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** → *25*)
  fixes *vdash* :: $'a \Rightarrow bool$ ($\vdash$ - [*20*] *20*)
  assumes *ax1*: $\vdash \varphi \to \psi \to \varphi$
  assumes *ax2*: $\vdash (\varphi \to \psi \to \chi) \to (\varphi \to \psi) \to (\varphi \to \chi)$
  assumes *mp*: $\vdash \varphi \to \psi \implies \vdash \varphi \implies \vdash \psi$

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
qed (*fastsimp+*)
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*: $\vdash \varphi \to \varphi$

proof –
  from *ax1* [**where** φ=φ **and** ψ=$\varphi \rightarrow \varphi$]
    ax2 [**where** φ=φ **and** ψ=$\varphi \rightarrow \varphi$ **and** χ=φ]
    ax1 [**where** φ=φ **and** ψ=φ]
    show *?thesis* **by** (*blast intro*: *mp*)
qed

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

lemma (**in** *MinAx*) *weaken*: $\vdash \varphi \Longrightarrow \vdash \psi \rightarrow \varphi$
by (*blast intro*: *mp ax1*)

lemma (**in** *MinAx*) *hs*: $\vdash \varphi \rightarrow \psi \Longrightarrow \vdash \psi \rightarrow \chi \Longrightarrow \vdash \varphi \rightarrow \chi$
proof –
  assume $\vdash \varphi \rightarrow \psi$ $\vdash \psi \rightarrow \chi$
  moreover
   from *this*
     weaken [**where** ψ=φ **and** φ=$\psi \rightarrow \chi$]
    have $\vdash \varphi \rightarrow \psi \rightarrow \chi$ **by** *simp*
  moreover
   from *ax2* [**where** φ=φ **and** ψ=ψ **and** χ=χ]
    have $\vdash (\varphi \rightarrow \psi \rightarrow \chi) \rightarrow (\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi)$.
  ultimately show *?thesis* **by** (*blast intro*: *mp*)
qed

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**
  ([] :→ φ) = φ
| ((ψ # ψs) :→ φ) = (ψ → (ψs :→ φ))

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*) (ψs :→ φ) = *foldr* (% ψ φ. $\psi \rightarrow \varphi$) ψs φ
by (*induct* ψs) *simp-all*

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* $\Rightarrow$ $'a$ $\Rightarrow$ *bool* (**infix** $:\vdash$ *10*) **where**
  $(\Gamma :\vdash \varphi) \equiv (\vdash \Gamma :\to \varphi)$

lemma (**in** *MinAx*) *lift*: $\vdash \varphi \Longrightarrow \Gamma :\vdash \varphi$
  by (*induct* Γ, *auto*, *simp add*: *weaken*)

lemma (**in** *MinAx*) *lift-ax2*: $\vdash (\varphi s :\to (\psi \to \chi)) \to (\varphi s :\to \psi) \to (\varphi s :\to \chi)$
  proof (*induct* φs, *simp add*: *refl*)
    — We can solve the base case automatically.
    — It suffices to prove the inductive step.
  fix φs :: $'a$ *list*
  fix $a\ \psi\ \chi$ :: $'a$
  assume $\vdash (\varphi s :\to \psi \to \chi) \to (\varphi s :\to \psi) \to (\varphi s :\to \chi)$
  moreover from *this weaken*
    have $\vdash a \to ((\varphi s :\to \psi \to \chi) \to (\varphi s :\to \psi) \to (\varphi s :\to \chi))$
     by *fast*
  from *this ax2*
     show $\vdash (a \# \varphi s :\to \psi \to \chi) \to (a \# \varphi s :\to \psi) \to (a \# \varphi s :\to \chi)$
     by(*auto*, *blast intro*: *mp hs*)
qed

lemma (**in** *MinAx*) *lift-mp*: $\Gamma :\vdash \varphi \to \psi \Longrightarrow \Gamma :\vdash \varphi \Longrightarrow \Gamma :\vdash \psi$
by (*blast intro*: *mp lift-ax2*)

lemma (**in** *MinAx*) *lift-weaken*: $\Gamma :\vdash \varphi \Longrightarrow \Gamma :\vdash \psi \to \varphi$
by (*blast intro*: *ax1 lift lift-mp*)

lemma (**in** *MinAx*) *lift-ax1*: $\vdash \varphi \to (\psi s :\to \varphi)$
proof (*induct* ψs, *simp add*: *refl*)
  — Once again, base case is trivial so we only do inductive case
  fix ψs :: $'a$ *list*
  fix a :: $'a$
  assume $\vdash \varphi \to (\psi s :\to \varphi)$
  hence $[\varphi] :\vdash \psi s :\to \varphi$ **by** *simp*
  hence $[\varphi] :\vdash a \to (\psi s :\to \varphi)$ **by** (*blast intro*: *lift-weaken*)
  thus $\vdash \varphi \to (a \# \psi s :\to \varphi)$ **by** *simp*
qed

lemma (**in** *MinAx*) *lift-hs*: $\Gamma :\vdash \varphi \to \psi \Longrightarrow \Gamma :\vdash \psi \to \chi \Longrightarrow \Gamma :\vdash \varphi \to \chi$
proof –
  — We just follow the proof of the unlifted hypothetical syllogism
  assume $\Gamma :\vdash \varphi \to \psi$ $\Gamma :\vdash \psi \to \chi$

moreover
  from *this*
    lift-weaken [**where** Γ=Γ **and** ψ=φ **and** φ=$\psi \to \chi$]
   have $\Gamma :\vdash \varphi \to \psi \to \chi$ **by** *simp*
moreover
  from *ax2* [**where** φ=φ **and** ψ=ψ **and** χ=χ]
    lift
   have $\Gamma :\vdash (\varphi \to \psi \to \chi) \to (\varphi \to \psi) \to (\varphi \to \chi)$ **by** *simp*
ultimately show *?thesis* **by** (*blast intro*: *lift-mp*)
qed

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*: $\vdash (\varphi \to \psi \to \chi) \to (\psi \to \varphi \to \chi)$
proof –
  let $?\alpha = \varphi \to \psi \to \chi$
  from *refl* [**where** φ=ψ]
    weaken
    lift-weaken [**where** Γ=[$?\alpha$, ψ]
      and φ=ψ
      and ψ=φ]
    have $[?\alpha\ ,\psi,\ \varphi] :\vdash \psi$ **by** *auto*
  moreover
  from *refl* [**where** φ=$?\alpha$]
    lift-weaken [**where** Γ=[$?\alpha$]
      and ψ=ψ
      and φ=$?\alpha$]
    lift-weaken [**where** Γ=[$?\alpha$,ψ]
      and ψ=φ
      and φ=$?\alpha$]
    have $[?\alpha\ ,\psi,\ \varphi] :\vdash\ ?\alpha$ **by** *auto*
  moreover
  from *refl* [**where** φ=φ]
    lift [**where** Γ=[$?\alpha$,ψ]
      and φ=$\varphi \to \varphi$]
    have $[?\alpha\ ,\psi,\ \varphi] :\vdash \varphi$ **by** *auto*
  ultimately have $[?\alpha,\psi,\varphi] :\vdash \chi$ **by** (*blast intro*: *lift-mp*)
  thus *?thesis* **by** *auto*
qed

We next establish two analogues in using sequents

lemma (**in** *MinAx*) *lift-flip1*:
  $\vdash (\psi \to (\psi s :\to \varphi)) \to (\psi s :\to (\psi \to \varphi))$

proof (*induct* ψs, *auto simp add*: *refl*)
  fix ψs :: $'a$ *list* **fix** a :: $'a$
  assume $\vdash (\psi \to (\psi s :\to \varphi)) \to (\psi s :\to \psi \to \varphi)$
  hence $\vdash (a \to \psi \to (\psi s :\to \varphi)) \to a \to (\psi s :\to \psi \to \varphi)$
    by (*blast intro*: *weaken ax2 mp*)
  thus $\vdash (\psi \to a \to (\psi s :\to \varphi)) \to a \to (\psi s :\to \psi \to \varphi)$
    by (*blast intro*: *flip hs*)
qed

lemma (**in** *MinAx*) *lift-flip2*:
  $\vdash (\psi s :\to (\psi \to \varphi)) \to (\psi \to (\psi s :\to \varphi))$
proof (*induct* ψs, *auto simp add*: *refl*)
  fix ψs :: $'a$ *list* **fix** a :: $'a$
  assume $\vdash (\psi s :\to \psi \to \varphi) \to \psi \to (\psi s :\to \varphi)$
  hence $\vdash (a \to (\psi s :\to \psi \to \varphi)) \to a \to \psi \to (\psi s :\to \varphi)$
    by (*blast intro*: *weaken ax2 mp*)
  thus $\vdash (a \to (\psi s :\to \psi \to \varphi)) \to \psi \to a \to (\psi s :\to \varphi)$
    by (*blast intro*: *flip hs*)
qed

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*: $\vdash (\chi \to \chi \to \varphi) \to \chi \to \varphi$
proof –
  from *ax2* **have** $[\chi \to \chi \to \varphi] :\vdash (\chi \to \chi) \to (\chi \to \varphi)$
    by *auto*
  hence $[\chi \to \chi \to \varphi] :\vdash \chi \to \varphi$
    by (*blast intro*: *refl lift lift-mp*)
  thus *?thesis* **by** *auto*
qed

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*]:
$\vdash (\psi s :\to \varphi) \to ((removeAll\ \chi\ \psi s) :\to (\chi \to \varphi))$
proof(*induct* ψs, *auto simp add*: *ax1*)
  — Evidently there are two things to prove
  — The first is a trivial consequence of ax2
  fix ψs :: $'a$ *list* **fix** a :: $'a$
  assume $\vdash (\psi s :\to \varphi) \to (removeAll\ \chi\ \psi s :\to \chi \to \varphi)$
  thus $\vdash (a \to (\psi s :\to \varphi)) \to a \to (removeAll\ \chi\ \psi s :\to \chi \to \varphi)$
    by (*blast intro*: *weaken ax2 mp*)
  next
  — So we turn to the more involved part of the proof;

```
  — This really is the meat of the induction
  fix ψs :: 'a list
  assume A: ⊢ (ψs :→ φ) → (removeAll χ ψs :→ χ → φ)
  thus ⊢ (χ → (ψs :→ φ)) → (removeAll χ ψs :→ χ → φ)
  proof -
    let ?α = ψs :→ φ
    let ?β = removeAll χ ψs :→ χ → φ
    from A have ⊢ (χ → ?α) → χ → ?β by (blast intro: ax2 weaken mp)
    moreover
    from lift-flip1 have ⊢ (χ → ?β) → (removeAll χ ψs :→ χ → χ → φ) .
    moreover
    have
      ⊢ (removeAll χ ψs :→ (χ → χ → φ)) → ?β
      by (blast intro: imp-remove lift lift-ax2 mp)
    ultimately show ?thesis by (blast intro: hs)
  qed
qed
```

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

```
lemma (in MinAx) disch: Γ :⊢ φ ⟹ removeAll ψ Γ :⊢ ψ → φ
using lift-removeAll [where ψs=Γ and φ=φ and χ=ψ]
by (auto, blast intro: mp)

lemma (in MinAx) undisch [iff]: (Γ :⊢ ψ → φ) = (ψ # Γ :⊢ φ)
proof -
  have (Γ @ [ψ] :→ φ) = (Γ :→ (ψ → φ))
  proof (induct Γ, simp)
    fix Γ :: 'a list fix a :: 'a
    assume (Γ @ [ψ] :→ φ) = (Γ :→ ψ → φ)
    moreover have (a # Γ) @ [ψ] = a # (Γ @ [ψ])
      by (induct Γ) simp-all
    ultimately show ((a # Γ) @ [ψ] :→ φ) = (a # Γ :→ ψ → φ) by simp
  qed
  note ♠ = this
  moreover
  hence (ψ # Γ :⊢ φ) = (Γ @ [ψ] :⊢ φ)
    by (auto simp add: ♠,
          (blast intro: mp lift-flip1 lift-flip2)+)
  ultimately show ?thesis by fastsimp
qed

lemma (in MinAx) cut:
  assumes a: ψ # Γ :⊢ φ
      and b: Γ :⊢ ψ
```

shows $\Gamma :\vdash \varphi$
using *a b*
proof –
from *a undisch* **have** $\Gamma :\vdash \psi \rightarrow \varphi$ **by** *fastsimp*
with *b lift-mp* **show** *?thesis* **by** *blast*
qed

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*:
$\forall\ \psi s\ \varphi.\ set\ \psi s \subseteq set\ \chi s \longrightarrow (\vdash (\psi s :\rightarrow \varphi) \rightarrow (\chi s :\rightarrow \varphi))$
proof (*induct* χs)
{ fix $\psi s :: {'a}\ list$ **fix** $\varphi :: {'a}$
assume $set\ \psi s \subseteq set\ []$
hence $\vdash (\psi s :\rightarrow \varphi) \rightarrow ([] :\rightarrow \varphi)$ **by** (*auto*, *simp add*: *refl*) **}**
thus $\forall \psi s\ \varphi.\ set\ \psi s \subseteq set\ [] \longrightarrow (\vdash (\psi s :\rightarrow \varphi) \rightarrow ([] :\rightarrow \varphi))$ **by** *auto*
next
fix $a :: {'a}$ **fix** $\chi s :: {'a}\ list$
assume ♣: $\forall \psi s\ \varphi.\ set\ \psi s \subseteq set\ \chi s \longrightarrow (\vdash (\psi s :\rightarrow \varphi) \rightarrow (\chi s :\rightarrow \varphi))$
thus $\forall \psi s\ \varphi.\ set\ \psi s \subseteq set\ (a\ \#\ \chi s) \longrightarrow (\vdash (\psi s :\rightarrow \varphi) \rightarrow (a\ \#\ \chi s :\rightarrow \varphi))$
proof –
— To prove the above we first prove something more general
{ fix $\chi s\ \psi s :: {'a}\ list$ **fix** $a\ \varphi :: {'a}$
assume *a1*: $\forall \psi s\ \varphi.\ set\ \psi s \subseteq set\ \chi s \longrightarrow (\vdash (\psi s :\rightarrow \varphi) \rightarrow (\chi s :\rightarrow \varphi))$
assume *a2*: $set\ \psi s \subseteq set\ (a\ \#\ \chi s)$
from *a1 a2* **have** $\vdash (\psi s :\rightarrow \varphi) \rightarrow (a\ \#\ \chi s :\rightarrow \varphi)$
proof –
have $set\ \psi s \subseteq set\ \chi s \vee \sim (set\ \psi s \subseteq set\ \chi s)$ **by** *fast*
— Thus, we have two cases to prove for
moreover
{ assume $set\ \psi s \subseteq set\ \chi s$
with *a1* **have** $[\psi s :\rightarrow \varphi] :\vdash (\chi s :\rightarrow \varphi)$ **by** *fastsimp*
moreover with *ax1* **have** $[\chi s :\rightarrow \varphi] :\vdash (a\ \#\ \chi s :\rightarrow \varphi)$ **by** *fastsimp*
ultimately have $\vdash (\psi s :\rightarrow \varphi) \rightarrow (a\ \#\ \chi s :\rightarrow \varphi)$
by (*fastsimp intro*: *hs*)
}
moreover
{ let $?ls = removeAll\ a\ \psi s$
assume $\sim (set\ \psi s \subseteq set\ \chi s)$
with *a2* **have** $set\ ?ls \subseteq set\ \chi s$ **by** *fastsimp*
with *a1* **have** $\vdash (?ls :\rightarrow a \rightarrow \varphi) \rightarrow (\chi s :\rightarrow a \rightarrow \varphi)$
by *fastsimp*

hence $\vdash$ (*?ls* $:\to$ $a \to \varphi$) $\to$ (a # χs $:\to$ φ)
by (*auto*, *blast intro*: *lift-flip2 hs*)
hence $\vdash$ (ψs $:\to$ φ) $\to$ (a # χs $:\to$ φ)
by (*blast intro*: *lift-removeAll hs*)
}
ultimately show *?thesis* **by** *fast*
qed
}
— This evidently suffices to prove the theorem
with ♣ **show** *?thesis* **by** *fastsimp*
qed
qed

Finally, we can state *monotonicity*...

lemma (**in** *MinAx*) *lift-mono*: *set* $\Gamma \subseteq$ *set* $\Psi \Longrightarrow \Gamma :\vdash \varphi \Longrightarrow \Psi :\vdash \varphi$
using *imp-mono mp* **by** *blast*

lemma (**in** *MinAx*) *lift-eq*: *set* Γ = *set* $\Psi \Longrightarrow (\Gamma :\vdash \varphi) = (\Psi :\vdash \varphi)$
using *lift-mono*
equalityD1 [**where** *A*=*set* Γ **and** *B*=*set* Ψ]
equalityD2 [**where** *A*=*set* Γ **and** *B*=*set* Ψ]
by *blast*

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

lemma (**in** *MinAx*) *lift-elm*:
$\varphi \in$ *set* $\Gamma \Longrightarrow \Gamma :\vdash \varphi$
proof –
have $[\varphi] :\vdash \varphi$ **by** (*auto*, *simp add*: *refl*)
moreover assume $\varphi \in$ *set* Γ
hence *set* $[\varphi] \subseteq$ *set* Γ **by** *simp*
ultimately show *?thesis*
by (*blast intro*: *lift-mono*)
qed

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

lemma (**in** *MinAx*) *super-cut*:
assumes $\forall\ \varphi \in$ *set* Δ. $\Gamma :\vdash \varphi$
and Δ @ $\Gamma :\vdash \psi$
shows $\Gamma :\vdash \psi$
using *assms*
proof(*induct* Δ, *simp*)
fix a :: *'a* **fix** Δ :: *'a list*
assume a: $[\![\forall \varphi \in$*set* Δ. $\Gamma :\vdash \varphi$; Δ @ $\Gamma :\vdash \psi]\!] \Longrightarrow \Gamma :\vdash \psi$
and b: $\forall \varphi \in$*set* (a # Δ). $\Gamma :\vdash \varphi$

```
    and c: (a # Δ) @ Γ :⊢ ψ
  hence d: Γ :⊢ a by fastsimp
  have set Γ ⊆ set (Δ @ Γ)
    by (induct Δ) fastsimp+
  with d lift-mono [where Ψ=Δ @ Γ and Γ=Γ]
  have e: Δ @ Γ :⊢ a
    by simp
  have (a # Δ) @ Γ = a # Δ @ Γ
    by (induct Δ) simp-all
  with c e cut have f: Δ @ Γ :⊢ ψ by fastsimp
  from b have ∀ φ ∈ set Δ. Γ :⊢ φ by fastsimp
  with a f show ?thesis by auto
qed

end
```

