma \in$ *Atoms* $\Phi$
        and $\varphi \in \Gamma$
        and $\psi \in \Phi$
        and *list* $\Gamma$ :⊢ $\varphi \to \psi$
      shows $\psi \in \Gamma$
⟨proof⟩

lemma evil-Atom-pBB-intro:
    assumes finite $\Phi$
        and $\Gamma \in$ Atoms $\Phi$
        and $[-]\ X\ \varphi \in \Gamma$
        and $[-]'\ X\ \varphi \in \Phi$
      shows $[-]'\ X\ \varphi \in \Gamma$
⟨proof⟩

lemma evil-Atom-BB-intro:
    assumes finite $\Phi$
        and $\Gamma \in$ Atoms $\Phi$
        and $[-]'\ X\ \varphi \in \Gamma$
        and $[-]\ X\ \varphi \in \Phi$
      shows $[-]\ X\ \varphi \in \Gamma$
⟨proof⟩

lemma evil-Atom-pBBI-intro:
    assumes finite $\Phi$
        and $\Gamma \in$ Atoms $\Phi$
        and $[+]\ X\ \varphi \in \Gamma$
        and $[+]'\ X\ \varphi \in \Phi$
      shows $[+]'\ X\ \varphi \in \Gamma$
⟨proof⟩

lemma evil-Atom-BBI-intro:
    assumes finite $\Phi$
        and $\Gamma \in$ Atoms $\Phi$
        and $[+]'\ X\ \varphi \in \Gamma$
        and $[+]\ X\ \varphi \in \Phi$
      shows $[+]\ X\ \varphi \in \Gamma$
⟨proof⟩

lemma evil-FL-mem-prv:
  assumes $\Phi \in W(\angle\ \varphi)$
      and $\psi \in \Phi$
    shows list $\Phi$ :⊢ $\psi$
⟨proof⟩
```

thm *evil-mem-prv2*

lemma *evil-FL-mem-prv2*:
  assumes $\Phi \in W(\angle\ \varphi)$
    and $\psi \in \Sigma\ (\delta\ \varphi)\ \varphi$
    and *list* Φ :⊢ ψ
   shows $\psi \in \Phi$
⟨*proof*⟩

lemma *evil-FL-pneg-nded*:
  assumes $\Phi \in W(\angle\ \varphi)$
    and *list* Φ :⊢ ψ
   shows ~ (*list* Φ :⊢ ~ ψ)
⟨*proof*⟩

lemma *evil-FL-mem-intro*:
  assumes $\Phi \in W(\angle\ \varphi)$
     and $\psi \in \Phi$
     and $\chi \in \Sigma\ (\delta\ \varphi)\ \varphi$
     and *list* Φ :⊢ $\psi \rightarrow \chi$
    shows $\chi \in \Phi$
⟨*proof*⟩

lemma *evil-FL-pBB-intro*:
   assumes $\Phi \in W(\angle\ \varphi)$
     and $[-]\ X\ \psi \in \Phi$
    shows $[-]'\ X\ \psi \in \Phi$
⟨*proof*⟩

lemma *evil-FL-BB-intro*:
   assumes $\Phi \in W(\angle\ \varphi)$
     and $[-]'\ X\ \psi \in \Phi$
     and $[-]\ X\ \psi \in \Sigma\ (\delta\ \varphi)\ \varphi$
    shows $[-]\ X\ \psi \in \Phi$
⟨*proof*⟩

lemma *evil-FL-pBBI-intro*:
  assumes $\Phi \in W(\angle\ \varphi)$
     and $[+]\ X\ \psi \in \Phi$
    shows $[+]'\ X\ \psi \in \Phi$
⟨*proof*⟩

lemma *evil-push*:
  assumes *finite A*
    and *list* $(\{\psi\} \cup A)$:⊢ φ

shows *list A* :⊢ ψ → φ
⟨*proof*⟩

lemma *evil-push-dneg*:
assumes *finite A*
and *list* ({∼ ψ} ∪ *A*) :⊢ ⊥
shows *list A* :⊢ ψ
⟨*proof*⟩

lemma *map-to-comp*:
assumes *set L* = *S*
shows *set* (*map f L*) = {*f x* | *x*. *x* ∈ *S*}
⟨*proof*⟩

lemma *image-of-comp*:
f ‘ {*g* χ | χ . *P*(χ)} = {*f* (*g* χ) | χ . *P*(χ)}
⟨*proof*⟩

lemma *evil-unions-to-appends*:
assumes *finite A*
and *finite B*
shows (*list* (*A* ∪ *B*) @ Δ :⊢ ψ)
= (*list A* @ *list B* @ Δ :⊢ ψ)
⟨*proof*⟩

lemma *evil-B-forward*:
assumes *H1*: □ *X* ψ ∈ Φ
and *H2*: (Φ,Ψ) ∈ *RB*(∠ φ) *X*
shows ψ ∈ Ψ
⟨*proof*⟩

lemma *evil-BB-forward*:
assumes *H1*: [−] *X* ψ ∈ Φ
and *H2*: (Φ,Ψ) ∈ *RBB*(∠ φ) *X*
shows ψ ∈ Ψ
⟨*proof*⟩

lemma *evil-BBI-forward*:
assumes *H1*: [+] *X* ψ ∈ Φ
and *H2*: (Φ,Ψ) ∈ *RBBI*(∠ φ) *X*
shows ψ ∈ Ψ
⟨*proof*⟩

lemma *evil-B-back*:
assumes Φ ∈ *W*(∠ φ)

```
        and □ X ψ ∉ Φ
        and □ X ψ ∈ Σ (δ φ) φ
        shows ∃ Ψ. (Φ,Ψ) ∈ RB(∠ φ) X ∧ (~ ψ) ∈ Ψ
⟨proof⟩

end
```

10 Dual EviL Grammar and Semantics

theory *Dual-EviL-Semantics*
imports *EviL-Semantics*
begin

It should be noted that the previous grammar and semantics for EviL we have given are convenient for certain parts of the model theory of EviL and inconvenient for others. For instance, since classical logic may be axiomatized so succinctly using just letters, implication and falsum, and then confers Lindenbaum constructions to *any* extension, it is useful to have a grammar that reflects this. Likewise, the celebrated *axiom K* suggests that modal logic is naturally captured by extending the grammar of classical logic in precisely the manner we have, that is by incorporating modal □ operators. On the other hand, inductive arguments in this grammar and resulting can be challenging at times. However, the same inductive arguments in the *dual* grammar, incorporating letters, disjunction, negation, verum, and modal ◇ can be significantly simpler.

In this file, we give an alternate, *dual* grammar and semantics for both the Kripke and set-theoretic semantics for EviL, and in both cases we show that the original semantics are equivalent to the dual semantics under translation.

```
datatype ('a,'b) devil-form =
    DE-P 'a                          (P#' -)
  | DE-Top                           (⊤)
  | DE-Conj ('a,'b) devil-form
          ('a,'b) devil-form    (infixr ∧ 30)
  | DE-Neg ('a,'b) devil-form      (¬ - [40] 40)
  | DE-D 'b ('a,'b) devil-form    (◇)
  | DE-PP 'b                         (⊙')
  | DE-DD 'b ('a,'b) devil-form   ((−))
  | DE-DDI 'b ('a,'b) devil-form  ((+))

fun devil-eval :: ('a,'b) evil-world set
                      ⇒ ('a,'b) evil-world
                        ⇒ ('a,'b) devil-form
```

```
                   ⇒ bool (-,- ||⊨ - 50) where
  (-,(a,-) ||⊨ P#′ p) = (p ∈ a)
| (-,- ||⊨ ⊤) = True
| (Ω,(a,A) ||⊨ φ ∧ ψ) =
    ((Ω,(a,A) ||⊨ φ) ∧ (Ω,(a,A) ||⊨ ψ))
| (Ω,(a,A) ||⊨ ¬ φ) = (~ (Ω,(a,A) ||⊨ φ))
| (Ω,(a,A) ||⊨ ◇ X φ) =
    (∃ (b,B) ∈ Ω. (∀ χ ∈ A(X). b ⊨ χ)
                ∧ Ω,(b,B) ||⊨ φ)
| (Ω,(a,A) ||⊨ ⊙′ X) = (∀ χ ∈ A(X). a ⊨ χ)
| (Ω,(a,A) ||⊨ ⟨−⟩ X φ) = (∃ (b,B) ∈ Ω. a = b
                          ∧ B(X) ⊆ A(X)
                          ∧ Ω,(b,B) ||⊨ φ)
| (Ω,(a,A) ||⊨ ⟨+⟩ X φ) =
    (∃ (b,B) ∈ Ω. a = b ∧ B(X) ⊇ A(X) ∧ Ω,(b,B) ||⊨ φ)