2 A Classical Logic Axiom Class

```
theory ClassAxClass
imports MinAxClass
begin

class ClassAx = MinAx +
  fixes bot :: 'a     (⊥)
  assumes ax3: ⊢ ((φ → ⊥) → (ψ → ⊥)) → ψ → φ

instantiation bool :: ClassAx
begin
definition bot-bool-def[iff]: ⊥ = False

instance proof
qed (fastsimp+)
end

no-notation
Not  (¬ - [40] 40)

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

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 –
  from *ax1* **have** $\vdash \neg\, \varphi \to \neg\, \psi \to \neg\, \varphi$ **.**
  moreover from *ax3* **have** $\vdash (\neg\, \psi \to \neg\, \varphi) \to \varphi \to \psi$ **.**
  ultimately show *?thesis* **by** (*blast intro*: *hs*)
qed

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 –
  from *neg-elim* **have** $\vdash \neg\, \neg\, \varphi \to \neg\, \varphi \to \neg\, \neg\, \neg\, \varphi$ **.**
  moreover from *ax3* **have** $\vdash (\neg\, \varphi \to \neg\, \neg\, \neg\, \varphi) \to \neg\, \neg\, \varphi \to \varphi$ **.**
  ultimately have $\vdash \neg\, \neg\, \varphi \to \neg\, \neg\, \varphi \to \varphi$ **by** (*blast intro*: *hs*)
  thus *?thesis* **by** (*blast intro*: *imp-remove mp*)
qed

lemma (**in** *ClassAx*) *dblneg2*: $\vdash \varphi \to \neg\, \neg\, \varphi$
proof –
  from *dblneg1* **have** $\vdash \neg\, \neg\, \neg\, \varphi \to \neg\, \varphi$ **.**
  moreover from *ax3* **have** $\vdash (\neg\, \neg\, \neg\, \varphi \to \neg\, \varphi) \to \varphi \to \neg\, \neg\, \varphi$ **.**
  ultimately show *?thesis* **by** (*blast intro*: *mp*)
qed

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 –
  from *refl* **have** $\vdash \bot \to \bot$ **.**
  with *weaken* **have** $\vdash (\varphi \to \bot) \to (\bot \to \bot)$ **.**
  with *mp ax3* [**where** φ=φ **and** ψ=$\bot$]
    show *?thesis* **by** *blast*
qed

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

no-notation
op | (**infixr** $\vee$ *30*)

abbreviation (**in** *ClassAx*)
disj :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** $\vee$ *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*: $\vdash \varphi \lor \psi \to \psi \lor \varphi$
proof –
  from *refl* **have** $[\neg\ \varphi \to \psi] \mathrel{:}\vdash \neg\ \varphi \to \psi$ **by** *auto*
  moreover from *dblneg2 lift*
        have $[\neg\ \varphi \to \psi] \mathrel{:}\vdash \psi \to \neg\ \neg\ \psi$ **by** *blast*
  moreover note *lift-hs*
  ultimately have $[\neg\ \varphi \to \psi] \mathrel{:}\vdash \neg\ \varphi \to \neg\ \neg\ \psi$ **by** *blast*
  moreover from *ax3 lift*
   have $[\neg\ \varphi \to \psi] \mathrel{:}\vdash (\neg\ \varphi \to \neg\ \neg\ \psi) \to \neg\ \psi \to \varphi$
    by *blast*
  moreover note *lift-mp*
  ultimately have $[\neg\ \varphi \to \psi] \mathrel{:}\vdash \neg\ \psi \to \varphi$ **by** *blast*
  thus *?thesis* **by** *auto*
qed

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*:
$\vdash \varphi \lor \psi \to (\varphi \to \chi) \to (\psi \to \chi) \to \chi$
proof –
 let $?\Gamma = [\varphi \lor \psi, \varphi \to \chi, \psi \to \chi]$
 have $?\Gamma \mathrel{:}\vdash \varphi \lor \chi$
 proof –
  have $(\varphi \lor \psi) \in$ *set* $?\Gamma$ **by** *simp*
  with *lift-elm* **have** $?\Gamma \mathrel{:}\vdash \varphi \lor \psi$ **.**
  moreover have $(\psi \to \chi) \in$ *set* $?\Gamma$ **by** *simp*
  with *lift-elm* **have** $?\Gamma \mathrel{:}\vdash \psi \to \chi$ **.**
  moreover note *lift-hs*
  ultimately show *?thesis* **by** *blast*
 qed
 with *lift disj-comm lift-mp*
  have $?\Gamma \mathrel{:}\vdash \chi \lor \varphi$ **by** *blast*
 with *lift lift-hs dblneg2*
  have $?\Gamma \mathrel{:}\vdash \chi \lor \neg\ \neg\ \varphi$ **by** *blast*
 with *lift ax2 lift-mp*
  have $?\Gamma \mathrel{:}\vdash (\neg\ \chi \to \neg\ \varphi) \to \neg\ \neg\ \chi$
   by *blast*
 moreover have $?\Gamma \mathrel{:}\vdash \neg\ \chi \to \neg\ \varphi$
  proof –
   have $(\varphi \to \chi) \in$ *set* $?\Gamma$ **by** *simp*
   with *lift-elm* **have** $?\Gamma \mathrel{:}\vdash \varphi \to \chi$ **.**
   with *lift dblneg1 lift-hs*

```
      have ?Γ :⊢ ¬ ¬ φ → χ by blast
    with lift disj-comm lift-mp
      show ?thesis by blast
  qed
moreover
note lift-mp
ultimately have ?Γ :⊢ ¬ ¬ χ by best
with lift lift-mp dblneg1 [where φ=χ]
have ?Γ :⊢ χ by blast
thus ?thesis by auto
qed

lemma (in ClassAx) cdil:
  assumes a: Γ :⊢ φ ∨ ψ
    and b: Γ :⊢ φ → χ
    and c: Γ :⊢ ψ → χ
  shows Γ :⊢ χ
using a b c
proof –
  let ?α=φ ∨ ψ → (φ → χ) → (ψ → χ) → χ
  from disjE [where φ=φ and ψ=ψ and χ=χ]
      lift [where Γ=Γ and φ=?α]
  have Γ :⊢ ?α by auto
  with a lift-mp [where Γ=Γ and φ=φ ∨ ψ]
  have Γ :⊢ (φ → χ) → (ψ → χ) → χ by blast
  with b lift-mp [where Γ=Γ and φ=φ → χ]
  have Γ :⊢ (ψ → χ) → χ by blast
  with c lift-mp [where Γ=Γ and φ=ψ → χ]
  show ?thesis by blast
qed

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 $\exists\, ls.\ set\ ls = A$
  using *fin*
proof (*induct*, *simp*)
   — We only have to show for one case
   case (*insert a A*)
    then have $\exists\ ls\ .\ insert\ a\ A = set\ (a\ \#\ ls)$ **by** *fastsimp*
    thus *?case* **by** *blast*
qed

lemma *set-of-list-is-finite*:
  finite $(set\ \Gamma)$
by (*induct* Γ, *simp*, *clarify*)

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 $list :: {}'a\ set \Rightarrow {}'a\ list$ **where**
$list\ A = (SOME\ ls.\ set\ ls = A)$

lemma *set-list*: $finite\ A \longleftrightarrow (set\ (list\ A) = A)$
proof
    assume *finite A*
    with *finite-set-list-ex* [**where** $A{=}A$]
         some-eq-ex [**where** $P{=}\%\ ls.\ set\ ls = A$]
    show $set\ (list\ A) = A$
      by (*induct*, *simp add*: *list-def*)
  next
    assume $set\ (list\ A) = A$
    with *set-of-list-is-finite* [**where** $\Gamma{=}list\ A$]
    show *finite A* **by** *simp*
qed

lemma *empty-set-list*[*simp*]: $list\ \{\} = []$
proof –
  { fix *ls*
    have $\tilde{}(ls = []) \longrightarrow \tilde{}(set\ ls = \{\})$
      by (*induct ls*, *fastsimp*, *auto*) **}**

```
  hence ∃! ls. set ls = {} by fastsimp+
  with some1-equality [where a=[]
                        and P=% ls. set ls = {}]
  show ?thesis by (simp add: list-def)
qed
```

We now turn to showing that if $A::'a \Rightarrow bool$ is finite and $B::'b \Rightarrow bool$ is infinite, then there exists a function $f::'a \Rightarrow 'b$ 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)
using one-one
proof –
  { fix b from one-one have b ∈ (f ‘ A) = (EX! x. x ∈ A ∧ b = f x)
      by (unfold inj-on-def, unfold image-def, blast) }
  note ♡ = this
  — We now turn to crafting our choice function
  let ?g = % b. SOME x. x ∈ A ∧ b = f x
  — We'll prove that it's one-one now
  { fix b c
    assume I:  b ∈ (f ‘ A)
       and II: c ∈ (f ‘ A)
    have (b = c) = (?g b = ?g c)
    proof (auto)
      — We shall only show right to left
      assume ?g b = ?g c
      moreover from ♡ I someI-ex [where P=% x. x ∈ A ∧ b = f x]
         have A: b = f (?g b) by blast
      moreover from ♡ II someI-ex [where P=% x. x ∈ A ∧ c = f x]
         have B: c = f (?g c) by blast
      ultimately show b = c by fastsimp
    qed }
  with inj-on-def have inj-on ?g (f ‘ A) by blast
  moreover
  — Evidently λb. SOME x. x ∈ A ∧ b = f x is a left-inverse of f
  — relative to A
  { fix x assume x ∈ A
```

with ♡ *one-one*
someI-ex [**where** P=% $y.\ y \in A \wedge f\ x = f\ y$]
have $x = (?g\ o\ f)\ x$
by (*unfold inj-on-def*, *unfold comp-def*, *blast*) **}**
moreover
— $\lambda b.$ *SOME* $x.\ x \in A \wedge b = f\ x$ is also a right-inverse of f
— relative to f ‘ A
{ fix y **assume** A: $y \in f$ ‘ A
with ♡ *someI-ex* [**where** P=% $x.\ x \in A \wedge y = f\ x$]
have $(f\ o\ ?g)\ y = y$
by (*unfold comp-def*, *fastsimp*) **}**
ultimately show *?thesis* **by** (*rule-tac* x=$?g$ **in** *exI*, *simp*)
qed

lemma *fin-inj-on-infi*:
assumes *fin-A*: *finite* $(A :: 'a\ set)$
and *infi-B*: *infinite* $(B :: 'b\ set)$
shows $\exists\, g{::}'a \Rightarrow 'b.$ *inj-on* $g\ A \wedge$ *range* $g \subseteq B$
using *fin-A infi-B*
proof –
from *fin-A*
finite-imp-nat-seg-image-inj-on
obtain $n\ f$
where $A = (f{::}nat \Rightarrow 'a)$ ‘ $\{i.\ i < (n{::}nat)\} \wedge$ *inj-on* $f\ \{i.\ i < n\}$
by *fastsimp*
moreover with *inj-on-inj-off* **obtain** g
where *inj-on* $(g{::}'a \Rightarrow nat)$ $(f$ ‘ $\{i.\ i < n\})$ **by** *blast*
ultimately have *inj-on* $g\ A$ **by** *fastsimp*
note ♡ = *this*
from *infi-B infinite-countable-subset* [**where** S=B]
obtain h **where** *inj* $(h{::}nat \Rightarrow 'b) \wedge$ *range* $h \subseteq B$
by *fastsimp*
note ♠ = *this*
hence *inj-on* h $(g$ ‘ $A)$ **by** (*unfold inj-on-def*,*blast*)
with ♡ *comp-inj-on*
have *inj-on* $(h\ o\ g)\ A$ **by** *blast*
moreover
{ fix $g\ h$ **have** *range* $(h\ o\ g) \subseteq$ *range* h
by (*unfold comp-def*, *blast*) **}**
with ♠ **have** *range* $(h\ o\ g) \subseteq B$ **by** *fastsimp*
ultimately show *?thesis* **by** *fastsimp*
qed

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 *cases*
  assume ∃ ψ. (¬ ψ) = φ
  — For clarification, the *someI-ex* states: ∃ *x*. *?P x* ⟹ *?P* (*SOME x*. *?P x*)
  with *someI-ex* [**where** *P*=% ψ . (¬ ψ) = φ]
    pneg-def [**where** φ=φ]
   have (¬ ~ φ) = φ **by** *fastsimp*
  moreover from *dblneg1* **have** ⊢ ¬ ~ φ ∨ ~ φ **.**
  with *disj-comm mp* **have** ⊢ ~ φ ∨ ¬ ~ φ **by** *blast*
  ultimately show *?thesis* **by** *simp*

  next assume ~ (∃ ψ. (¬ ψ) = φ)
  with *pneg-def* [**where** φ=φ]
   have (~ φ) = (¬ φ) **by** *fastsimp*
  moreover from *dblneg1* **have** ⊢ ¬ φ ∨ φ **.**
  ultimately show *?thesis* **by** *simp*
qed

lemma (**in** *ClassAx*) *pneg-negimpI*: ⊢ ¬ φ → ~ φ
  by (*blast intro*: *pneg-tnd disj-comm mp*)

lemma (**in** *ClassAx*) *pneg-negimpII*: ⊢ ~ φ → ¬ φ
proof *cases*
  assume *a*: ∃ ψ. (¬ ψ) = φ
  then have (~ φ) = (*SOME* ψ. (¬ ψ) = φ)
   by (*simp add*: *pneg-def*)
  with *a*

```
    someI-ex [where P=% ψ. (¬ ψ) = φ]
  have (¬ (~ φ)) = φ by simp
  with dblneg1 have ⊢ ¬ ¬ φ → ¬ ~ φ
    by simp
  with ax3 [where φ=¬ φ
            and ψ=~ φ]
  show ?thesis by (blast intro: mp)
  next assume b: ~ (∃ψ. (¬ ψ) = φ)
  then have (~ φ) = (¬ φ)
      by (simp add: pneg-def)
  with  refl show ?thesis
   by simp
qed
```

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)
  using ♡
proof –
  { assume a: φ # Γ :⊢ ψ
      and b: (~ φ) # Γ :⊢ ψ
    from a undisch have Γ :⊢ φ → ψ by simp

    moreover from b undisch have Γ :⊢ ~ φ → ψ
      by simp

    moreover from pneg-tnd have ⊢ (~ φ) ∨ φ .
    with lift have Γ :⊢ (~ φ) ∨ φ by fast

    moreover note cdil [where Γ=Γ]

    ultimately have Γ :⊢ ψ by blast }
  with ♡ show ?thesis by fastsimp
qed
```

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 $'a \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$, which first takes a formula $\psi{::}'a$. It then takes a $\varphi{::}'a$ off the top of the second argument $\Phi{::}'a\ list$ and adds it to the consistently first argument $\Gamma{::}'a\ list$ if it may be consistently added without proving ψ, and adds $\sim\varphi$ otherwise. The procedure then recurses.

primrec (**in** *ClassAx*) *lind* :: $'a \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$ **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*:
$\forall\Gamma$. *set* $\Gamma \subseteq$ *set* (*lind* ψ Γ Φ)
proof (*induct* Φ, *simp*)
 — The base case is trivial, so we only show the inductive step
 fix φ :: $'a$ **fix** Φ :: $'a$ *list*
 assume *indh*: $\forall\Gamma$. *set* $\Gamma \subseteq$ *set* (*lind* ψ Γ Φ)
 — From this assumption, we will show the consequent,
 — where Γ is arbitrary
 { fix Γ :: $'a$ *list*
  let *?if* = *if* ~(φ # Γ :⊢ ψ)
        then φ
        else (~ φ)
  — Next, observe these two facts:
  have *a*: *set* $\Gamma \subseteq$ *set* (*?if* # Γ)
   by *fastsimp*
  have *lind* ψ Γ (φ # Φ) = *lind* ψ (*?if* # Γ) Φ **by** *simp*
  with *indh* **have** *b*: *set* (*?if* # Γ) $\subseteq$ *set* (*lind* ψ Γ (φ # Φ))
   by *fastsimp*
  — With these two facts, we can show, for any x,
  — that $x \in$ *set* (*lind* ψ Γ (φ # Φ))
  { fix x :: $'a$ **assume** $x \in$ *set* Γ
   with *a* **have** $x \in$ *set* (*?if* # Γ) **by** *fast*
   with *b* **have** $x \in$ *set* (*lind* ψ Γ (φ # Φ)) **by** *fast* **}**
  hence *set* $\Gamma \subseteq$ *set* (*lind* ψ Γ (φ # Φ)) **by** *fast* **}**
 — This suffices to prove the theorem
 thus $\forall\Gamma$. *set* $\Gamma \subseteq$ *set* (*lind* ψ Γ (φ # Φ)) **by** *fast*
qed

```
lemma (in ClassAx) lind-is-max:
∀Γ. ∀φ ∈ set Φ. φ ∈ set (lind ψ Γ Φ) ∨ (~ φ) ∈ set (lind ψ Γ Φ)
proof (induct Φ, simp)
  — We shall only prove the inductive step
  fix χ :: 'a fix Φ :: 'a list
  assume ind-hyp: ∀ Γ. ∀φ∈set Φ. φ ∈ set (lind ψ Γ Φ)
                                    ∨ (~ φ) ∈ set (lind ψ Γ Φ)
  — First, let Γ and φ be arbitrary
  { fix Γ :: 'a list fix φ :: 'a
    — Next, let we'll use our previous abbreviation
    let ?if = if ~(χ # Γ :⊢ ψ)
                then χ
                else (~ χ)
    — The following identity will turn out to be crucial
    have ♡:
      lind ψ (?if # Γ) Φ = lind ψ Γ (χ # Φ) by fastsimp
    — Next, assume the proper domain conditions for φ
    assume ◇: φ ∈ set (χ # Φ)
    have φ ∈ set (lind ψ Γ (χ # Φ))
         ∨ (~ φ) ∈ set (lind ψ Γ (χ # Φ))
    proof cases
      assume φ ∈ set Φ
      with ind-hyp have
       φ ∈ set (lind ψ (?if # Γ) Φ)
        ∨ (~ φ) ∈ set (lind ψ (?if # Γ) Φ) by fast
      with ♡ show ?thesis by fastsimp
     next
      assume φ ∉ set Φ
      with ◇ have φ = χ by (induct Φ, fastsimp+)
      hence φ = ?if ∨ (~ φ) = ?if by fastsimp
      moreover have ?if ∈ set (?if # Γ) by fastsimp
      with ♡ lind-is-mono have
       ?if ∈ set (lind ψ Γ (χ # Φ)) by fastsimp
      ultimately show ?thesis by fastsimp
    qed }
  thus
  ∀Γ. ∀φ∈set (χ # Φ). φ ∈ set (lind ψ Γ (χ # Φ))
                        ∨ (~ φ) ∈ set (lind ψ Γ (χ # Φ))
   by fast
qed

lemma (in ClassAx) lind-is-bounded:
  assumes pneg-closed: (∀ φ ∈ set Φ. (~ φ) ∈ set Φ)
    shows ∀ Γ. set (lind ψ Γ Φ) ⊆ set Γ ∪ set Φ
using pneg-closed
```

proof –
— We cannot see how to perform this proof through direct
— induction, so we shall prove it a little, more obliquely.
— Observe the following statement:
let *?pnegset* = % Φ . $\{\varphi.\ \exists\ \psi\ .\ \varphi = (\sim \psi) \wedge \psi \in$ *set* $\Phi\}$
from *pneg-closed* **have** *?pnegset* $\Phi \subseteq$ *set* Φ **by** *fastsimp*
— This inspires an inductive proof which may be performed
moreover
have $\forall\ \Gamma$. *set* (*lind* $\psi\ \Gamma\ \Phi$) $\subseteq$ *set* $\Gamma\ \cup$ *set* $\Phi\ \cup$ *?pnegset* Φ
proof (*induct* Φ, *simp*)
fix χ :: *'a* **fix** Φ :: *'a list*
assume *ind-hyp*:
$\forall\,\Gamma$. *set* (*lind* $\psi\ \Gamma\ \Phi$) $\subseteq$ *set* $\Gamma\ \cup$ *set* $\Phi\ \cup$ *?pnegset* Φ
— As usual, we will show for Γ arbitrary
{ fix Γ :: *'a list*
from *ind-hyp* **have**
set (*lind* $\psi\ \Gamma\ (\chi\ \#\ \Phi)$) $\subseteq$
set $\Gamma\ \cup$ *set* $(\chi\ \#\ \Phi)\ \cup$ *?pnegset* $(\chi\ \#\ \Phi)$
by *fastsimp* **}**

thus
$\forall\,\Gamma$. *set* (*lind* $\psi\ \Gamma\ (\chi\ \#\ \Phi)$) $\subseteq$
set $\Gamma\ \cup$ *set* $(\chi\ \#\ \Phi)\ \cup$ *?pnegset* $(\chi\ \#\ \Phi)$
by *fast*
qed
ultimately show *?thesis* **by** *blast*
qed

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

lemma (**in** *ClassAx*) *lind-is-cnst*:
$\forall\,\Gamma$. ~ (Γ :⊢ ψ) $\longrightarrow$ ~ (*lind* $\psi\ \Gamma\ \Phi$:⊢ ψ)
proof (*induct* Φ, *simp*)
— As expected, all we prove is the inductive step.
fix $\psi\ \chi$ :: *'a* **fix** Φ :: *'a list*
assume *ind-hyp*: $\forall\,\Gamma$. ~ (Γ :⊢ ψ) $\longrightarrow$ ~ (*lind* $\psi\ \Gamma\ \Phi$:⊢ ψ)
— We shall show the statement of the theorem where Γ is free
{ fix Γ :: *'a list*
let *?if* = *if* ~($\chi\ \#\ \Gamma$:⊢ ψ)
then χ
else ($\sim \chi$)
— We will need this key fact
have *key*: *lind* $\psi\ \Gamma\ (\chi\ \#\ \Phi)$ = *lind* ψ (*?if* $\#\ \Gamma$) Φ **by** *simp*
— From this, we turn to completing the proof
assume ♡: ~ (Γ :⊢ ψ)

have ~ (*lind* ψ Γ (χ # Φ) :⊢ ψ)
proof *cases*
assume *a*: χ # Γ :⊢ ψ
with ♡ *cnst* **have**
~ ((~ χ) # Γ :⊢ ψ) **by** *fastsimp*
with *ind-hyp* **have**
~ (*lind* ψ ((~ χ) # Γ) Φ :⊢ ψ) **by** *blast*
with *a key* **show** *?thesis* **by** *fastsimp*
next
assume *b*: ~(χ # Γ :⊢ ψ)
with *ind-hyp* **have**
~ (*lind* ψ (χ # Γ) Φ :⊢ ψ) **by** *blast*
with *b key* **show** *?thesis* **by** *fastsimp*
qed }
thus $\forall\Gamma$. ~ (Γ :⊢ ψ) $\longrightarrow$ ~ (*lind* ψ Γ (χ # Φ) :⊢ ψ) **by** *fast*
qed

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**
At Φ Γ ≡ $\Gamma \subseteq \Phi$
∧ ($\forall \varphi \in \Phi$. $\varphi \in \Gamma \vee (\sim \varphi) \in \Gamma$)
∧ ~(*list* Γ :⊢ ⊥)

lemma (**in** *ClassAx*) *coincidence*:
assumes *A*: *finite* Φ
and *B*: $\Gamma \in At(\Phi)$
and *C*: $\varphi \in \Phi$
and *D*: *P* (~ φ) = (~ *P* φ)
shows ($\varphi \in \Gamma$) = (*list* Γ :⊢ φ)
and *P* φ = (*list* Γ :⊢ φ) $\Longrightarrow$ *P* (~ φ) = (*list* Γ :⊢ ~ φ)
using *A B C D*
proof –
— We shall first make some observations:
from *A B C*
mem-def [**where** *x*=Γ
and *S*=*At*(Φ)]
Atoms-def [**where** Γ=Γ
and Φ=Φ]
have $\Gamma \subseteq \Phi$
and *E*: $\varphi \in \Gamma \vee (\sim \varphi) \in \Gamma$

```
 and F: ~(list Γ :⊢ ⊥)
       by fastsimp+
with A finite-subset [where A=Γ
                      and B=Φ]
    set-list [where A=Γ]
have G: Γ = set (list Γ) by fastsimp

— Our coincidence lemma has two statements; here is the first:
show H: (φ ∈ Γ) = (list Γ :⊢ φ)
proof –
  — The first direction in this case is trivial
  from G lift-elm have φ ∈ Γ ⟹ list Γ :⊢ φ
     by blast
  — The other direction is evidently more challenging
  moreover
  { assume ♡: list Γ :⊢ φ
    have φ ∈ Γ
    proof –
      — The proof proceeds by contradiction:
      { assume φ ∉ Γ
        with E have (~ φ) ∈ Γ by fastsimp
        with G lift-elm have list Γ :⊢ ~ φ by blast
        with pneg-negimpII
             lift [where Γ=list Γ]
             lift-mp [where Γ=list Γ]
        have list Γ :⊢ ¬ φ by blast
        with ♡ lift-mp have list Γ :⊢ ⊥ by fast
        with F have False by fast }
      thus ?thesis by fast
    qed }
  ultimately show ?thesis by blast
qed

— We now turn to the second statement; but we shall first
— make a critical observation:
have I: φ ∉ Γ = (list Γ :⊢ ~ φ)
proof –
— Left to right:
  { assume φ ∉ Γ
      with E have (~ φ) ∈ Γ by auto
      with G lift-elm
            have list Γ :⊢ ~ φ by blast }
moreover
— Right to left:
  { assume φ ∈ Γ
```

and *list* $\Gamma :\vdash \sim \varphi$
moreover with *G lift-elm*
have *list* $\Gamma :\vdash \varphi$ **by** *blast*
moreover note *F*
pneg-negimpII [**where** φ=φ]
lift [**where** Γ=*list* Γ]
lift-mp [**where** Γ=*list* Γ]
ultimately have *False* **by** *blast* }
ultimately show *?thesis* **by** *auto*
qed

— This is enough to finally show the second statement:
assume $P\ \varphi$ = (*list* $\Gamma :\vdash \varphi$)
with *D H* **have** $P\ (\sim \varphi) = (\varphi \notin \Gamma)$ **by** *simp*
with *I* **show** $P\ (\sim \varphi)$ = (*list* $\Gamma :\vdash (\sim \varphi)$) **by** *simp*
qed

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*: $\tilde{}$(*list* $\Gamma :\vdash \psi$)
shows $\exists\ \Gamma'$. *At* $\Phi\ \Gamma'$
$\wedge\ \Gamma \subseteq \Gamma'$
$\wedge$ $\tilde{}$(*list* $\Gamma' :\vdash \psi$)
using *A B C D*
proof –
from *A C finite-subset* **have**
finite Γ **by** *fastsimp*
with *set-list* [**where** *A*=Γ] **have**
E: Γ = *set* (*list* Γ) **by** *auto*
from *A set-list* [**where** *A*=Φ] **have**
F: Φ = *set* (*list* Φ) **by** *auto*
let *?lindy* = *set* (*lind* ψ (*list* Γ) (*list* Φ))
from *set-of-list-is-finite* **have**
finite ?lindy **by** *fastsimp*
with *set-list* [**where** *A*=*?lindy*] **have**
G: *?lindy* = *set* (*list ?lindy*) **by** *auto*

— We have many things to prove:
from *B C E F*

lind-is-bounded [**where** Φ=*list* Φ]
have *I*: *?lindy* $\subseteq \Phi$
by *blast*

from *F lind-is-max*
have *II*: $\forall\, \varphi \in \Phi.\ \varphi \in$ *?lindy* $\vee$ ($\sim \varphi$) $\in$ *?lindy*
by *fastsimp*

from *D G*
lind-is-cnst [**where** Φ=*list* Φ]
lift-eq [**where** Γ=*lind* ψ (*list* Γ) (*list* Φ)
and Ψ=*list ?lindy*]
have *III*: ~ (*list ?lindy* :⊢ ψ)
by *blast*

from *III*
expls [**where** φ=ψ]
lift [**where** Γ=*list ?lindy*]
lift-mp [**where** Γ=*list ?lindy*]
have *IV*: ~ (*list ?lindy* :⊢ $\perp$)
by *blast*

from *E*
lind-is-mono [**where** ψ=ψ
and Φ=*list* Φ]
have *V*: $\Gamma \subseteq$ *?lindy*
by *blast*

from *I II IV*
Atoms-def [**where** Φ=Φ
and Γ=*?lindy*]
have *VI*: *At* Φ *?lindy* **by** *fastsimp*

from *III V VI* **show** *?thesis* **by** *fastsimp*
qed

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 $(\bot)$ **and**
  imp $(\textbf{infixr} \to \mathit{25})$ **and**
  vdash $(\vdash\ \text{-}\ [\mathit{20}]\ \mathit{20})$ **and**
  lift-vdash $(\textbf{infix} :\vdash\ \mathit{10})$ **and**
  Not $(\neg\ \text{-}\ [\mathit{40}]\ \mathit{40})$ **and**
  neg $(\neg\ \text{-}\ [\mathit{40}]\ \mathit{40})$ **and**
  pneg $(\sim\ \text{-}\ [\mathit{40}]\ \mathit{40})$

datatype *'a cl-form* =
  CL-P 'a $(P\#)$
  | *CL-Bot* $(\bot)$
  | *CL-Imp 'a cl-form 'a cl-form* $(\textbf{infixr} \to \mathit{25})$

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

fun *cl-eval* :: *'a set* ⇒ *'a cl-form* ⇒ *bool* $(\textbf{infix} \vDash \mathit{20})$ **where**
  $(S \vDash P\#\ p) = (p \in S)$
  | $(\text{-} \vDash \bot) = \mathit{False}$
  | $(S \vDash \varphi \to \psi) = ((S \vDash \varphi) \longrightarrow (S \vDash \psi))$

abbreviation
cl-neg :: *'a cl-form* ⇒ *'a cl-form* $(\neg\ \text{-}\ [\mathit{40}]\ \mathit{40})$ **where**
$\neg\ \varphi \equiv (\varphi \to \bot)$

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: $(\varphi \to \psi \to \varphi) \in CL$ |
  cl-ax2: $((\varphi \to \psi \to \chi) \to (\varphi \to \psi) \to (\varphi \to \chi)) \in CL$ |
  cl-ax3: $(((\varphi \to \bot) \to \psi \to \bot) \to \psi \to \varphi) \in CL$ |
  cl-mp: $[\![\ (\varphi \to \psi) \in CL;\ \varphi \in CL\ ]\!] \Longrightarrow \psi \in CL$

abbreviation *cl-vdash* :: *'a cl-form* ⇒ *bool* $(\vdash\ \text{-}\ [\mathit{20}]\ \mathit{20})$ **where**

$(\vdash \varphi) \equiv \varphi \in CL$

As per tradition, soundness is trivial:

lemma *cl-soundness*: $\vdash \varphi \Longrightarrow S \models \varphi$
  by (*induct set*: *CL*, *auto*)

Furthermore, This trivially implies that that CL is consistent:

lemma *cl-const*: ~ $(\vdash \bot)$
using *cl-soundness*
  by *fastsimp*

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* $\rightarrow$ *cl-vdash* $\bot$
proof qed (*fastsimp intro*: *CL.intros*)+

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

primrec *FL* :: *'a cl-form* $\Rightarrow$ *'a cl-form set* **where**
  $FL\ (P\#\ p) = \{P\#\ p, \neg\ (P\#\ p), \bot, \neg\ \bot\}$
| $FL\ \bot = \{\bot, \neg\ \bot\}$
| $FL\ (\varphi \rightarrow \psi) = \{\ \varphi \rightarrow \psi, \neg\ (\varphi \rightarrow \psi),$
        $\varphi, \neg\ \varphi, \psi, \neg\ \psi\ , \bot, \neg\ \bot\}$
      $\cup\ FL\ \varphi \cup FL\ \psi$

lemma *finite-FL*: *finite* $(FL\ \varphi)$
  by (*induct* φ) *simp-all*

lemma *imp-closed-FL*: $(\psi \rightarrow \chi) \in FL\ \varphi$
        $\Longrightarrow \psi \in FL\ \varphi \wedge \chi \in FL\ \varphi$
proof –
  assume ♡: $(\psi \rightarrow \chi) \in FL\ \varphi$
  hence $\psi \in FL\ \varphi$
    by (*induct* φ, *fastsimp*+)
  moreover from ♡ **have** $\chi \in FL\ \varphi$
    by (*induct* φ, *fastsimp*+)
  ultimately show *?thesis* **by** *auto*
qed

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 (¬ φ) = φ

abbreviation cl-pneg :: 'a cl-form ⇒ 'a cl-form (~' - [40] 40)
  where
  ~' φ ≡ (if (∃ ψ. (¬ ψ) = φ)
           then (dest-neg φ)
           else ¬ φ)

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

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

lemma cl-pneg-eq: (~' φ) = (~ φ)
proof cases
   assume a: ∃ ψ. (¬ ψ) = φ
   hence ∃! ψ. (¬ ψ) = φ by fastsimp
   moreover
   then have (¬ ~' φ) = φ by fastsimp
   moreover from a
              pneg-def [where φ=φ]
   have (~ φ) = (SOME ψ . (¬ ψ) = φ) by fastsimp
   moreover note
   — some1-equality states [[∃!x. ?P x; ?P ?a]] ⟹ (SOME x. ?P x) = ?a
     some1-equality [where P=% ψ . (¬ ψ) = φ
                      and a=~' φ]
   ultimately show ?thesis by auto
  next
   assume b: ~ (∃ ψ. (¬ ψ) = φ)
   with pneg-def [where φ=φ]
   show ?thesis by fastsimp
qed

lemma neg-pneg-sem-eq: (~ (S ⊨ φ)) = (S ⊨ ~ φ)
proof cases
   assume a: ∃ ψ. (¬ ψ) = φ
   hence (¬ ~' φ) = φ by fastsimp
   hence (¬ ~ φ) = φ by (simp add: cl-pneg-eq)
   moreover
   have (~ (S ⊨ ¬ ~ φ)) = (S ⊨ ~ φ)
```

by *simp*
ultimately show *?thesis* **by** *simp*
next
assume *b*: $\tilde{}$ ($\exists$ ψ. ($\neg$ ψ) = φ)
hence ($\sim'$ φ) = ($\neg$ φ) **by** *simp*
hence ($\sim$ φ) = ($\neg$ φ) **by** (*simp add*: *cl-pneg-eq*)
moreover
have ($\tilde{}$ ($S \models \varphi$)) = ($S \models \neg$ φ) **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

lemma *pneg-FL*: $\forall \psi \in FL(\varphi)$. ($\sim$ ψ) $\in FL(\varphi)$
proof –
have $\forall$ $\psi \in FL(\varphi)$. ($\sim'$ ψ) $\in FL(\varphi)$
by (*induct* φ, (*auto*|*fastsimp*)+)
thus *?thesis* **by** (*simp add*: *cl-pneg-eq*)
qed

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** :→ *24*)

abbreviation *cl-lift-vdash* :: *'a cl-form list* ⇒ *'a cl-form* ⇒ *bool* (**infix** :⊢ *10*)
where
(Γ :⊢ φ) ≡ (⊢ Γ :→ φ)

abbreviation *cl-mod* :: *'a cl-form set* ⇒ *'a set* (†-) **where**
†Γ ≡ {*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* ψ)

and B: $\varphi \in FL(\psi)$
shows $(\dagger\Gamma \models \varphi) = (list\ \Gamma :\vdash \varphi)$
and $(\dagger\Gamma \models \sim \varphi) = (list\ \Gamma :\vdash \sim \varphi)$
using A B
proof$(induct\ \varphi)$
— Propositional case:
fix $a :: 'a$
assume $P\#\ a \in FL\ \psi$
with A *finite-FL neg-pneg-sem-eq*
coincidence [**where** $P{=}\%\ \varphi.\ \dagger\Gamma \models \varphi$]
have $(P\#\ a \in \Gamma) = (list\ \Gamma :\vdash P\#\ a)$
and $\heartsuit$: $(\dagger\Gamma \models P\#\ a) = (list\ \Gamma :\vdash P\#\ a)$
$\Longrightarrow (\dagger\Gamma \models \sim P\#\ a) = (list\ \Gamma :\vdash \sim P\#\ a)$
by *blast+*
thus $(\dagger\Gamma \models P\#\ a) = (list\ \Gamma :\vdash P\#\ a)$
by *fastsimp*
with $\heartsuit$ **show** $(\dagger\Gamma \models \sim P\#\ a) = (list\ \Gamma :\vdash \sim P\#\ a)$
by *fastsimp*

next
— Bottom case – similar to the propositional case:
assume $\bot \in FL\ \psi$
with A *finite-FL neg-pneg-sem-eq*
coincidence [**where** $P{=}\%\ \varphi.\ \dagger\Gamma \models \varphi$]
have $(\bot \in \Gamma) = (list\ \Gamma :\vdash \bot)$
and $\clubsuit$: $(\dagger\Gamma \models \bot) = (list\ \Gamma :\vdash \bot)$
$\Longrightarrow (\dagger\Gamma \models \sim \bot) = (list\ \Gamma :\vdash \sim \bot)$
by *blast+*
with A *Atoms-def* [**where** $\Phi{=}FL\ \psi$]
show $(\dagger\Gamma \models \bot) = (list\ \Gamma :\vdash \bot)$
by (*simp add*: *mem-def*)
with $\clubsuit$ **show** $(\dagger\Gamma \models \sim \bot) = (list\ \Gamma :\vdash \sim \bot)$
by *fastsimp*

next
— Last case: implication is the most challenging
fix $\varphi\ \chi :: 'a\ cl\text{-}form$
assume $\star$: $(\varphi \rightarrow \chi) \in FL\ \psi$
and $[\![\ \Gamma \in At\ (FL\ \psi);\ \varphi \in FL\ \psi\ ]\!]$
$\Longrightarrow (\dagger\Gamma \models \varphi) = (list\ \Gamma :\vdash \varphi)$
and $[\![\ \Gamma \in At\ (FL\ \psi);\ \varphi \in FL\ \psi\ ]\!]$
$\Longrightarrow (\dagger\Gamma \models \sim \varphi) = (list\ \Gamma :\vdash \sim \varphi)$
and $[\![\ \Gamma \in At\ (FL\ \psi);\ \chi \in FL\ \psi\ ]\!]$
$\Longrightarrow (\dagger\Gamma \models \chi) = (list\ \Gamma :\vdash \chi)$
with A

```
    imp-closed-FL[where φ=ψ
                  and ψ=φ
                  and χ=χ]
have
     c1: (†Γ ⊨ φ) = (list Γ :⊢ φ)
  and c2: (†Γ ⊨ ∼ φ) = (list Γ :⊢ ∼ φ)
  and c3: (†Γ ⊨ χ) = (list Γ :⊢ χ)
    by fastsimp+