fun devil-modal-eval :: ('w,'a,'b) evil-kripke
                  ⇒ 'w
                   ⇒ ('a,'b) devil-form
                    ⇒ bool (-,- ||⊢ - 50) where
  (M,w ||⊢ P#′ p) = (p ∈ V(M)(w))
| (-,- ||⊢ ⊤) = True
| (M,w ||⊢ φ ∧ ψ) =
    ((M,w ||⊢ φ) ∧ (M,w ||⊢ ψ))
| (M,w ||⊢ ¬ φ) = (~ (M,w ||⊢ φ))
| (M,w ||⊢  ◇ X φ) =
    (∃ v ∈ W(M). (w,v) ∈ RB(M)(X) ∧ M,v ||⊢ φ)
| (M,w ||⊢ ⊙′ X) = (w ∈ PP(M)(X))
| (M,w ||⊢ ⟨−⟩ X φ) =
    (∃ v ∈ W(M). (w,v) ∈ RBB(M)(X) ∧ M,v ||⊢ φ)
| (M,w ||⊢ ⟨+⟩ X φ) =
    (∃ v ∈ W(M). (w,v) ∈ RBBI(M)(X) ∧ M,v ||⊢ φ)

primrec devil :: ('a,'b) evil-form
              ⇒ ('a,'b) devil-form where
  devil P# p = P#′ p
| devil ⊥ = (¬ ⊤)
| devil (φ → ψ) = (¬ ((devil φ) ∧ ¬ (devil ψ)))
| devil (□ X φ) = (¬ (◇ X  (¬ (devil φ))))
| devil (⊙ X) = ⊙′ X
| devil ([−] X φ) = (¬ (⟨−⟩ X  (¬ (devil φ))))
| devil ([+] X φ) = (¬ (⟨+⟩ X  (¬ (devil φ))))
```

In all cases, the equivalence of the semantics follows from routine, utterly mechanical induction.

```
lemma evil-devil1:
  ∀ M. ∀ w. (M,w |⊨ φ) = (M,w ||⊨ devil φ)
⟨proof⟩

lemma evil-devil2:
  ∀ M. ∀ w. (M,w |⊢ φ) = (M,w ||⊢ devil φ)
⟨proof⟩
```

Next, we present a primitive recursive subformula operation. We show that it results in a finite list.

```
primrec devil-subforms
 :: ('a,'b) devil-form ⇒ ('a,'b) devil-form set (↓)
where
    ↓(P#' p) = {P#' p}
  | ↓(⊤) = {⊤}
  | ↓(¬ φ) = {¬ φ} ∪ ↓(φ)
  | ↓(φ ∧ ψ) = {φ ∧ ψ} ∪ ↓(φ) ∪ ↓(ψ)
  | ↓(◇ X φ) = {◇ X φ} ∪ ↓(φ)
  | ↓(⊙' X) = {⊙' X}
  | ↓(⟨−⟩ X φ) = {⟨−⟩ X φ} ∪ ↓(φ)
  | ↓(⟨+⟩ X φ) = {⟨+⟩ X φ} ∪ ↓(φ)

lemma finite-devil-subforms:
finite (↓ φ)
   ⟨proof⟩

lemma subform-refl [simp]:
φ ∈ ↓φ
  ⟨proof⟩
```

We next define a locale for a letter grabbing operation ϱ, which we shall employ in various model theoretic arguments.

```
locale EviL-ϱ =
  fixes φ :: ('a, 'b) devil-form
  fixes Ws :: 'w set
  fixes L :: 'a set
  assumes infi-L: infinite L
     and fini-Ws: finite Ws