— We will show that in three cases, which exhaust
— all possibility, the conclusion follows.
show (†Γ ⊨ φ → χ) = (list Γ :⊢ φ → χ)
proof –
 { assume †Γ ⊨ χ
    with c3 lift-weaken [where Γ=list Γ]
    have list Γ :⊢ φ → χ
     and †Γ ⊨ φ → χ by simp+
    hence ?thesis by simp }
 moreover
 { assume ~ (†Γ ⊨ φ)
   moreover
   with c2 neg-pneg-sem-eq
   have list Γ :⊢ ∼ φ by fastsimp
   with pneg-negimpII
        lift [where Γ=list Γ]
        lift-mp [where Γ=list Γ]
    have list Γ :⊢ ¬ φ
      by blast
    with neg-elim
         lift [where Γ=list Γ]
         lift-mp [where Γ=list Γ]
    have list Γ :⊢ φ → χ
      by blast
    ultimately have ?thesis by fastsimp }
 moreover
 { assume a: †Γ ⊨ φ
      and b: ~ (†Γ ⊨ χ)
   — We proceed by reductio ad absurdem
   { assume list Γ :⊢ φ → χ
     with a c1 lift-mp [where Γ=list Γ]
     have list Γ :⊢ χ by blast
     with c3 b have False by simp }
   with a b have ?thesis by fastsimp }
 ultimately show ?thesis by fast
qed
```

with $\star$ *A finite-FL neg-pneg-sem-eq*
coincidence [**where** P=% φ. $\dagger\Gamma \vDash \varphi$]
show $(\dagger\Gamma \vDash \sim (\varphi \rightarrow \chi)) = (\mathit{list}\ \Gamma :\vdash \sim (\varphi \rightarrow \chi))$
by *blast*
qed

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 $\exists\ S.\ \tilde{}\ (S \vDash \psi)$
using *dnp*
proof –
from *dnp* **have** $\tilde{}\ ([] :\vdash \psi)$
by *simp*
hence $\tilde{}\ (\mathit{list}\ \{\} :\vdash \psi)$
by (*simp add*: *empty-set-list*)
with *little-lindy* [**where** Φ=*FL* ψ
and Γ={}]
finite-FL [**where** φ=ψ]
pneg-FL [**where** φ=ψ]
have $\exists\Gamma.\ At\ (FL\ \psi)\ \Gamma \wedge \tilde{}\ (\mathit{list}\ \Gamma :\vdash \psi)$
by *fastsimp*
from *this* **obtain** Γ **where** $At\ (FL\ \psi)\ \Gamma \wedge \tilde{}\ (\mathit{list}\ \Gamma :\vdash \psi)$
by *fast*
moreover have $\psi \in FL\ \psi$
by (*induct* ψ) *simp-all*
moreover note *henkin-truth* [**where** ψ=ψ **and** φ=ψ]
mem-def [**where** x=Γ **and** S=$At\ (FL\ \psi)$]
ultimately show *?thesis* **by** *fastsimp*
qed

lemma *cl-equiv*: $(\vdash \psi) = (\forall\, S.\ S \vDash \psi)$
using *cl-soundness cl-completeness*
by *blast*

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*!]: $\forall\, S.\ S \vDash \psi \Longrightarrow\ \vdash \psi$
using *cl-equiv*

by *blast*

lemma $\vdash ((\psi \to \varphi) \to \psi) \to \psi$
by *fastsimp*

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 ($\vdash$ - [*20*] *20*) **and**
Classic.cl-ClassAx.Atoms (*At*) **and**
Classic.cl-ClassAx.lift-imp (**infix** :$\to$ *24*) **and**
cl-lift-vdash (**infix** :$\vdash$ *10*) **and**
Classic.cl-ClassAx.pneg (~ - [*40*] *40*) **and**
cl-pneg (~′ - [*40*] *40*) **and**
cl-mod (†-)

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

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

lemma (**in** *ClassAx*) *cl-translate*: $\varphi \in CL \Longrightarrow \vdash$ *cltr* φ
by (*induct set*: *CL*,
(*fastsimp intro*: *ax1 ax2 ax3 mp*)+)

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.

$$
\begin{array}{l}
\textbf{fun}\ \textit{evil-eval} :: ('a,'b)\ \textit{evil-world set} \\
\qquad\qquad\qquad \Rightarrow ('a,'b)\ \textit{evil-world} \\
\qquad\qquad\qquad\quad \Rightarrow ('a,'b)\ \textit{evil-form} \\
\qquad\qquad\qquad\qquad \Rightarrow \textit{bool}\ (\text{-,-} \models \text{-}\ \textit{50})\ \textbf{where} \\
\quad (\text{-},(a,\text{-}) \models P\#\ p) = (p \in a) \\
\mid\ (\text{-,-} \models \bot) = \textit{False} \\
\mid\ (\Omega,(a,A) \models \varphi \to \psi) = \\
\qquad ((\Omega,(a,A) \models \varphi) \longrightarrow (\Omega,(a,A) \models \psi)) \\
\mid\ (\Omega,(a,A) \models \Box\ X\ \varphi) = \\
\qquad (\forall\,(b,B) \in \Omega.\ (\forall\,\chi \in A(X).\ b \vDash \chi) \\
\qquad\qquad\qquad \longrightarrow\ \Omega,(b,B) \models \varphi) \\
\mid\ (\Omega,(a,A) \models \odot\ X) = (\forall\,\chi \in A(X).\ a \vDash \chi) \\
\mid\ (\Omega,(a,A) \models [-]\ X\ \varphi) = \\
\qquad (\forall\,(b,B) \in \Omega.\ a = b \\
\qquad\quad \longrightarrow B(X) \subseteq A(X) \\
\qquad\quad\ \longrightarrow\ \Omega,(b,B) \models \varphi) \\
\mid\ (\Omega,(a,A) \models [+]\ X\ \varphi) = \\
\qquad (\forall\,(b,B) \in \Omega.\ a = b \\
\qquad\quad \longrightarrow B(X) \supseteq A(X) \\
\qquad\quad\ \longrightarrow\ \Omega,(b,B) \models \varphi)
\end{array}
$$

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

$$
\begin{array}{l}
\textbf{record}\ ('w,'a,'b)\ \textit{evil-kripke} = \\
\quad W :: 'w\ \textit{set} \\
\quad V :: 'w \Rightarrow 'a \Rightarrow \textit{bool} \\
\quad PP :: 'b \Rightarrow 'w\ \textit{set} \\
\quad RB :: 'b \Rightarrow ('w * 'w)\ \textit{set} \\
\quad RBB :: 'b \Rightarrow ('w * 'w)\ \textit{set} \\
\quad RBBI :: 'b \Rightarrow ('w * 'w)\ \textit{set}
\end{array}
$$

$$
\begin{array}{l}
\textbf{fun}\ \textit{evil-modal-eval} :: ('w,'a,'b)\ \textit{evil-kripke} \\
\qquad\qquad\qquad\qquad \Rightarrow 'w \\
\qquad\qquad\qquad\qquad\quad \Rightarrow ('a,'b)\ \textit{evil-form} \\
\qquad\qquad\qquad\qquad\qquad \Rightarrow \textit{bool}\ (\text{-,-} \Vdash \text{-}\ \textit{50})\ \textbf{where} \\
\quad (M,w \Vdash P\#\ p) = (p \in V(M)(w)) \\
\mid\ (\text{-,-} \Vdash \bot) = \textit{False} \\
\mid\ (M,w \Vdash \varphi \to \psi) = \\
\qquad\quad ((M,w \Vdash \varphi) \longrightarrow (M,w \Vdash \psi)) \\
\mid\ (M,w \Vdash\ \Box\ X\ \varphi) = \\
\qquad\quad (\forall\ v \in W(M).\ (w,v) \in RB(M)(X)
\end{array}
$$

```
                 ⟶  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: $(\odot\ X \rightarrow (\Box\ X\ \varphi) \rightarrow \varphi) \in$ *EviL* |

— If X's evidence is sound,
— then any subset of it she can consider must be sound too
evil-ax12: $(\odot\ X \rightarrow [-]\ X\ (\odot\ X)) \in$ *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: $(\varphi \rightarrow [+]\ X\ (\langle - \rangle\ X\ \varphi)) \in$ *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: $(\varphi \rightarrow [-]\ X\ (\langle + \rangle\ \ X\ \varphi)) \in$ *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: $((\Box\ X\ (\varphi \rightarrow \psi)) \rightarrow (\Box\ X\ \varphi) \rightarrow \Box\ X\ \psi) \in$ *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: $(([-]\ X\ (\varphi \rightarrow \psi)) \rightarrow ([-]\ X\ \varphi) \rightarrow [-]\ X\ \psi) \in$ *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: $(([+]\ X\ (\varphi \rightarrow \psi)) \rightarrow ([+]\ X\ \varphi) \rightarrow [+]\ X\ \psi) \in$ *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: $[\![(\varphi \to \psi) \in EviL; \varphi \in EviL]\!] \Longrightarrow \psi \in EviL$

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

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

theorem *evil-soundness*: $\vdash \varphi \Longrightarrow \forall\ (a,A) \in \Omega.\ \Omega,(a,A) \models \varphi$
by (*induct set*: *EviL*, (*simp add*: *Ball-def*|*blast*)+)

theorem *evil-consistency*: ~ $(\vdash \bot)$
proof –
let $?\Omega=\{(\{\},(\%b\ \varphi.\ False))\}$
have ~$(\forall\ (a,A) \in ?\Omega.\ ?\Omega,(a,A) \models \bot)$ **by** *simp*
with *evil-soundness* [**where** $\Omega=?\Omega$ **and** $\varphi=\bot$]
show *?thesis* **by** *fastsimp*
qed

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 qed (*fastsimp intro*: *EviL.intros*)+

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* (-[-'/-] [300, 0, 0] 300) **where**
$(P\#\ a)[\varphi/\psi] = (if\ ((P\#\ a) = \varphi)\ then\ \psi\ else\ (P\#\ a))$
$|\ \bot[\varphi/\psi] = (if\ (\bot = \varphi)\ then\ \psi\ else\ \bot)$

| $(\odot\ X)[\varphi/\psi]$ = $(if\ ((\odot\ X) = \varphi)\ then\ \psi\ else\ (\odot\ X))$
| $(\delta \to \kappa)[\varphi/\psi]$ = $(if\ ((\delta \to \kappa) = \varphi)\ then\ \psi$
$else\ (\delta[\varphi/\psi] \to \kappa[\varphi/\psi]))$
| $(\Box\ X\ \kappa)[\varphi/\psi]$ = $(if\ ((\Box\ X\ \kappa) = \varphi)\ then\ \psi$
$else\ (\Box\ X\ (\kappa[\varphi/\psi])))$
| $([-]\ X\ \kappa)[\varphi/\psi]$ = $(if\ (([-]\ X\ \kappa) = \varphi)\ then\ \psi$
$else\ ([-]\ X\ (\kappa[\varphi/\psi])))$
| $([+]\ X\ \kappa)[\varphi/\psi]$ = $(if\ (([+]\ X\ \kappa) = \varphi)\ then\ \psi$
$else\ ([+]\ X\ (\kappa[\varphi/\psi])))$

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

abbreviation *cl-iff* ::
$['a$ *cl-form*, $'a$ *cl-form*$] \Rightarrow\ 'a$ *cl-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 –
have $\vdash_{CL} (P\#_{CL}\ \varphi) \leftrightarrow_{CL} (P\#_{CL}\ \varphi)$ **by** *fastsimp*
with *evil-ClassAx.cl-translate*
[**where** φ=$(P\#_{CL}\ \varphi) \leftrightarrow_{CL} (P\#_{CL}\ \varphi)$]
show *?thesis* **by** *simp*
qed

lemma *evil-eq-symm* [*sym*]: $\vdash \varphi \leftrightarrow \psi \Longrightarrow\ \vdash \psi \leftrightarrow \varphi$
proof –
assume *eq*: $\vdash \varphi \leftrightarrow \psi$
let $?\vartheta = ((P\#_{CL}\ \varphi) \leftrightarrow_{CL} (P\#_{CL}\ \psi))$
$\to_{CL} ((P\#_{CL}\ \psi) \leftrightarrow_{CL} (P\#_{CL}\ \varphi))$
have $?\vartheta \in CL$ **by** *fastsimp*
with *evil-ClassAx.cl-translate* [**where** φ=$?\vartheta$]
have $\vdash (\varphi \leftrightarrow \psi) \to (\psi \leftrightarrow \varphi)$ **by** *simp*
with *evil-mp eq* **show** *?thesis* **by** *blast*
qed

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

assume A: $\vdash \varphi \leftrightarrow \psi$
  and B: $\vdash \psi \leftrightarrow \chi$
let $?\vartheta = ((P\#_{CL}\ \varphi) \leftrightarrow_{CL} (P\#_{CL}\ \psi))$
  $\rightarrow_{CL} ((P\#_{CL}\ \psi) \leftrightarrow_{CL} (P\#_{CL}\ \chi))$
  $\rightarrow_{CL} ((P\#_{CL}\ \varphi) \leftrightarrow_{CL} (P\#_{CL}\ \chi))$
have $?\vartheta \in CL$ **by** *fastsimp*
with *evil-ClassAx.cl-translate* [**where** $\varphi=?\vartheta$]
have $\vdash (\varphi \leftrightarrow \psi) \rightarrow (\psi \leftrightarrow \chi) \rightarrow (\varphi \leftrightarrow \chi)$ **by** *simp*
with *evil-mp A B* **show** *?thesis* **by** *blast*
qed

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*: $\vdash \varphi \leftrightarrow \psi \Longrightarrow \vdash \varphi \rightarrow \psi$
proof –
assume *eq*: $\vdash \varphi \leftrightarrow \psi$
let $?\vartheta = ((P\#_{CL}\ \varphi) \leftrightarrow_{CL} (P\#_{CL}\ \psi)) \rightarrow_{CL}$
  $(P\#_{CL}\ \varphi) \rightarrow_{CL} (P\#_{CL}\ \psi)$
have $?\vartheta \in CL$ **by** *fastsimp*
with *evil-ClassAx.cl-translate* [**where** $\varphi=?\vartheta$]
have $\vdash (\varphi \leftrightarrow \psi) \rightarrow \varphi \rightarrow \psi$ **by** *simp*
with *evil-mp eq* **show** *?thesis* **by** *blast*
qed

lemma *evil-eq-mp*: $\vdash \varphi \leftrightarrow \psi \Longrightarrow \vdash \varphi \Longrightarrow \vdash \psi$
proof –
assume *eq*: $\vdash \varphi \leftrightarrow \psi$ **and** *hyp*: $\vdash \varphi$
with *evil-eq-weaken* **have** $\vdash \varphi \rightarrow \psi$ **by** *fast*
with *evil-mp hyp* **show** *?thesis* **by** *fast*
qed

lemma *evil-eq-intro*: $\vdash \varphi \rightarrow \psi \Longrightarrow \vdash \psi \rightarrow \varphi \Longrightarrow \vdash \varphi \leftrightarrow \psi$
proof –
assume A: $\vdash \varphi \rightarrow \psi$
  and B: $\vdash \psi \rightarrow \varphi$
let $?\vartheta=(P\#_{CL}\ \varphi \rightarrow_{CL} P\#_{CL}\ \psi)$
  $\rightarrow_{CL} (P\#_{CL}\ \psi \rightarrow_{CL} P\#_{CL}\ \varphi)$
  $\rightarrow_{CL} (P\#_{CL}\ \varphi \leftrightarrow_{CL} P\#_{CL}\ \psi)$
have $?\vartheta \in CL$ **by** *fastsimp*
with *evil-ClassAx.cl-translate* [**where** $\varphi=?\vartheta$]
have $\vdash (\varphi \rightarrow \psi) \rightarrow (\psi \rightarrow \varphi) \rightarrow (\varphi \leftrightarrow \psi)$ **by** *simp*
with *A B evil-mp* **show** *?thesis* **by** *blast*
qed

lemma *evil-contrapose*:

$\vdash \varphi \rightarrow \psi \Longrightarrow \vdash \neg\, \psi \rightarrow \neg\, \varphi$
proof –
  let *?ϑ*$=(P\#_{CL}\ \varphi \rightarrow_{CL} P\#_{CL}\ \psi)$
      $\rightarrow_{CL} (\neg_{CL}\ (P\#_{CL}\ \psi) \rightarrow_{CL} \neg_{CL}\ (P\#_{CL}\ \varphi))$
  have *?ϑ* $\in$ *CL* **by** *fastsimp*
  with *evil-ClassAx.cl-translate* [**where** φ=*?ϑ*]
  have $\vdash (\varphi \rightarrow \psi) \rightarrow (\neg\, \psi \rightarrow \neg\, \varphi)$ **by** *simp*
  moreover assume $\vdash \varphi \rightarrow \psi$
  moreover note *evil-mp*
  ultimately show *?thesis* **by** *blast*
qed

notation
  evil-ClassAx.lift-imp (**infix** $:\rightarrow$ *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 –
  assume $\vdash \varphi \rightarrow \psi$
  with *evil-B-nec* **have** $\vdash \Box\ X\ (\varphi \rightarrow \psi)$ **by** *fast*
  with *evil-ax15* [**where** X=X **and** φ=φ **and** ψ=ψ]
     evil-mp **show** *?thesis* **by** *fast*
qed

lemma *evil-lift-ax15*:
   assumes *notnil*: $\varphi s \neq []$
    shows $\vdash \Box\ X\ (\varphi s :\rightarrow \psi)$
        $\rightarrow ((map\ (\lambda\ \varphi.\ \Box\ X\ \varphi)\ \varphi s) :\rightarrow \Box\ X\ \psi)$
using *notnil*
proof (*induct* φs)
 case *Nil* **thus** *?case* **by** *fast*
 next case (*Cons* φ φs)
  note *ind-hyp* = *this*
  show *?case*
  proof *cases*
   assume $\varphi s = []$
    with *evil-ax15* [**where** X=X]
     show *?case* **by** *simp*
   next
   let *?A* $= \Box\ X\ ((\varphi \# \varphi s) :\rightarrow \psi)$
   and *?B* $= \Box\ X\ \varphi$

```
    and ?C = □ X (φs :→ ψ)
    and ?D = ((map (λ φ. □ X φ) φs) :→ □ X ψ)
    assume notnil: φs ≠ []
      with ind-hyp
           evil-ClassAx.lift [where Γ=[?A]]
        have map: [?A] :⊢ ?C → ?D by fast
      from evil-ax15 [where X=X]
        have [?A] :⊢ ?B → ?C by simp
      with map
           evil-ClassAx.lift-hs [where Γ=[?A]
                                 and φ=?B
                                 and ψ=?C
                                 and χ=?D]
        show ?case by simp
  qed
qed

lemma evil-B-lift-map:
 assumes seq: φs :⊢ ψ
   shows (map (λ φ. □ X φ) φs) :⊢ □ X ψ
using seq
proof (induct φs)
  case Nil with evil-B-nec [where X=X]
    show ?case by simp
  next case (Cons φ φs)
    with evil-B-nec [where X=X and φ=(φ # φs) :→ ψ]
         evil-mp
         evil-lift-ax15 [where X=X
                         and φs=φ # φs
                         and ψ=ψ]
    show ?case by simp
qed

lemma evil-DB-map: ⊢ φ → ψ ⟹ ⊢ ◇ X φ → ◇ X ψ
proof –
  assume ⊢ φ → ψ
  with evil-contrapose have ⊢ ¬ ψ → ¬ φ .
  with evil-B-map have ⊢ □ X (¬ ψ) → □ X (¬ φ) .
  with evil-contrapose show ?thesis .
qed

lemma evil-BB-map: ⊢ φ → ψ ⟹ ⊢ [−] X φ → [−] X ψ
proof –
  assume ⊢ φ → ψ
  with evil-BB-nec have ⊢ [−] X (φ → ψ) by fast
```

```
  with evil-ax16 [where X=X and φ=φ and ψ=ψ]
      evil-mp show ?thesis by fast
qed

lemma evil-lift-ax16:
   assumes notnil: φs ≠ []
     shows ⊢ [−] X (φs :→ ψ)
              → ((map (λ φ. [−] X φ) φs) :→ [−] X ψ)
using notnil
proof (induct φs)
 case Nil thus ?case by fast
 next case (Cons φ φs)
  note ind-hyp = this
  show ?case
  proof cases
    assume φs = []
      with evil-ax16 [where X=X]
        show ?case by simp
    next
    let ?A = [−] X ((φ # φs) :→ ψ)
    and ?B = [−] X φ
    and ?C = [−] X (φs :→ ψ)
    and ?D = ((map (λ φ. [−] X φ) φs) :→ [−] X ψ)
    assume notnil: φs ≠ []
      with ind-hyp
           evil-ClassAx.lift [where Γ=[?A]]
        have map: [?A] :⊢ ?C → ?D by fast
      from evil-ax16 [where X=X]
        have [?A] :⊢ ?B → ?C by simp
      with map
           evil-ClassAx.lift-hs [where Γ=[?A]
                                   and φ=?B
                                   and ψ=?C
                                   and χ=?D]
        show ?case by simp
  qed
qed

lemma evil-BB-lift-map:
 assumes seq: φs :⊢ ψ
   shows (map (λ φ. [−] X φ) φs) :⊢ [−] X ψ
using seq
proof (induct φs)
  case Nil with evil-BB-nec [where X=X]
    show ?case by simp
```

```
  next case (Cons φ φs)
    with evil-BB-nec [where X=X and φ=(φ # φs) :→ ψ]
         evil-mp
         evil-lift-ax16 [where X=X
                          and φs=φ # φs
                          and ψ=ψ]
    show ?case by simp
qed

lemma evil-DBB-map: ⊢ φ → ψ ⟹ ⊢ ⟨−⟩ X φ → ⟨−⟩ X ψ
proof −
  assume ⊢ φ → ψ
  with evil-contrapose have ⊢ ¬ ψ → ¬ φ .
  with evil-BB-map have ⊢ [−] X (¬ ψ) → [−] X (¬ φ) .
  with evil-contrapose show ?thesis .
qed

lemma evil-BBI-map: ⊢ φ → ψ ⟹ ⊢ [+] X φ → [+] X ψ
proof −
  assume ⊢ φ → ψ
  with evil-BBI-nec have ⊢ [+] X (φ → ψ) by fast
  with evil-ax17 [where X=X and φ=φ and ψ=ψ]
       evil-mp show ?thesis by fast
qed

lemma evil-lift-ax17:
   assumes notnil: φs ≠ []
     shows ⊢ [+] X (φs :→ ψ)
              → ((map (λ φ. [+] X φ) φs) :→ [+] X ψ)
using notnil
proof (induct φs)
 case Nil thus ?case by fast
 next case (Cons φ φs)
  note ind-hyp = this
  show ?case
  proof cases
    assume φs = []
      with evil-ax17 [where X=X]
        show ?case by simp
    next
    let ?A = [+] X ((φ # φs) :→ ψ)
    and ?B = [+] X φ
    and ?C = [+] X (φs :→ ψ)
    and ?D = ((map (λ φ. [+] X φ) φs) :→ [+] X ψ)
    assume notnil: φs ≠ []
```

```
    with ind-hyp
        evil-ClassAx.lift [where Γ=[?A]]
     have map: [?A] :⊢ ?C → ?D by fast
    from evil-ax17 [where X=X]
     have [?A] :⊢ ?B → ?C by simp
    with map
        evil-ClassAx.lift-hs [where Γ=[?A]
                              and φ=?B
                              and ψ=?C
                              and χ=?D]
     show ?case by simp
  qed
qed

lemma evil-BBI-lift-map:
 assumes seq: φs :⊢ ψ
   shows (map (λ φ. [+] X φ) φs) :⊢ [+] X ψ
using seq
proof (induct φs)
  case Nil with evil-BBI-nec [where X=X]
   show ?case by simp
  next case (Cons φ φs)
   with evil-BBI-nec [where X=X and φ=(φ # φs) :→ ψ]
        evil-mp
        evil-lift-ax17 [where X=X
                        and φs=φ # φs
                        and ψ=ψ]
   show ?case by simp
qed

lemma evil-DBBI-map: ⊢ φ → ψ ⟹ ⊢ ⟨+⟩ X φ → ⟨+⟩ X ψ
proof –
  assume ⊢ φ → ψ
  with evil-contrapose have ⊢ ¬ ψ → ¬ φ .
  with evil-BBI-map have ⊢ [+] X (¬ ψ) → [+] X (¬ φ) .
  with evil-contrapose show ?thesis .
qed

lemma evil-sub:
  assumes eq: ⊢ φ ↔ ψ
  shows ⊢ χ ↔ χ[φ/ψ]
using eq
proof (induct χ, (fastsimp intro: evil-eq-refl)+)
  — Most cases are delt with automatically;
  — we are left with implication and the three boxes
```

case (*E-Imp* δ κ)
  hence A: ($\vdash \delta \leftrightarrow \delta[\varphi/\psi]$)
    and B: ($\vdash \kappa \leftrightarrow \kappa[\varphi/\psi]$) **by** *fast+*
  — This case follows from a lengthy tautology
  let $?\vartheta$=$(P\#_{CL}\ \delta \leftrightarrow_{CL} P\#_{CL}\ (\delta[\varphi/\psi]))$
    $\rightarrow_{CL} (P\#_{CL}\ \kappa \leftrightarrow_{CL} P\#_{CL}\ (\kappa[\varphi/\psi]))$
    $\rightarrow_{CL} ((P\#_{CL}\ \delta \rightarrow_{CL} P\#_{CL}\ \kappa)$
      $\leftrightarrow_{CL} (P\#_{CL}\ (\delta[\varphi/\psi]) \rightarrow_{CL} P\#_{CL}\ (\kappa[\varphi/\psi])))$
  have $?\vartheta \in CL$ **by** *fastsimp*
  with *evil-ClassAx.cl-translate* [**where** φ=$?\vartheta$]
  have $\vdash (\delta \leftrightarrow \delta[\varphi/\psi]) \rightarrow (\kappa \leftrightarrow \kappa\ [\varphi/\psi])$
    $\rightarrow ((\delta \rightarrow \kappa) \leftrightarrow (\delta[\varphi/\psi] \rightarrow \kappa[\varphi/\psi]))$ **by** *simp*
  with A *evil-mp* [**where** φ=$\delta \leftrightarrow \delta[\varphi/\psi]$]
    B *evil-mp* [**where** φ=$\kappa \leftrightarrow \kappa[\varphi/\psi]$] **have**
  $\vdash (\delta \rightarrow \kappa) \leftrightarrow (\delta[\varphi/\psi] \rightarrow \kappa[\varphi/\psi])$ **by** *blast*
  with *eq evil-eq-refl* **show** *?case* **by** *fastsimp*

— The next three cases are all basically the same
next case (*E-B* X χ)
  hence A: $\vdash \chi \leftrightarrow \chi[\varphi/\psi]$ **by** *fast*
  from A *evil-eq-weaken evil-B-map*
    have $\vdash \Box\ X\ \chi \rightarrow \Box\ X\ (\chi[\varphi/\psi])$ **by** *fast*
  moreover from A *evil-eq-symm evil-eq-weaken evil-B-map*
    have $\vdash \Box\ X\ (\chi[\varphi/\psi]) \rightarrow \Box\ X\ \chi$ **by** *fast*
  moreover note *evil-eq-intro*
  ultimately have $\vdash \Box\ X\ \chi \leftrightarrow \Box\ X\ (\chi[\varphi/\psi])$ **by** *fast*
  with *eq* **show** *?case* **by** *fastsimp*
next case (*E-BB* X χ)
  hence A: $\vdash \chi \leftrightarrow \chi[\varphi/\psi]$ **by** *fast*
  from A *evil-eq-weaken evil-BB-map*
    have $\vdash [-]\ X\ \chi \rightarrow [-]\ X\ (\chi[\varphi/\psi])$ **by** *fast*
  moreover from A *evil-eq-symm evil-eq-weaken evil-BB-map*
    have $\vdash [-]\ X\ (\chi[\varphi/\psi]) \rightarrow [-]\ X\ \chi$ **by** *fast*
  moreover note *evil-eq-intro*
  ultimately have $\vdash [-]\ X\ \chi \leftrightarrow [-]\ X\ (\chi[\varphi/\psi])$ **by** *fast*
  with *eq* **show** *?case* **by** *fastsimp*
next case (*E-BBI* X χ)
  hence A: $\vdash \chi \leftrightarrow \chi[\varphi/\psi]$ **by** *fast*
  from A *evil-eq-weaken evil-BBI-map*
    have $\vdash [+]\ X\ \chi \rightarrow [+]\ X\ (\chi[\varphi/\psi])$ **by** *fast*
  moreover from A *evil-eq-symm evil-eq-weaken evil-BBI-map*
    have $\vdash [+]\ X\ (\chi[\varphi/\psi]) \rightarrow [+]\ X\ \chi$ **by** *fast*
  moreover note *evil-eq-intro*
  ultimately have $\vdash [+]\ X\ \chi \leftrightarrow [+]\ X\ (\chi[\varphi/\psi])$ **by** *fast*
  with *eq* **show** *?case* **by** *fastsimp*

qed

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* $\Rightarrow$ $('a,'b)$ *evil-form set* $(\Downarrow)$
where
$\Downarrow(P\#\ p) = \{\}$
$|\ \Downarrow(\bot) = \{\}$
$|\ \Downarrow(\odot\ X) = \{\}$
$|\ \Downarrow(\varphi \rightarrow \psi) = \{\varphi, \psi\} \cup \Downarrow(\varphi) \cup \Downarrow(\psi)$
$|\ \Downarrow(\Box\ X\ \varphi) = \{\varphi\} \cup\ \Downarrow(\varphi)$
$|\ \Downarrow([-]\ X\ \varphi) = \{\varphi\} \cup\ \Downarrow(\varphi)$
$|\ \Downarrow([+]\ X\ \varphi) = \{\varphi\} \cup\ \Downarrow(\varphi)$

— Here's a series of obvious inequalities we shall reuse
lemma *evil-limp-neq*[*intro*]: $\forall\ \chi.\ (\psi \rightarrow \chi) \neq \psi$
by (*induct* ψ, *simp-all*)

lemma *evil-rimp-neq*[*intro*]: $\forall\ \psi.\ (\psi \rightarrow \chi) \neq \chi$
by (*induct* χ, *simp-all*)

lemma *evil-B-neq*[*intro*]: $(\Box\ X\ \varphi) \neq \varphi$
by (*induct* φ, *fastsimp+*)

lemma *evil-BB-neq*[*intro*]: $([-]\ X\ \varphi) \neq \varphi$
by (*induct* φ, *fastsimp+*)

lemma *evil-BBI-neq*[*intro*]: $([+]\ X\ \varphi) \neq \varphi$
by (*induct* φ, *fastsimp+*)

lemma *evil-not-neq*[*intro*]: $(\neg\ \varphi) \neq \varphi$
by (*induct* φ, *fastsimp+*)

— Here's a series of deconstruction lemmas
lemma *evil-psform-limp-elim*[*intro*]:
$(\delta \rightarrow \kappa) \in \Downarrow \psi \Longrightarrow \delta \in \Downarrow \psi$
by (*induct* ψ, *fastsimp+*)

lemma *evil-psform-rimp-elim*[*intro*]:
$(\delta \to \kappa) \in \Downarrow \psi \Longrightarrow \kappa \in \Downarrow \psi$
by (*induct* ψ, *fastsimp*+)

lemma *evil-psform-B-elim*[*intro*]:
$\Box\ X\ \psi \in \Downarrow \varphi \Longrightarrow \psi \in \Downarrow \varphi$
by (*induct* φ, *fastsimp*+)

lemma *evil-psform-BB-elim*[*intro*]:
$[-]\ X\ \psi \in \Downarrow \varphi \Longrightarrow \psi \in \Downarrow \varphi$
by (*induct* φ, *fastsimp*+)

lemma *evil-psform-BBI-elim*[*intro*]:
$[+]\ X\ \psi \in \Downarrow \varphi \Longrightarrow \psi \in \Downarrow \varphi$
by (*induct* φ, *fastsimp*+)

— All of the above lemmas are used implicitly by what follows:
lemma *evil-psform-nin* [*intro*!]: $\varphi \notin \Downarrow \varphi$
proof (*induct* φ)
case *E-P* **show** *?case* **by** *simp*
next case *E-Bot* **show** *?case* **by** *simp*
next case *E-PP* **show** *?case* **by** *simp*
next case (*E-Imp* ψ χ) **thus** *?case*
using *evil-limp-neq* [**where** ψ=ψ]
evil-rimp-neq [**where** χ=χ]
by (*simp*,*blast*)
next case (*E-B* X φ) **thus** *?case*
using *evil-B-neq* [**where** X=X **and** φ=φ]
by (*simp*,*blast*)
next case (*E-BB* X φ) **thus** *?case*
using *evil-BB-neq* [**where** X=X **and** φ=φ]
by (*simp*,*blast*)
next case (*E-BBI* X φ) **thus** *?case*
using *evil-BBI-neq* [**where** X=X **and** φ=φ]
by (*simp*,*blast*)
qed

lemma *sub-neq* [*intro*!]:
assumes *sf*: $\psi \in \Downarrow \varphi$
shows $\psi \neq \varphi$
using *sf*
proof –
from *sf* **have** $\psi = \varphi \longrightarrow \varphi \in \Downarrow \varphi$ **by** *auto*
with *evil-psform-nin* **show** *?thesis* **by** *fast*
qed

lemma *sub-nosub* [*intro*]:
  assumes *psub*: $\psi \in \Downarrow \varphi$
    shows $\psi[\varphi/\chi] = \psi$
using *psub*
proof (*induct* ψ)
      case *E-P* **thus** *?case* **by** *fastsimp*
  next case *E-Bot* **thus** *?case* **by** *fastsimp*
  next case *E-PP* **thus** *?case* **by** *fastsimp*
  next case (*E-Imp* δ κ)
   moreover hence $(\delta \to \kappa) \neq \varphi$ **by** *fast*
   ultimately show *?case* **by** (*simp*, *blast*)
  next case (*E-B* X ψ)
   moreover hence $\Box\ X\ \psi \neq \varphi$ **by** *fast*
   ultimately show *?case* **by** (*simp*, *blast*)
  next case (*E-BB* X ψ)
   moreover hence $[-]\ X\ \psi \neq \varphi$ **by** *fast*
   ultimately show *?case* **by** (*simp*, *blast*)
  next case (*E-BBI* X ψ)
   moreover hence $[+]\ X\ \psi \neq \varphi$ **by** *fast*
   ultimately show *?case* **by** (*simp*, *blast*)
qed

lemma *evil-dneg-eq*: $\vdash \neg (\neg \varphi) \leftrightarrow \varphi$
proof –
  let $?\vartheta = (\neg_{CL} (\neg_{CL} (P\#_{CL}\ \varphi)) \leftrightarrow_{CL} P\#_{CL}\ \varphi)$
  have $?\vartheta \in CL$ **by** *fastsimp*
  with *evil-ClassAx.cl-translate* [**where** φ=$?\vartheta$]
  show *?thesis* **by** *simp*
qed

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*: $\vdash \langle + \rangle\ X\ ([-]\ X\ \varphi) \to \varphi$
proof –
  from *evil-ax13* [**where** φ=$\neg \varphi$ **and** X=X]
  moreover have $(\neg \varphi) \in \Downarrow (\neg \neg \varphi)$ **by** *simp*
    with *sub-nosub* **have** $(\neg \varphi)[\neg \neg \varphi / \varphi] = (\neg \varphi)$ **by** *blast*
  moreover have $\bot \in \Downarrow (\neg \neg \varphi)$ **by** *simp*
    with *sub-nosub* **have** $\bot[\neg \neg \varphi / \varphi] = \bot$ **by** *blast*
  moreover note
    evil-sub [**where** φ=$\neg \neg \varphi$

and ψ=φ
and χ=$\neg\ \varphi \rightarrow [+]\ X\ (\langle-\rangle\ X\ (\neg\ \varphi))$]
evil-not-neq [**where** φ=φ]
evil-dneg-eq [**where** φ=φ]
evil-eq-mp
ultimately
have $\vdash \neg\ \varphi \rightarrow [+]\ X\ (\neg\ [-]\ X\ \varphi)$ **by** *auto*
moreover
let $?\vartheta$=$(\neg_{CL}\ (P\#_{CL}\ \varphi) \rightarrow_{CL} P\#_{CL}\ ([+]\ X\ (\neg\ [-]\ X\ \varphi)))$
$\rightarrow_{CL} (\neg_{CL}\ (P\#_{CL}\ ([+]\ X\ (\neg\ [-]\ X\ \varphi))) \rightarrow_{CL} P\#_{CL}\ \varphi)$
have $?\vartheta \in CL$ **by** *fastsimp*
moreover note *evil-ClassAx.cl-translate* [**where** φ=$?\vartheta$]
evil-mp
ultimately show *?thesis* **by** *fastsimp*
qed

lemma *evil-dax14*: $\vdash \langle-\rangle\ X\ ([+]\ X\ \varphi) \rightarrow \varphi$
proof –
from *evil-ax14* [**where** φ=$\neg\ \varphi$ **and** X=X]
moreover have $(\neg\ \varphi) \in \Downarrow (\neg\ \neg\ \varphi)$ **by** *simp*
with *sub-nosub* **have** $(\neg\ \varphi)[\neg\ \neg\ \varphi/\ \varphi] = (\neg\ \varphi)$ **by** *blast*
moreover have $\bot \in \Downarrow (\neg\ \neg\ \varphi)$ **by** *simp*
with *sub-nosub* **have** $\bot[\neg\ \neg\ \varphi/\ \varphi] = \bot$ **by** *blast*
moreover note
evil-sub [**where** φ=$\neg\ \neg\ \varphi$
and ψ=φ
and χ=$\neg\ \varphi \rightarrow [-]\ X\ (\langle+\rangle\ X\ (\neg\ \varphi))$]
evil-not-neq [**where** φ=φ]
evil-dneg-eq [**where** φ=φ]
evil-eq-mp
ultimately
have $\vdash \neg\ \varphi \rightarrow [-]\ X\ (\neg\ [+]\ X\ \varphi)$ **by** *auto*
moreover
let $?\vartheta$=$(\neg_{CL}\ (P\#_{CL}\ \varphi) \rightarrow_{CL} P\#_{CL}\ ([-]\ X\ (\neg\ [+]\ X\ \varphi)))$
$\rightarrow_{CL} (\neg_{CL}\ (P\#_{CL}\ ([-]\ X\ (\neg\ [+]\ X\ \varphi))) \rightarrow_{CL} P\#_{CL}\ \varphi)$
have $?\vartheta \in CL$ **by** *fastsimp*
moreover note *evil-ClassAx.cl-translate* [**where** φ=$?\vartheta$]
evil-mp
ultimately show *?thesis* **by** *fastsimp*
qed

lemma *evil-BBIax8*: $\vdash (\Box\ X\ \varphi) \rightarrow [+]\ X\ (\Box\ X\ \varphi)$
proof –
from *evil-ax8* **have** $\vdash \Diamond\ X\ (\neg\ \varphi) \rightarrow [-]\ X\ (\Diamond\ X\ (\neg\ \varphi))$ **.**
with *evil-DBBI-map*

have $\vdash \langle + \rangle\ X\ (\Diamond\ X\ (\neg\ \varphi)) \to \langle + \rangle\ X\ ([-]\ X\ (\Diamond\ X\ (\neg\ \varphi)))$.
with *evil-ClassAx.hs evil-dax13* [**where** X=X
and φ=$\Diamond\ X\ (\neg\ \varphi)$]
have $\vdash \langle + \rangle\ X\ (\Diamond\ X\ (\neg\ \varphi)) \to \Diamond\ X\ (\neg\ \varphi)$ **by** *blast*
with *evil-mp evil-ax3* [**where** φ=$[+]\ X\ (\neg\ (\Diamond\ X\ (\neg\ \varphi)))$
and ψ=$\Box\ X\ (\neg\ \neg\ \varphi)$]
have $\vdash \Box\ X\ (\neg\ \neg\ \varphi) \to [+]\ X\ (\neg\ \Diamond\ X\ (\neg\ \varphi))$ **by** *blast*
moreover have $\Box\ X\ (\neg\ \neg\ \varphi) \in \Downarrow (\neg\ \Diamond\ X\ (\neg\ \varphi))$ **by** *simp*
with *sub-nosub* **have**
$\Box\ X\ (\neg\ \neg\ \varphi)[\neg\ \Diamond\ X\ (\neg\ \varphi)/\Box\ X\ (\neg\ \neg\ \varphi)] = \Box\ X\ (\neg\ \neg\ \varphi)$
by *blast*
moreover note
evil-sub [**where** φ=$\neg\ \Diamond\ X\ (\neg\ \varphi)$
and ψ=$\Box\ X\ (\neg\ \neg\ \varphi)$
and χ=$\Box\ X\ (\neg\ \neg\ \varphi) \to [+]\ X\ (\neg\ \Diamond\ X\ (\neg\ \varphi))$]
evil-dneg-eq [**where** φ=$\Box\ X\ (\neg\ \neg\ \varphi)$]
evil-eq-mp
ultimately have $\vdash \Box\ X\ (\neg\ \neg\ \varphi) \to [+]\ X\ (\Box\ X\ (\neg\ \neg\ \varphi))$
by *fastsimp*
with
evil-sub [**where** φ=$\neg\ \neg\ \varphi$
and ψ=φ
and χ=$\Box\ X\ (\neg\ \neg\ \varphi) \to [+]\ X\ (\Box\ X\ (\neg\ \neg\ \varphi))$]
evil-dneg-eq [**where** φ=φ]
evil-eq-mp
show *?thesis*
by *fastsimp*
qed

lemma *evil-BBax4*: $\vdash [-]\ X\ \varphi \to \varphi$
— If φ holds no matter what X tries to forget,
— then it must be that φ holds simpliciter
proof –
from *evil-ax13 evil-ax4 evil-ClassAx.hs*
have $\vdash (\neg\ \varphi) \to \langle - \rangle\ X\ (\neg\ \varphi)$ **by** *fast*
moreover have $\varphi \in \Downarrow (\neg\ \neg\ \varphi)$ **by** *simp*
with *sub-nosub* **have** $\varphi[\neg\ \neg\ \varphi/\ \varphi] = \varphi$ **by** *blast*
moreover have $\bot \in \Downarrow (\neg\ \neg\ \varphi)$ **by** *simp*
with *sub-nosub* **have** $\bot[\neg\ \neg\ \varphi/\ \varphi] = \bot$ **by** *blast*
moreover note
evil-sub [**where** φ=$\neg\ \neg\ \varphi$
and ψ=φ
and χ=$(\neg\ \varphi) \to \langle - \rangle\ X\ (\neg\ \varphi)$]
evil-not-neq [**where** φ=φ]
evil-dneg-eq [**where** φ=φ]

evil-eq-mp
ultimately have
$\vdash \neg\, \varphi \to \neg\, [-]\ X\ \varphi$ **by** *auto*
with *evil-ax3 evil-mp*
show *?thesis* **by** *blast*
qed