definition (in EviL-ϱ) ϱ :: 'w ⇒ 'a
where ϱ == SOME g. inj-on g Ws
                  ∧ range g ⊆ (L − {p. (P#' p) ∈ (↓ φ)})
```

Above, we have picked ϱ to have the properties we desire, but we really have to prove that something like this exists or else we are talking nonsense (alas, this is the eternal curse of Brouwer's fallen angels, who forsook intuition and instead chose choice). Fortunately, the existance of the desired function is a consequence of various other facts we have as background.

lemma (**in** *EviL-ϱ*) *ϱ-works*:
  inj-on ϱ *Ws*
    $\wedge$ *range* $\varrho \subseteq L - \{p.\ (P\#'\ p) \in (\downarrow \varphi)\}$
⟨*proof*⟩

Next we'll show that φ can't really talk about $P\#'\ \varrho\ w$, and that ϱ preserves equality in *Ws*.

lemma (**in** *EviL-ϱ*) *φ-vocab*:
shows $P\#'\ \varrho(w) \notin \downarrow\varphi$
⟨*proof*⟩

lemma (**in** *EviL-ϱ*) *ϱ-eq*:
shows $\{w,v\} \subseteq Ws \Longrightarrow (w = v) = (\varrho(w) = \varrho(v))$
⟨*proof*⟩

end

11 EviL Column Lemmas

theory *EviL-Columns*
imports *EviL-Semantics EviL-Properties*
begin

We now turn to formalizing the concept of a *column* in the Kripke models we have been investigating, and show that *partly EviL* models make true certain lemmas regarding columns, which shall be key in the subsequent model theory that we shall develop.

definition
  col :: ('w,'a,'b) *evil-kripke* ⇒ 'w ⇒ 'w *set* **where**
  col M w ==
    $((\bigcup X.\ RBB(M)(X) \cup (RBB(M)(X))\hat{}{-1})\hat{}{*})$ `` $\{w\}$

We admit that the above definition is somewhat challenging, but it can be understood by observing the following elementary fact about relations.

lemma *crazy-Un-equiv*:
  equiv UNIV $((\bigcup i{\in}S.\ (r\ i) \cup (r\ i)\hat{}{-1})\hat{}{*})$
⟨*proof*⟩

This means evidently that *col M w* is an *equivalence class*, and by the properties of our definition it is a parition on the *universe* of possible worlds, regardless of whether they happen to be in the scope of whatever Kripke model we are worried about. Intuitively, we can think of the above definition as breaking up the universe into *connected components* of the graph that *RBB M* induces. This has several immediate consequences:

lemma *col-refl*:
  $w \in col\ M\ w$
⟨*proof*⟩

lemma *col-mem-eq*:
  $(v \in col\ M\ w) = (col\ M\ v = col\ M\ w)$
⟨*proof*⟩

The previous lemma weakens to the following equality:

lemma *weak-col-mem-eq*:
  $(v \in col\ M\ w) = (w \in col\ M\ v)$
⟨*proof*⟩

Next, we show, for partly EviL Kripke models, if $w \in W\ M$ then $col\ M\ w \subseteq W\ M$

lemma (**in** *partly-EviL*) *mem-col-subseteq*:
  $(w \in W(M)) = (col\ M\ w \subseteq W(M))$
⟨*proof*⟩

We now turn to proving a central equality regarding valuation functions for partly EviL Kripke models over columns, and give a equivalent formulation that is our preference.

lemma (**in** *partly-EviL*) *col-V-eqp*:
  shows $V(M)(w) = \bigcup\ V(M)$ ‘$(col\ M\ w)$
⟨*proof*⟩

lemma (**in** *partly-EviL*) *col-V-eq*:
  assumes $v \in col\ M\ w$
  shows $V(M)(w) = V(M)(v)$
⟨*proof*⟩

Finally, the other main lemma we present here regards visibility with *RB M X* and columns. We also give two equivalent formulations; once again we prefer the second formulation.

lemma (**in** *partly-EviL*) *col-RB-eqp*:
  $(w,v) \in RB(M)(X) = (\forall\ u \in col\ M\ v.\ (w,u) \in RB(M)(X))$
⟨*proof*⟩

lemma (**in** *partly-EviL*) *col-RB-eq*:
assumes $v \in col\ M\ u$
  shows $(w,v) \in RB(M)(X) = ((w,u) \in RB(M)(X))$
$\langle proof \rangle$

All of the above results suggest that columns are irreducible in at least three different ways. The following lemmas express this:

lemma (**in** *partly-EviL*) *col-W-irr*:
  shows $(\exists\, u \in col\ M\ v.\ u \in W(M))$
    $= (\forall\, u \in col\ M\ v.\ u \in W(M))$
$\langle proof \rangle$

lemma (**in** *partly-EviL*) *col-V-irr*:
  shows $(\exists\, u \in col\ M\ v.\ V(M)(u)(p))$
    $= (\forall\, u \in col\ M\ v.\ V(M)(u)(p))$
$\langle proof \rangle$

lemma (**in** *partly-EviL*) *col-RB-irr*:
  shows $(\exists\, u \in col\ M\ v.\ (w,u) \in RB(M)(X))$
    $= (\forall\, u \in col\ M\ v.\ (w,u) \in RB(M)(X))$
$\langle proof \rangle$

end