lemma *evil-BBdax5*: $\vdash \langle - \rangle\ X\ (\langle - \rangle\ X\ \varphi) \to \langle - \rangle\ X\ \varphi$
— If φ is true no matter what X
— tries to forget, then it's true no matter
— what further evidence she disregards
proof –
from *EviL.intros* **have** $\vdash \varphi \to [+]\ X\ (\langle - \rangle\ X\ \varphi)$ **by** *fast*
with *evil-ax5* [**where** X=X]
evil-ClassAx.hs
have $\vdash \varphi \to [+]\ X\ ([+]\ X\ (\langle - \rangle\ X\ \varphi))$ **by** *blast*
with *evil-DBB-map* [**where** X=X] **have**
$\vdash \langle - \rangle\ X\ (\langle - \rangle\ X\ \varphi)$
$\to \langle - \rangle\ X\ (\langle - \rangle\ X\ ([+]\ X\ ([+]\ X\ (\langle - \rangle\ X\ \varphi))))$ **by** *blast*
with *evil-dax14* [**where** X=X
and φ=$[+]\ X\ (\langle - \rangle\ X\ \varphi)$]
evil-DBB-map [**where** X=X]
evil-ClassAx.hs
have $\vdash \langle - \rangle\ X\ (\langle - \rangle\ X\ \varphi) \to \langle - \rangle\ X\ ([+]\ X\ (\langle - \rangle\ X\ \varphi))$ **by** *blast*
with *evil-dax14* [**where** X=X
and φ=$\langle - \rangle\ X\ \varphi$]
evil-DBB-map [**where** X=X]
evil-ClassAx.hs
show *?thesis* **by** *blast*
qed

lemma *evil-BBax5*: $\vdash [-]\ X\ \varphi \to [-]\ X\ ([-]\ X\ \varphi)$
— If φ is true no matter what X
— tries to forget, then it's true no matter
— what further evidence she disregards
proof –
from *evil-BBdax5*
have $\vdash \langle - \rangle\ X\ (\langle - \rangle\ X\ (\neg\, \varphi)) \to \langle - \rangle\ X\ (\neg\, \varphi)$ **.**
moreover have $\bot \in \Downarrow (\neg\, \neg\, \varphi)$ **by** *simp*
hence $\bot[\neg\, \neg\, \varphi /\ \varphi] = \bot$ **by** *fast*
moreover have $\varphi \in \Downarrow ([-]\ X\ (\neg\, \neg\, \varphi))$ **by** *simp*
hence $\varphi \neq [-]\ X\ (\neg\, \neg\, \varphi)$ **by** *fast*
moreover note
evil-sub [**where** φ=$\neg\, \neg\, \varphi$
and ψ=φ

and χ=⟨−⟩ X (⟨−⟩ X (¬ φ)) → ⟨−⟩ X (¬ φ)]
evil-dneg-eq [**where** φ=φ]
evil-eq-mp
ultimately have
⊢ ⟨−⟩ X (¬ ([−] X φ)) → ¬ ([−] X φ) **by** *fastsimp*
moreover have ⊥ ∈ ⇓ (¬ ¬ [−] X φ) **by** *simp*
hence ⊥[¬ ¬ [−] X φ / [−] X φ] = ⊥ **by** *fast*
moreover have (¬ [−] X φ) ∈ ⇓ (¬ ¬ [−] X φ) **by** *simp*
hence (¬ [−] X φ)[¬ ¬ [−] X φ / [−] X φ]
= (¬ [−] X φ) **by** *fast*
moreover note
evil-sub [**where** φ=¬ ¬ [−] X φ
and ψ=[−] X φ
and χ=⟨−⟩ X (¬ [−] X φ) → (¬ [−] X φ)]
evil-dneg-eq [**where** φ=[−] X φ]
evil-eq-mp
ultimately have ⊢ ¬ [−] X ([−] X φ) → ¬ ([−] X φ)
by *fastsimp*
with *evil-ax3 evil-mp* **show** *?thesis* **by** *blast*
qed

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) ∈ $RB(M)(X)$⟧

$\Longrightarrow (v,u) \in RB(M)(X)$
and *prop7*: $(w,v) \in RBB(M)(X)$
$\Longrightarrow ((u,w) \in RB(M)(Y)) = ((u,v) \in RB(M)(Y))$
and *prop8*: $w \in PP(M)(X) \Longrightarrow (w,w) \in RB(M)(X)$

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*: $(w \in PP(M)(X)) = ((w,w) \in RB(M)(X))$

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 (⊥) **and**
CL-Imp (**infixr** → *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 φ) $\leftrightarrow$ [+] X φ
— Further further beliefs are the same as further beliefs
using *evil-ax5* [**where** X=X]
  evil-ax4 [**where** X=X **and** φ=[+] X φ]
  evil-eq-intro
 by *blast*

lemma *evil-BB-eq*: $\vdash$ [−] X ([−] X φ) $\leftrightarrow$ [−] X φ
— To discard beliefs and then discard beliefs again
— is the same as discarding beliefs only once
using *evil-BBax5* [**where** X=X]
  evil-BBax4 [**where** X=X **and** φ=[−] X φ]
  evil-eq-intro
 by *blast*

lemma *evil-eq-neg*: $\vdash$ $\varphi \leftrightarrow \psi \Longrightarrow \vdash \neg\ \varphi \leftrightarrow \neg\ \psi$
proof −
 assume $\vdash$ $\varphi \leftrightarrow \psi$
 moreover
  let *?*ϑ = $((P\#_{CL}\ \varphi) \leftrightarrow_{CL} P\#_{CL}\ \psi) \rightarrow_{CL} (\neg_{CL}\ (P\#_{CL}\ \varphi) \leftrightarrow_{CL} \neg_{CL}\ (P\#_{CL}\ \psi))$
  have *?*$\vartheta \in CL$ **by** *fastsimp*
 moreover note *evil-ClassAx.cl-translate* [**where** φ=*?*ϑ]
   evil-mp
 ultimately show *?thesis* **by** *fastsimp*
qed

lemma *evil-DD-eq*: $\vdash$ $\langle-\rangle$ X ($\langle-\rangle$ X φ) $\leftrightarrow$ $\langle-\rangle$ X φ
proof −
 have $\perp \in \Downarrow (\neg\ \neg\ [-]\ X\ (\neg\ \varphi))$ **by** *simp*
 moreover from *evil-dneg-eq*
  have $\vdash \neg\ \neg\ [-]\ X\ (\neg\ \varphi) \leftrightarrow [-]\ X\ (\neg\ \varphi)$
   by *fast*
 moreover note *evil-sub* [**where** χ=$\langle-\rangle$ X ($\langle-\rangle$ X φ)
   and φ=$\neg\ \neg$ [−] X ($\neg\ \varphi$)
   and ψ=[−] X ($\neg\ \varphi$)]
 ultimately have $\vdash \langle-\rangle\ X\ (\langle-\rangle\ X\ \varphi) \leftrightarrow \neg\ [-]\ X\ ([-]\ X\ (\neg\ \varphi))$ **by** *fastsimp*
 moreover
 from *evil-BB-eq* **have** $\vdash [-]\ X\ ([-]\ X\ (\neg\ \varphi)) \leftrightarrow [-]\ X\ (\neg\ \varphi)$ **.**
 with *evil-eq-neg* **have** $\vdash \neg\ [-]\ X\ ([-]\ X\ (\neg\ \varphi)) \leftrightarrow \langle-\rangle\ X\ \varphi$ **.**
 moreover note *evil-eq-trans*
 ultimately show *?thesis* **by** *blast*

qed

lemma *evil-DDI-eq*: $\vdash \langle+\rangle\ X\ (\langle+\rangle\ X\ \varphi) \leftrightarrow \langle+\rangle\ X\ \varphi$
proof –
  have $\bot \in \Downarrow (\neg\ \neg\ [+]\ X\ (\neg\ \varphi))$ **by** *simp*
  moreover from *evil-dneg-eq*
    have $\vdash \neg\ \neg\ [+]\ X\ (\neg\ \varphi) \leftrightarrow [+]\ X\ (\neg\ \varphi)$
      by *fast*
  moreover note *evil-sub* [**where** χ=$\langle+\rangle\ X\ (\langle+\rangle\ X\ \varphi)$
                            and φ=$\neg\ \neg\ [+]\ X\ (\neg\ \varphi)$
                            and ψ=$[+]\ X\ (\neg\ \varphi)$]
  ultimately have $\vdash \langle+\rangle\ X\ (\langle+\rangle\ X\ \varphi) \leftrightarrow \neg\ [+]\ X\ ([+]\ X\ (\neg\ \varphi))$ **by** *fastsimp*
  moreover
  from *evil-BBI-eq* **have** $\vdash [+]\ X\ ([+]\ X\ (\neg\ \varphi)) \leftrightarrow [+]\ X\ (\neg\ \varphi)$.
  with *evil-eq-neg* **have** $\vdash \neg\ [+]\ X\ ([+]\ X\ (\neg\ \varphi)) \leftrightarrow \langle+\rangle\ X\ \varphi$.
  moreover note *evil-eq-trans*
  ultimately show *?thesis* **by** *blast*
qed

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*]: $[-]'\ X\ ([-]'\ X\ \varphi) = [-]'\ X\ \varphi$ **by** *fastsimp*
lemma *pBBI-eq* [*simp*]: $[+]'\ X\ ([+]'\ X\ \varphi) = [+]'\ X\ \varphi$ **by** *fastsimp*

lemma *pBB-BB-subform-sub*: $\Downarrow ([-]'\ X\ \varphi) \subseteq \Downarrow ([-]\ X\ \varphi)$
proof *cases*
  assume $\exists\ \psi.\ ([-]\ X\ \psi) = \varphi$ **thus** *?thesis* **by** *fastsimp*
  next assume $\tilde{}\ (\exists\ \psi.\ ([-]\ X\ \psi) = \varphi)$
    thus *?thesis* **by** *fastsimp*
qed

lemma *pBBI-BBI-subform-sub*: $\Downarrow ([+]'\ X\ \varphi) \subseteq \Downarrow ([+]\ X\ \varphi)$
proof *cases*
  assume $\exists\ \psi.\ ([+]\ X\ \psi) = \varphi$ **thus** *?thesis* **by** *fastsimp*
  next assume $\tilde{}\ (\exists\ \psi.\ ([+]\ X\ \psi) = \varphi)$
    thus *?thesis* **by** *fastsimp*
qed

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 *cases*
  assume $\exists\ \psi.\ ([-]\ X\ \psi) = \varphi$
  with *this* **obtain** ψ **where** $[-]\ X\ \psi = \varphi$ **by** *auto*
  hence $[-]\ X\ \psi = [-]'\ X\ \varphi$
   and $[-]\ X\ ([-]\ X\ \psi) = [-]\ X\ \varphi$ **by** *fastsimp+*
  moreover from *evil-eq-symm evil-BB-eq* **have**
   $\vdash [-]\ X\ \psi \leftrightarrow [-]\ X\ ([-]\ X\ \psi)$ **by** *fast*
  ultimately show *?thesis* **by** *simp*
 next
  assume $\tilde{}\ (\exists\ \psi.\ ([-]\ X\ \psi) = \varphi)$
  hence $[-]'\ X\ \varphi = [-]\ X\ \varphi$ **by** *simp*
  with *evil-eq-refl* **show** *?thesis* **by** *simp*
qed

lemma *evil-BBI-pBBI-eq*: $\vdash [+]'\ X\ \varphi \leftrightarrow [+]\ X\ \varphi$
proof *cases*
  assume $\exists\ \psi.\ ([+]\ X\ \psi) = \varphi$
  with *this* **obtain** ψ **where** $[+]\ X\ \psi = \varphi$ **by** *auto*
  hence $[+]\ X\ \psi = [+]'\ X\ \varphi$

and $[+]\ X\ ([+]\ X\ \psi) = [+]\ X\ \varphi$ **by** *fastsimp+*
moreover from *evil-eq-symm evil-BBI-eq* **have**
$\vdash [+]\ X\ \psi \leftrightarrow [+]\ X\ ([+]\ X\ \psi)$ **by** *fast*
ultimately show *?thesis* **by** *simp*
next
assume $\tilde{}\ (\exists\ \psi.\ ([+]\ X\ \psi) = \varphi)$
hence $[+]'\ X\ \varphi = [+]\ X\ \varphi$ **by** *simp*
with *evil-eq-refl* **show** *?thesis* **by** *simp*
qed

lemma *evil-eq-contrapose*:
$\vdash \varphi \leftrightarrow \psi \Longrightarrow \vdash \neg\ \varphi \leftrightarrow \neg\ \psi$
using *evil-eq-weaken*
evil-eq-symm
evil-contrapose
evil-eq-intro [**where** φ=$\neg\ \varphi$ **and** ψ=$\neg\ \psi$]
by *fast*

lemma *evil-DD-pDD-eq*: $\vdash \langle - \rangle'\ X\ \varphi \leftrightarrow \langle - \rangle\ X\ \varphi$
using *evil-BB-pBB-eq* [**where** X=X **and** φ=$\neg\ \varphi$]
evil-eq-contrapose
by *fast*

lemma *evil-DDI-pDDI-eq*: $\vdash \langle + \rangle'\ X\ \varphi \leftrightarrow \langle + \rangle\ X\ \varphi$
using *evil-BBI-pBBI-eq* [**where** X=X **and** φ=$\neg\ \varphi$]
evil-eq-contrapose
by *fast*

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*: $\vdash [+]'\ X\ \varphi \rightarrow \varphi$
using *evil-eq-weaken*
evil-BBI-pBBI-eq [**where** X=X **and** φ=φ]
evil-ax4 [**where** X=X **and** φ=φ]
evil-ClassAx.hs
by *blast*

lemma *evil-pBBax4*: $\vdash [-]'\ X\ \varphi \rightarrow \varphi$
using *evil-eq-weaken*
evil-BB-pBB-eq [**where** X=X **and** φ=φ]
evil-BBax4 [**where** X=X **and** φ=φ]
evil-ClassAx.hs
by *blast*

lemma *evil-pax5*: $\vdash [+]'\ X\ \varphi \to [+]'\ X\ ([+]'\ X\ \varphi)$
proof –
  from *evil-eq-weaken*
      evil-BBI-pBBI-eq [**where** X=X **and** φ=φ]
      evil-eq-symm
      evil-BBI-map [**where** X=X
          and φ=$[+]\ X\ \varphi$
          and ψ=$[+]'\ X\ \varphi$]
  have $\vdash [+]\ X\ ([+]\ X\ \varphi) \to [+]\ X\ ([+]'\ X\ \varphi)$ **by** *blast*
  with *evil-ax5* [**where** X=X **and** φ=φ]
      evil-ClassAx.hs
  have $\vdash [+]\ X\ \varphi \to [+]\ X\ ([+]'\ X\ \varphi)$ **by** *blast*
  with *evil-eq-weaken*
      evil-BBI-pBBI-eq [**where** X=X **and** φ=φ]
      evil-ClassAx.hs
  have $\vdash [+]'\ X\ \varphi \to [+]\ X\ ([+]'\ X\ \varphi)$ **by** *blast*
  with *evil-BBI-pBBI-eq* [**where** X=X **and** φ=$[+]'\ X\ \varphi$]
      evil-eq-symm
      evil-eq-weaken
      evil-ClassAx.hs
  show *?thesis* **by** *blast*
qed

lemma *evil-pBBax5*: $\vdash [-]'\ X\ \varphi \to [-]'\ X\ ([-]'\ X\ \varphi)$
proof –
  from *evil-eq-weaken*
      evil-BB-pBB-eq [**where** X=X **and** φ=φ]
      evil-eq-symm
      evil-BB-map [**where** X=X
          and φ=$[-]\ X\ \varphi$
          and ψ=$[-]'\ X\ \varphi$]
  have $\vdash [-]\ X\ ([-]\ X\ \varphi) \to [-]\ X\ ([-]'\ X\ \varphi)$ **by** *blast*
  with *evil-BBax5* [**where** X=X **and** φ=φ]
      evil-ClassAx.hs
  have $\vdash [-]\ X\ \varphi \to [-]\ X\ ([-]'\ X\ \varphi)$ **by** *blast*
  with *evil-eq-weaken*
      evil-BB-pBB-eq [**where** X=X **and** φ=φ]
      evil-ClassAx.hs
  have $\vdash [-]'\ X\ \varphi \to [-]\ X\ ([-]'\ X\ \varphi)$ **by** *blast*
  with *evil-BB-pBB-eq* [**where** X=X **and** φ=$[-]'\ X\ \varphi$]
      evil-eq-symm
      evil-eq-weaken
      evil-ClassAx.hs
  show *?thesis* **by** *blast*

qed

lemma *evil-pax6*: $\vdash P\#\ p \to [+]'\ X\ (P\#\ p)$
using *evil-ax6* [**where** X=X **and** p=p]
by *simp*

lemma *evil-pax7*: $\vdash P\#\ p \to [-]'\ X\ (P\#\ p)$
using *evil-ax7* [**where** X=X **and** p=p]
by *simp*

lemma *evil-pax8*: $\vdash \Diamond\ X\ \varphi \to [-]'\ X\ (\Diamond\ X\ \varphi)$
using *evil-ax8* [**where** X=X **and** φ=φ]
by *simp*

lemma *evil-pBBIax8*: $\vdash \Box\ X\ \varphi \to [+]'\ X\ (\Box\ X\ \varphi)$
using *evil-BBIax8* [**where** X=X **and** φ=φ]
by *simp*

lemma *evil-pax9*: $\vdash \Box\ X\ \varphi \to \Box\ X\ ([+]'\ Y\ \varphi)$
using *evil-eq-symm*
    evil-eq-weaken
    evil-B-map [**where** X=X]
    evil-BBI-pBBI-eq [**where** X=Y **and** φ=φ]
    evil-ax9 [**where** X=X **and** Y=Y **and** φ=φ]
    evil-ClassAx.hs
by *blast*

lemma *evil-pax10*: $\vdash \Box\ X\ \varphi \to \Box\ X\ ([-]'\ Y\ \varphi)$
using *evil-eq-symm*
    evil-eq-weaken
    evil-B-map [**where** X=X]
    evil-BB-pBB-eq [**where** X=Y **and** φ=φ]
    evil-ax10 [**where** X=X **and** Y=Y **and** φ=φ]
    evil-ClassAx.hs
by *blast*

— Skipping axiom 11

lemma *evil-pax12*: $\vdash \odot\ X \to [-]'\ X\ (\odot\ X)$
using *evil-ax12* [**where** X=X]
by *simp*

lemma *evil-pax13*: $\vdash \varphi \to [+]'\ X\ (\langle - \rangle'\ X\ \varphi)$
using *evil-ax13* [**where** X=X **and** φ=φ]
by *simp*

lemma *evil-pax14*: $\vdash \varphi \rightarrow [-]'\ X\ (\langle+\rangle'\ X\ \varphi)$
using *evil-ax14* [**where** X=X **and** φ=φ]
by *simp*

— Skipping axiom 15

lemma *evil-pax16*: $\vdash [-]'\ X\ (\varphi \rightarrow \psi) \rightarrow [-]'\ X\ \varphi \rightarrow [-]'\ X\ \psi$
proof –
  let *?A* = $[-]\ X\ (\varphi \rightarrow \psi)$
  and *?A'* = $[-]'\ X\ (\varphi \rightarrow \psi)$
  and *?B* = $[-]\ X\ \varphi$
  and *?B'* = $[-]'\ X\ \varphi$
  and *?C* = $[-]\ X\ \psi$
  and *?C'* = $[-]'\ X\ \psi$
  from *evil-BB-pBB-eq* [**where** X=X]
    evil-eq-symm
    evil-eq-weaken
  have *a*: $\vdash$ *?A'* $\rightarrow$ *?A*
   and *b*: $\vdash$ *?B'* $\rightarrow$ *?B*
   and *c*: $\vdash$ *?C* $\rightarrow$ *?C'* **by** *blast+*
  moreover from *evil-ax16* **have** $\vdash$ *?A* $\rightarrow$ *?B* $\rightarrow$ *?C* **.**
  with *a evil-ClassAx.hs* **have** $\vdash$ *?A'* $\rightarrow$ *?B* $\rightarrow$ *?C* **by** *blast*
  hence [*?A'*] $:\vdash$ *?B* $\rightarrow$ *?C* **by** *simp*
  moreover from *evil-ClassAx.lift* [**where** Γ=[*?A'*]]
      b c
  have [*?A'*] $:\vdash$ *?B'* $\rightarrow$ *?B* **and** [*?A'*] $:\vdash$ *?C* $\rightarrow$ *?C'* **by** *blast+*
  moreover note *evil-ClassAx.lift-hs* [**where** Γ=[*?A'*]]
  ultimately have [*?A'*] $:\vdash$ *?B'* $\rightarrow$ *?C'* **by** *blast*
  thus *?thesis* **by** *simp*
qed

lemma *evil-pax17*: $\vdash [+]'\ X\ (\varphi \rightarrow \psi) \rightarrow [+]'\ X\ \varphi \rightarrow [+]'\ X\ \psi$
proof –
  let *?A* = $[+]\ X\ (\varphi \rightarrow \psi)$
  and *?A'* = $[+]'\ X\ (\varphi \rightarrow \psi)$
  and *?B* = $[+]\ X\ \varphi$
  and *?B'* = $[+]'\ X\ \varphi$
  and *?C* = $[+]\ X\ \psi$
  and *?C'* = $[+]'\ X\ \psi$
  from *evil-BBI-pBBI-eq* [**where** X=X]
    evil-eq-symm
    evil-eq-weaken
  have *a*: $\vdash$ *?A'* $\rightarrow$ *?A*
   and *b*: $\vdash$ *?B'* $\rightarrow$ *?B*

```
  and c: ⊢ ?C → ?C' by blast+
 moreover from evil-ax17 have ⊢ ?A → ?B → ?C .
 with a evil-ClassAx.hs have ⊢ ?A' → ?B → ?C by blast
 hence [?A'] :⊢ ?B → ?C by simp
 moreover from evil-ClassAx.lift [where Γ=[?A']]
              b c
 have [?A'] :⊢ ?B' → ?B and [?A'] :⊢ ?C → ?C' by blast+
 moreover note evil-ClassAx.lift-hs [where Γ=[?A']]
 ultimately have [?A'] :⊢ ?B' → ?C' by blast
 thus ?thesis by simp
qed

lemma evil-pBB-nec:
    assumes prv: ⊢ φ
      shows ⊢ [−]' X φ
using prv
proof −
 from prv evil-BB-nec have ⊢ [−] X φ by fast
 with evil-BB-pBB-eq [where X=X]
      evil-eq-symm evil-eq-mp
 show ?thesis by blast
qed

lemma evil-pBBI-nec:
    assumes prv: ⊢ φ
      shows ⊢ [+]' X φ
using prv
proof −
 from prv evil-BBI-nec have ⊢ [+] X φ by fast
 with evil-BBI-pBBI-eq [where X=X]
      evil-eq-symm evil-eq-mp
 show ?thesis by blast
qed

lemma evil-lift-pax16:
 assumes notnil: φs ≠ []
   shows ⊢ [−]' X (φs :→ ψ)
            → ((map ([−]' X) φs) :→ [−]' X ψ)
using notnil
proof (induct φs)
 case Nil thus ?case by fast
 next case (Cons φ φs)
  note ind-hyp = this
  show ?case
  proof cases
```

```
    assume φs = []
      with evil-pax16 [where X=X]
        show ?case by fastsimp
    next
    let ?A = [−]′ X ((φ # φs) :→ ψ)
    and ?B = [−]′ X φ
    and ?C = [−]′ X (φs :→ ψ)
    and ?D = ((map ([−]′ X) φs) :→ [−]′ X ψ)
    assume notnil: φs ≠ []
      with ind-hyp
           evil-ClassAx.lift [where Γ=[?A]]
        have map: [?A] :⊢ ?C → ?D by fast
      from evil-pax16 [where X=X
                       and φ=φ
                       and ψ=φs :→ ψ]
        have [?A] :⊢ ?B → ?C by simp
      with map
           evil-ClassAx.lift-hs [where Γ=[?A]
                                 and φ=?B
                                 and ψ=?C
                                 and χ=?D]
        show ?case by (simp del: evil-pBB-def)
  qed
qed

lemma evil-pBB-lift-map:
 assumes seq: φs :⊢ ψ
   shows (map ([−]′ X) φs) :⊢ [−]′ X ψ
using seq
proof (induct φs)
  case Nil with evil-pBB-nec [where X=X]
    show ?case by fastsimp
  next case (Cons φ φs)
    with evil-pBB-nec [where X=X and φ=(φ # φs) :→ ψ]
         evil-mp
         evil-lift-pax16 [where X=X
                          and φs=φ # φs
                          and ψ=ψ]
    show ?case by (simp del: evil-pBB-def)
qed

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

using *notnil*
proof (*induct* φs)
 case *Nil* **thus** *?case* **by** *fast*
 next case (*Cons* φ φs)
  note *ind-hyp* = *this*
  show *?case*
  proof *cases*
   assume φs = []
    with *evil-pax17* [**where** X=X]
     show *?case* **by** *fastsimp*
   next
   let *?A* = $[+]'$ X ((φ # φs) :→ ψ)
   and *?B* = $[+]'$ X φ
   and *?C* = $[+]'$ X (φs :→ ψ)
   and *?D* = ((*map* ($[+]'$ X) φs) :→ $[+]'$ X ψ)
   assume *notnil*: $\varphi s \neq$ []
    with *ind-hyp*
      evil-ClassAx.lift [**where** Γ=[*?A*]]
     have *map*: [*?A*] :⊢ *?C* → *?D* **by** *fast*
    from *evil-pax17* [**where** X=X
              and φ=φ
              and ψ=φs :→ ψ]
     have [*?A*] :⊢ *?B* → *?C* **by** *simp*
    with *map*
      evil-ClassAx.lift-hs [**where** Γ=[*?A*]
                   and φ=*?B*
                   and ψ=*?C*
                   and χ=*?D*]
     show *?case* **by** (*simp del*: *evil-pBBI-def*)
  qed
qed

lemma *evil-pBBI-lift-map*:
 assumes *seq*: φs :⊢ ψ
  shows (*map* ($[+]'$ X) φs) :⊢ $[+]'$ X ψ
using *seq*
proof (*induct* φs)
  case *Nil* **with** *evil-pBBI-nec* [**where** X=X]
   show *?case* **by** *fastsimp*
  next case (*Cons* φ φs)
   with *evil-pBBI-nec* [**where** X=X **and** φ=(φ # φs) :→ ψ]
     evil-mp
     evil-lift-pax17 [**where** X=X
              and φs=φ # φs
              and ψ=ψ]

show *?case* **by** (*simp del*: *evil-pBBI-def*)
qed

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* ($\surd$)
 where $\surd$ $(P\#\ p) = P\#\ p$
  | $\surd$ $\bot = \bot$
  | $\surd$ $(\odot\ X) = \odot\ X$
  | $\surd$ $(\varphi \to \psi) = \varphi$
  | $\surd$ $(\Box\ X\ \varphi) = \varphi$
  | $\surd$ $([-]\ X\ \varphi) = \varphi$
  | $\surd$ $([+]\ X\ \varphi) = \varphi$

abbreviation *evil-pneg* :: $('a,'b)$ *evil-form*
  ⇒ $('a,'b)$ *evil-form* (~′ - [*40*] *40*)
 where
 ~′ $\varphi \equiv$ (*if* $(\exists\ \psi.\ (\neg\ \psi) = \varphi)$
   then ($\surd$ φ)
   else $\neg\ \varphi$)

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

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

— The new pseudo-negation is constructive(?) so always simplify to it
lemma *pneg-eq* [*simp*]: (~ φ) = (~′ φ)
proof *cases*
 assume *a*: $\exists\ \psi.\ (\neg\ \psi) = \varphi$
 hence $\exists!\ \psi.\ (\neg\ \psi) = \varphi$ **by** *fastsimp*
 moreover
 then have ($\neg$ ~′ φ) = φ **by** *fastsimp*
 moreover from *a*
   pneg-def [**where** φ=φ]
 have (~ φ) = (*SOME* ψ . $(\neg\ \psi) = \varphi$) **by** *fastsimp*
 moreover note
  some1-equality [**where** P=% ψ . $(\neg\ \psi) = \varphi$
    and *a*=~′ φ]
 ultimately show *?thesis* **by** *auto*
 next

assume *b*: ~ (∃ ψ. (¬ ψ) = φ)
with *pneg-def* [**where** φ=φ]
show *?thesis* **by** *fastsimp*
qed

— These silly lemmas show how pseudo-negation plays
— with boxes and pseudo-boxes
lemma *evil-pBB-pneg-eq* [*simp*]: (∼ [−]′ X φ) = (¬ [−]′ X φ)
by *fastsimp*

lemma *evil-pBBI-pneg-eq* [*simp*]: (∼ [+]′ X φ) = (¬ [+]′ X φ)
by *fastsimp*

lemma *evil-B-pneg-eq* [*simp*]: (∼ □ X φ) = (¬ □ X φ)
by *fastsimp*

lemma *evil-pBB-pneg-eq2* [*simp*]: (∼ ¬ [−]′ X φ) = ([−]′ X φ)
by *fastsimp*

lemma *evil-pBBI-pneg-eq2* [*simp*]: (∼ ¬ [+]′ X φ) = ([+]′ X φ)
by *fastsimp*

lemma *evil-B-pneg-eq2* [*simp*]: (∼ ¬ □ X φ) = (□ X φ)
by *fastsimp*

lemma *evil-Bot-pneg-eq* [*simp*]: (∼ ⊥) = (¬ ⊥) **by** *fastsimp*

lemma *evil-Bot-pneg-eq2* [*simp*]: (∼ (¬ ⊥)) = ⊥ **by** *fastsimp*

— As we can see pseudo-negation is logically the same
— as negation
lemma *evil-pneg-eq*: ⊢ ∼ φ ↔ ¬ φ
proof *cases*
assume ∃ ψ. (¬ ψ) = φ
with *this* **obtain** ψ **where** (¬ ψ) = φ **by** *auto*
moreover hence (∼ φ) = ψ **by** *fastsimp*
moreover note *evil-dneg-eq evil-eq-symm*
ultimately show *?thesis* **by** *fastsimp*
next
assume ~ (∃ ψ. (¬ ψ) = φ)
with *evil-eq-refl* **show** *?thesis* **by** *fastsimp*
qed

lemma *evil-pdneg-eq*: ⊢ ¬ ∼ φ ↔ φ
proof *cases*

```
  assume ∃ ψ. (¬ ψ) = φ
  with someI-ex [where P=% ψ . (¬ ψ) = φ]
      evil-ClassAx.pneg-def [where φ=φ]
    have (¬ ~ φ) = φ by fastsimp
  with evil-eq-refl show ?thesis by fastsimp
 next
  assume ˜ (∃ ψ. (¬ ψ) = φ)
  with evil-eq-refl evil-dneg-eq
  show ?thesis by fastsimp
qed

lemma neg-pneg-sem-eq [simp]: (M,w |⊢ ~ φ) = (˜ (M,w |⊢ φ))
proof cases
  assume ∃ ψ . (¬ ψ) = φ
  hence (¬ ~ φ) = φ by fastsimp
  moreover
  have (˜ (M,w |⊢ ¬ ~ φ)) = (M,w |⊢ ~ φ)
    by simp
  ultimately show ?thesis by fastsimp
 next
  assume ˜ (∃ ψ. (¬ ψ) = φ)
  hence (~ φ) = (¬ φ) by fastsimp
  moreover
  have (˜ (M,w |⊢ φ)) = (M,w |⊢ ¬ φ) by simp
  ultimately show ?thesis by fastsimp
qed
```

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* $(\delta\ \varphi)$
  by $(induct\ \varphi)$ *simp-all*

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*
    $\Rightarrow$ *('a,'b) evil-form*
    $\Rightarrow$ *('a,'b) evil-form set* (Σ) **where**
  evil-FL-P:
  $\Sigma\ \Delta\ (P\#\ p) = \{P\#\ p, \neg\ (P\#\ p), \bot, \neg\ \bot\}$
  $\cup \{[-]'\ X\ (P\#\ p) \mid X.\ X \in \Delta\}$
  $\cup \{[+]'\ X\ (P\#\ p) \mid X.\ X \in \Delta\}$
  $\cup \{\neg\ [-]'\ X\ (P\#\ p) \mid X.\ X \in \Delta\}$
  $\cup \{\neg\ [+]'\ X\ (P\#\ p) \mid X.\ X \in \Delta\}$
| *evil-FL-PP*:
  $\Sigma\ \Delta\ (\odot\ X) = \{\odot\ X, \neg\ (\odot\ X), \bot, \neg\ \bot,$
  $[-]'\ X\ (\odot\ X), \neg\ [-]'\ X\ (\odot\ X)\}$
| *evil-FL-Bot*:
  $\Sigma\ \Delta\ \bot = \{\bot, \neg\ \bot\}$
| *evil-FL-Imp*:
  $\Sigma\ \Delta\ (\varphi \to \psi) = \{\ \varphi \to \psi, \neg\ (\varphi \to \psi),$
  $\varphi, \psi, \neg\ \varphi, \neg\ \psi, \bot, \neg\ \bot\}$
  $\cup\ (\Sigma\ \Delta\ \varphi) \cup (\Sigma\ \Delta\ \psi)$
| *evil-FL-B*:
  $\Sigma\ \Delta\ (\Box\ X\ \varphi) = \{\ \Box\ X\ \varphi, \neg\ \Box\ X\ \varphi,$
  $[+]'\ X\ (\Box\ X\ \varphi), \neg\ [+]'\ X\ (\Box\ X\ \varphi),$
  $\varphi, \bot, \neg\ \bot\}$
  $\cup \{\Box\ X\ ([-]'\ Y\ \varphi) \mid Y.\ Y \in \Delta\}$
  $\cup \{\neg\ \Box\ X\ ([-]'\ Y\ \varphi) \mid Y.\ Y \in \Delta\}$
  $\cup \{\Box\ X\ ([+]'\ Y\ \varphi) \mid Y.\ Y \in \Delta\}$
  $\cup \{\neg\ \Box\ X\ ([+]'\ Y\ \varphi) \mid Y.\ Y \in \Delta\}$
  $\cup \{[-]'\ Y\ \varphi \mid Y.\ Y \in \Delta\}$
  $\cup \{\neg\ [-]'\ Y\ \varphi \mid Y.\ Y \in \Delta\}$
  $\cup \{[+]'\ Y\ \varphi \mid Y.\ Y \in \Delta\}$
  $\cup \{\neg\ [+]'\ Y\ \varphi \mid Y.\ Y \in \Delta\}$
  $\cup\ (\Sigma\ \Delta\ \varphi)$
| *evil-FL-BB*:
  $\Sigma\ \Delta\ ([+]\ X\ \varphi) = \{\ [+]\ X\ \varphi, \neg\ [+]\ X\ \varphi,$

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...
using *fin-dudes*
by (*induct* φ) *simp-all*

lemma *evil-FL-refl*: φ ∈ Σ Δ φ
by (*induct* φ, *simp-all*)

lemma *pneg-evil-FL*: ∀ ψ ∈ (Σ Δ φ). (~ ψ) ∈ (Σ Δ φ)
proof (*induct* φ)
case *E-P* **thus** *?case* **by** *fastsimp*
next case *E-Bot* **thus** *?case* **by** *fastsimp*
next case *E-PP* **thus** *?case* **by** *fastsimp*
next case *E-B* **thus** *?case* **by** (*unfold evil-FL-B*,
blast intro: *evil-FL-refl*
evil-B-pneg-eq
evil-B-pneg-eq2
evil-pBB-pneg-eq
evil-pBB-pneg-eq2
evil-pBBI-pneg-eq
evil-pBBI-pneg-eq2
evil-Bot-pneg-eq
evil-Bot-pneg-eq2)
next case *E-Imp* **thus** *?case* **by** (*simp*,*auto*)
next case *E-BB* **thus** *?case* **by** *fastsimp*
next case *E-BBI* **thus** *?case* **by** *fastsimp*
qed

lemma *evil-FL-subform-refl*: ⇓ φ ⊆ Σ Δ φ
proof (*induct* φ)
case *E-P* **thus** *?case* **by** *simp*
next case *E-Bot* **thus** *?case* **by** *simp*
next case *E-PP* **thus** *?case* **by** *simp*
next case (*E-Imp* φ ψ)
note *ih* = *this*

from *ih* **have** $\Downarrow \varphi \subseteq \Sigma\ \Delta\ (\varphi \to \psi)$ **by** *fastsimp*
moreover
have $\Sigma\ \Delta\ \psi \subseteq \Sigma\ \Delta\ (\varphi \to \psi)$ **by** *fastsimp*
with *ih* **have** $\Downarrow \psi \subseteq \Sigma\ \Delta\ (\varphi \to \psi)$ **by** *fast*
ultimately show *?case* **by** *simp*
next case (*E-B X* φ)
moreover have $\Sigma\ \Delta\ \varphi \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$ **by** *fastsimp*
ultimately show *?case* **by** *simp*
next case (*E-BB X* φ)
moreover have $\Sigma\ \Delta\ \varphi \subseteq \Sigma\ \Delta\ ([-]\ X\ \varphi)$ **by** *fastsimp*
ultimately show *?case* **by** *simp*
next case (*E-BBI X* φ)
moreover have $\Sigma\ \Delta\ \varphi \subseteq \Sigma\ \Delta\ ([+]\ X\ \varphi)$ **by** *fastsimp*
ultimately show *?case* **by** *simp*
qed

lemma *evil-FL-subforms*: $\forall\ \psi \in \Sigma\ \Delta\ \varphi.\ \Downarrow \psi \subseteq \Sigma\ \Delta\ \varphi$
proof (*induct* φ)
case *E-P* **thus** *?case* **by** *fastsimp*
next case *E-Bot* **thus** *?case* **by** *fastsimp*
next case *E-PP* **thus** *?case* **by** *fastsimp*
next case (*E-Imp* $\varphi\ \psi$)
hence *ih1*: $\forall \chi{\in}\Sigma\ \Delta\ \varphi.\ \Downarrow \chi \subseteq \Sigma\ \Delta\ \varphi$
and *ih2*: $\forall \chi{\in}\Sigma\ \Delta\ \psi.\ \Downarrow \chi \subseteq \Sigma\ \Delta\ \psi$ **by** *fast+*
have $\Sigma\ \Delta\ \varphi \subseteq \Sigma\ \Delta\ (\varphi \to \psi)$ **by** *fastsimp*
with *ih1* **have**
$\forall\ \chi \in \Sigma\ \Delta\ \varphi.\ \Downarrow \chi \subseteq \Sigma\ \Delta\ (\varphi \to \psi)$ **by** *fast*
moreover
have $\Sigma\ \Delta\ \psi \subseteq \Sigma\ \Delta\ (\varphi \to \psi)$ **by** *fastsimp*
with *ih2* **have**
$\forall\ \chi \in \Sigma\ \Delta\ \psi.\ \Downarrow \chi \subseteq \Sigma\ \Delta\ (\varphi \to \psi)$ **by** *fast*
ultimately have
$\forall \chi \in (\Sigma\ \Delta\ \varphi) \cup (\Sigma\ \Delta\ \psi).\ \Downarrow\chi \subseteq \Sigma\ \Delta\ (\varphi \to \psi)$
by *fastsimp*
moreover
from *evil-FL-subform-refl* [**where** $\varphi{=}\varphi \to \psi$]
have $\{\varphi,\psi\} \cup (\Downarrow \varphi) \cup (\Downarrow \psi) \subseteq \Sigma\ \Delta\ (\varphi \to \psi)$
by *simp*
hence $\Downarrow \varphi \subseteq \Sigma\ \Delta\ (\varphi \to \psi)$
and $\Downarrow \psi \subseteq \Sigma\ \Delta\ (\varphi \to \psi)$ **by** *fast+*
ultimately show *?case* **by** *simp*
next case (*E-B X* φ)
note *ih* = *this*
— I'm really pretty sad about this,
— but I must resort to using this very stupid lemma :(

{ **fix** $A\ B\ C\ D$
  assume $A = B \vee A = C$ **and** $B \subseteq D$ **and** $C \subseteq D$
  hence $A \subseteq D$ **by** *fastsimp* }
note *sublem* = *this*
have *sub*: $\Sigma\ \Delta\ \varphi \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$ **by** *fastsimp*
with *ih* **have** $\forall\, \psi{\in}\Sigma\ \Delta\ \varphi.\ \Downarrow \psi \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$ **by** *fast*
moreover from *sub evil-FL-subform-refl*
have *reflI*: $\Downarrow \varphi \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$ **by** *fast*
moreover
let *?A* = $\{\Box\ X\ ([-]'\ Y\ \varphi)\ | Y.\ Y \in \Delta\}$
and *?B* = $\{\neg\ \Box\ X\ ([-]'\ Y\ \varphi)\ | Y.\ Y \in \Delta\}$
and *?C* = $\{\Box\ X\ ([+]'\ Y\ \varphi)\ | Y.\ Y \in \Delta\}$
and *?D* = $\{\neg\ \Box\ X\ ([+]'\ Y\ \varphi)\ | Y.\ Y \in \Delta\}$
and *?E* = $\{[-]'\ Y\ \varphi\ | Y.\ Y \in \Delta\}$
and *?F* = $\{\neg\ [-]'\ Y\ \varphi\ | Y.\ Y \in \Delta\}$
and *?G* = $\{[+]'\ Y\ \varphi\ | Y.\ Y \in \Delta\}$
and *?H* = $\{\neg\ [+]'\ Y\ \varphi\ | Y.\ Y \in \Delta\}$

from *reflI* **have** *reflII*:
   $\{\varphi\} \cup \Downarrow \varphi \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$ **by** *simp*
{ **fix** χ **assume** $\Downarrow \chi = \{\varphi\} \cup \Downarrow \varphi \vee \Downarrow \chi = \Downarrow \varphi$
 with *sublem* [**where** $A2{=}\Downarrow \chi$
        and $B2{=}\{\varphi\} \cup \Downarrow \varphi$
        and $C2{=}\Downarrow\varphi$
        and $D2{=}\Sigma\ \Delta\ (\Box\ X\ \varphi)$]
    reflI reflII
 have $\Downarrow \chi \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$ **by** *fast* }
note *EGintro* = *this*
have $\forall\, \psi{\in}?E.\ \Downarrow \psi = \{\varphi\} \cup \Downarrow \varphi \vee \Downarrow \psi = \Downarrow \varphi$
   $\forall\, \psi{\in}?G.\ \Downarrow \psi = \{\varphi\} \cup \Downarrow \varphi \vee \Downarrow \psi = \Downarrow \varphi$
    by *fastsimp+*
with *EGintro*
have $\forall\, \psi{\in}?E.\ \Downarrow \psi \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$
 and $\forall\, \psi{\in}?G.\ \Downarrow \psi \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$ **by** *blast+*
note *EGsub* = *this*

moreover
have $\forall\, \psi{\in}?A.\ \sqrt{\psi} \in ?E$
 and $\forall\, \psi{\in}?C.\ \sqrt{\psi} \in ?G$
 and $\forall\, \psi{\in}?F.\ \sqrt{\psi} \in ?E$
 and $\forall\, \psi{\in}?H.\ \sqrt{\psi} \in ?G$
   by (*fastsimp simp del*: *evil-pBB-def*
            evil-pBBI-def)+

with *EGsub* **have**

$\forall\, \psi \in ?A.\ \{\surd\psi\} \cup \Downarrow (\surd\psi) \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$
and $\forall\, \psi \in ?C.\ \{\surd\psi\} \cup \Downarrow (\surd\psi) \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$
and $\forall\, \psi \in ?F.\ \{\surd\psi,\ \bot\} \cup \Downarrow (\surd\psi) \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$
and $\forall\, \psi \in ?H.\ \{\surd\psi,\ \bot\} \cup \Downarrow (\surd\psi) \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$
by *simp+*
note *destsub* = *this*

have $\forall\, \psi \in ?A.\ \Downarrow \psi = \{\surd\psi\} \cup \Downarrow (\surd\psi)$
and $\forall\, \psi \in ?C.\ \Downarrow \psi = \{\surd\psi\} \cup \Downarrow (\surd\psi)$
and $\forall\, \psi \in ?F.\ \Downarrow \psi = \{\surd\psi,\ \bot\} \cup \Downarrow (\surd\psi)$
and $\forall\, \psi \in ?H.\ \Downarrow \psi = \{\surd\psi,\ \bot\} \cup \Downarrow (\surd\psi)$
by (*fastsimp simp del*: *evil-pBB-def*
evil-pBBI-def)+

with *destsub*
have $\forall\, \psi \in ?A.\ \Downarrow \psi \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$
and $\forall\, \psi \in ?C.\ \Downarrow \psi \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$
and $\forall\, \psi \in ?F.\ \Downarrow \psi \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$
and $\forall\, \psi \in ?H.\ \Downarrow \psi \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$
by *simp+*
note *ACFHsub* = *this*

moreover
have $\forall\, \psi \in ?B.\ \surd\psi \in ?A$
and $\forall\, \psi \in ?D.\ \surd\psi \in ?C$
by (*fastsimp simp del*: *evil-pBB-def*
evil-pBBI-def)+

with *ACFHsub* **have**
$\forall\, \psi \in ?B.\ \{\surd\psi,\ \bot\} \cup \Downarrow (\surd\psi) \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$
and $\forall\, \psi \in ?D.\ \{\surd\psi,\ \bot\} \cup \Downarrow (\surd\psi) \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$
by *simp+*
note *destsub* = *this*

have $\forall\, \psi \in ?B.\ \Downarrow \psi = \{\surd\psi,\ \bot\} \cup \Downarrow (\surd\psi)$
and $\forall\, \psi \in ?D.\ \Downarrow \psi = \{\surd\psi,\ \bot\} \cup \Downarrow (\surd\psi)$
by (*fastsimp simp del*: *evil-pBB-def*
evil-pBBI-def)+

with *destsub*
have $\forall\, \psi \in ?B.\ \Downarrow \psi \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$
and $\forall\, \psi \in ?D.\ \Downarrow \psi \subseteq \Sigma\ \Delta\ (\Box\ X\ \varphi)$
by *simp+*

moreover from *reflI* **have**

```
      ⇓ ([+]′ X (□ X φ)) ⊆ Σ Δ (□ X φ)
    and ⇓ (¬ [+]′ X (□ X φ)) ⊆ Σ Δ (□ X φ)
      by simp+

    ultimately show ?case by (simp del: evil-pBB-def
                                        evil-pBBI-def
                                    add: Ball-def)
  next case (E-BB X φ)
    with evil-FL-subform-refl
    show ?case by fastsimp
  next case (E-BBI X φ)
    with evil-FL-subform-refl
    show ?case by fastsimp
qed

lemma evil-FL-BB-to-pBB: ∀ ψ X. [−] X ψ ∈ Σ Δ φ
                          ⟶ [−]′ X ψ ∈ Σ Δ φ
proof(induct φ)
    case E-P thus ?case by simp
  next case E-Bot thus ?case by simp
  next case E-PP thus ?case by simp
  next case E-Imp thus ?case by fastsimp
  next case (E-B Y φ)
   note ih = this
   { fix ψ fix X assume mem: [−] X ψ ∈ Σ Δ (□ Y φ)
    let ?A = {□ X ([−]′ Y φ) |Y. Y ∈ Δ}
    and ?B = {¬ □ X ([−]′ Y φ) |Y. Y ∈ Δ}
    and ?C = {□ X ([+]′ Y φ) |Y. Y ∈ Δ}
    and ?D = {¬ □ X ([+]′ Y φ) |Y. Y ∈ Δ}
    and ?F = {¬ [−]′ Y φ |Y. Y ∈ Δ}
    and ?G = {[+]′ Y φ |Y. Y ∈ Δ}
    and ?H = {¬ [+]′ Y φ |Y. Y ∈ Δ}
    have [−] X ψ ∉ ?A
     and [−] X ψ ∉ ?B
     and [−] X ψ ∉ ?C
     and [−] X ψ ∉ ?D
     and [−] X ψ ∉ ?F
     and [−] X ψ ∉ ?G
     and [−] X ψ ∉ ?H
     and [−] X ψ ≠ (□ Y φ)
     and [−] X ψ ≠ (¬ □ Y φ)
     and [−] X ψ ≠ ([+]′ Y (□ Y φ))
     and [−] X ψ ≠ (¬ [+]′ Y (□ Y φ))
     and [−] X ψ ≠ ⊥
     and [−] X ψ ≠ (¬ ⊥)
```

by *auto+*
with *mem*
have *tri1*:
$[-]\ X\ \psi = \varphi \vee$
$[-]\ X\ \psi \in \{[-]'\ Z\ \varphi \mid Z.\ Z \in \Delta\} \vee$
$[-]\ X\ \psi \in \Sigma\ \Delta\ \varphi$
by (*fastsimp del*: *evil-pBB-def*
evil-pBBI-def)
have $[-]\ X\ \psi \in \{[-]'\ Z\ \varphi \mid Z.\ Z \in \Delta\}$
$\Longrightarrow [-]\ X\ \psi = \varphi \vee [-]\ X\ \psi = [-]'\ X\ \psi$
by *fastsimp*
with *tri1* **have** *tri2*:
$[-]\ X\ \psi = \varphi \vee$
$[-]\ X\ \psi = [-]'\ X\ \psi \vee$
$[-]\ X\ \psi \in \Sigma\ \Delta\ \varphi$ **by** *fastsimp*
from *evil-FL-refl* [**where** Δ=Δ **and** φ=φ]
have $[-]\ X\ \psi = \varphi \Longrightarrow [-]\ X\ \psi \in \Sigma\ \Delta\ \varphi$
by *fastsimp*
with *tri2* **have**
$[-]\ X\ \psi = [-]'\ X\ \psi \vee$
$[-]\ X\ \psi \in \Sigma\ \Delta\ \varphi$ **by** *fastsimp*
with *ih*
have *bi*: $[-]\ X\ \psi = [-]'\ X\ \psi \vee [-]'\ X\ \psi \in \Sigma\ \Delta\ (\Box\ Y\ \varphi)$
by (*fastsimp simp del*: *evil-pBB-def*
evil-pBBI-def)
from *mem*
have $[-]\ X\ \psi = [-]'\ X\ \psi \Longrightarrow [-]'\ X\ \psi \in \Sigma\ \Delta\ (\Box\ Y\ \varphi)$
by (*fastsimp simp del*: *evil-pBB-def*
evil-pBBI-def)
with *bi* **have** $[-]'\ X\ \psi \in \Sigma\ \Delta\ (\Box\ Y\ \varphi)$ **by** *fastsimp* **}**
thus *?case* **by** *fast*
next case *E-BB* **thus** *?case* **by** *fastsimp*
next case *E-BBI* **thus** *?case* **by** *fastsimp*
qed

lemma *evil-FL-BBI-to-pBBI*: $\forall\ \psi\ X.\ [+]\ X\ \psi \in \Sigma\ \Delta\ \varphi$
$\longrightarrow\ [+]'\ X\ \psi \in \Sigma\ \Delta\ \varphi$
proof(*induct* φ)
case *E-P* **thus** *?case* **by** *simp*
next case *E-Bot* **thus** *?case* **by** *simp*
next case *E-PP* **thus** *?case* **by** *simp*
next case *E-Imp* **thus** *?case* **by** *fastsimp*
next case (*E-B* $Y\ \varphi$)
note *ih* = *this*
{ fix ψ **fix** X **assume** *mem*: $[+]\ X\ \psi \in \Sigma\ \Delta\ (\Box\ Y\ \varphi)$

let *?A* $= \{\Box\ X\ ([-]'\ Y\ \varphi)\ |Y.\ Y \in \Delta\}$
and *?B* $= \{\neg\ \Box\ X\ ([-]'\ Y\ \varphi)\ |Y.\ Y \in \Delta\}$
and *?C* $= \{\Box\ X\ ([+]'\ Y\ \varphi)\ |Y.\ Y \in \Delta\}$
and *?D* $= \{\neg\ \Box\ X\ ([+]'\ Y\ \varphi)\ |Y.\ Y \in \Delta\}$
and *?E* $= \{[-]'\ Y\ \varphi\ |Y.\ Y \in \Delta\}$
and *?F* $= \{\neg\ [-]'\ Y\ \varphi\ |Y.\ Y \in \Delta\}$
and *?H* $= \{\neg\ [+]'\ Y\ \varphi\ |Y.\ Y \in \Delta\}$
have $[+]\ X\ \psi \notin$ *?A*
and $[+]\ X\ \psi \notin$ *?B*
and $[+]\ X\ \psi \notin$ *?C*
and $[+]\ X\ \psi \notin$ *?D*
and $[+]\ X\ \psi \notin$ *?E*
and $[+]\ X\ \psi \notin$ *?F*
and $[+]\ X\ \psi \notin$ *?H*
and $[+]\ X\ \psi \neq (\Box\ Y\ \varphi)$
and $[+]\ X\ \psi \neq (\neg\ \Box\ Y\ \varphi)$
and $[-]\ X\ \psi \neq (\neg\ [+]'\ Y\ (\Box\ Y\ \varphi))$
and $[-]\ X\ \psi \neq \bot$
and $[-]\ X\ \psi \neq (\neg\ \bot)$
by *auto+*
with *mem*
have *quatro1*:
$[+]\ X\ \psi = \varphi\ \vee$
$[+]\ X\ \psi = [+]'\ Y\ (\Box\ Y\ \varphi)\ \vee$
$[+]\ X\ \psi \in \{[+]'\ Z\ \varphi\ |\ Z.\ Z \in \Delta\}\ \vee$
$[+]\ X\ \psi \in \Sigma\ \Delta\ \varphi$
by (*fastsimp del*: *evil-pBB-def*
evil-pBBI-def)
have $[+]\ X\ \psi = [+]'\ Y\ (\Box\ Y\ \varphi)$
$\Longrightarrow [+]\ X\ \psi = [+]'\ X\ \psi$ **by** *fastsimp*
with *quatro1* **have** *quatro2*:
$[+]\ X\ \psi = \varphi\ \vee$
$[+]\ X\ \psi = [+]'\ X\ \psi\ \vee$
$[+]\ X\ \psi \in \{[+]'\ Z\ \varphi\ |\ Z.\ Z \in \Delta\}\ \vee$
$[+]\ X\ \psi \in \Sigma\ \Delta\ \varphi$ **by** *fastsimp*
have $[+]\ X\ \psi \in \{[+]'\ Z\ \varphi\ |\ Z.\ Z \in \Delta\}$
$\Longrightarrow [+]\ X\ \psi = \varphi \vee [+]\ X\ \psi = [+]'\ X\ \psi$ **by** *fastsimp*
with *quatro2* **have** *tri*:
$[+]\ X\ \psi = \varphi\ \vee$
$[+]\ X\ \psi = [+]'\ X\ \psi\ \vee$
$[+]\ X\ \psi \in \Sigma\ \Delta\ \varphi$ **by** *fastsimp*
from *evil-FL-refl* [**where** Δ=Δ **and** φ=φ]
have $[+]\ X\ \psi = \varphi \Longrightarrow [+]\ X\ \psi \in \Sigma\ \Delta\ \varphi$
by *fastsimp*
with *tri* **have**

```
      [+] X ψ = [+]′ X ψ ∨
        [+] X ψ ∈ Σ Δ φ by fastsimp
    with ih
    have bi: [+] X ψ = [+]′ X ψ ∨ [+]′ X ψ ∈ Σ Δ (□ Y φ)
      by (fastsimp simp del: evil-pBB-def
                             evil-pBBI-def)
    from mem
    have [+] X ψ = [+]′ X ψ ⟹ [+]′ X ψ ∈ Σ Δ (□ Y φ)
      by (fastsimp simp del: evil-pBB-def
                             evil-pBBI-def)
    with bi have [+]′ X ψ ∈ Σ Δ (□ Y φ) by fastsimp }
  thus ?case by fast
 next case E-BB thus ?case by fastsimp
 next case E-BBI thus ?case by fastsimp
qed
```

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
  ⇒ (('a,'b) evil-form set,'a,'b) evil-kripke (∠)
where
∠ φ ≡
 (| W = Atoms (Σ (δ φ) φ),
   V = (λ w p. (P# p) ∈ w),
   PP = (λ X. {w.(⊙ X) ∈ w}),
   RB = (λ X.
         {(w,v). {w,v} ⊆ Atoms (Σ (δ φ) φ) ∧
                 {ψ. (□ X ψ) ∈ w} ⊆ v}),

   RBB = (λ X.
         {(w,v). {w,v} ⊆ Atoms (Σ (δ φ) φ) ∧
                 {ψ. ([−]′ X ψ) ∈ w} ⊆ v ∧
                 {([−]′ X ψ) | ψ. ([−]′ X ψ) ∈ w} ⊆ v ∧
                 {ψ. ([+]′ X ψ) ∈ v} ⊆ w ∧
                 {([+]′ X ψ) | ψ. ([+]′ X ψ) ∈ v} ⊆ w}),
   RBBI = (λ X.
```

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

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*]:
  $(\Gamma \in Atoms\ \Phi) \equiv$
    $(\Gamma \subseteq \Phi \wedge$
    $(\forall\ \varphi{\in}\Phi.\ \varphi \in \Gamma \vee (\sim \varphi) \in \Gamma) \wedge$
    $\tilde{}\ (list\ \Gamma \mathrel{:\vdash} \bot))$
using *evil-ClassAx.Atoms-def* [**where** Γ=Γ **and** Φ=Φ]
  by(*unfold mem-def*, *auto*)

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 $\Gamma \in Atoms\ \Phi$
    and $\varphi \in \Gamma$
   shows *list* $\Gamma \mathrel{:\vdash} \varphi$
using *assms*
proof –
  from *assms finite-subset* **have**
    finite Γ **by** *fastsimp*
  with *set-list* [**where** A=Γ]
  have *set* (*list* Γ) = Γ **by** *fastsimp*
  with *assms* **have** $\varphi \in$ *set* (*list* Γ) **by** *simp*
  with *evil-ClassAx.lift-elm* **show** *?thesis* **by** *fast*
qed

lemma *evil-mem-prv2*:

assumes *finite* Φ
and $\Gamma \in$ *Atoms* Φ
and $\varphi \in \Phi$
and *list* $\Gamma :\vdash \varphi$
shows $\varphi \in \Gamma$
using *assms*
proof –
from *assms finite-subset* **have**
finite Γ **by** *fastsimp*
with *set-list* [**where** A=Γ]
have *eq1*: *set* (*list* Γ) = Γ **by** *fastsimp*
— Now proceed by reductio
{ **assume** $\varphi \notin \Gamma$
with *assms* **have** $(\sim \varphi) \in \Gamma$ **by** *fastsimp*
with *eq1* **have** $(\sim \varphi) \in$ *set* (*list* Γ) **by** *simp*
with *evil-ClassAx.lift-elm*
have *list* $\Gamma :\vdash \sim \varphi$ **by** *blast*
moreover from *evil-pneg-eq*
evil-eq-weaken
have $\vdash (\sim \varphi) \to \neg \varphi$ **by** *blast*
with *evil-ClassAx.lift* **have**
list $\Gamma :\vdash (\sim \varphi) \to \neg \varphi$ **by** *blast*
moreover note *evil-ClassAx.lift-mp* [**where** Γ=*list* Γ]
assms
ultimately have *list* $\Gamma :\vdash \bot$ **by** *blast*
with *assms* **have** *False* **by** *simp*
}
with *assms* **show** *?thesis* **by** *fast*
qed

lemma *evil-pneg-nded*:
assumes *finite* Φ
and $\Gamma \in$ *Atoms* Φ
and *list* $\Gamma :\vdash \varphi$
shows ~(*list* $\Gamma :\vdash \sim \varphi$)
using *assms*
proof –
— By reductio ad absurdem
{ **assume** *list* $\Gamma :\vdash \sim \varphi$
moreover from *evil-pneg-eq*
evil-eq-weaken
have $\vdash (\sim \varphi) \to \neg \varphi$ **by** *blast*
with *evil-ClassAx.lift* **have**
list $\Gamma :\vdash (\sim \varphi) \to \neg \varphi$ **by** *blast*
moreover note *evil-ClassAx.lift-mp*

ultimately have *list* Γ :⊢ ¬ φ **by** *fastsimp*
with *evil-ClassAx.lift-mp assms*
have *False* **by** *fastsimp* **}**
thus *?thesis* **by** *fast*
qed

lemma *evil-Atom-mem-intro*:
assumes *finite* Φ
and $\Gamma \in$ *Atoms* Φ
and $\varphi \in \Gamma$
and $\psi \in \Phi$
and *list* Γ :⊢ $\varphi \rightarrow \psi$
shows $\psi \in \Gamma$
using *assms*
proof –
from *assms evil-mem-prv*
have *list* Γ :⊢ φ **by** *blast*
with *assms evil-ClassAx.lift-mp*
have ψ: *list* Γ :⊢ ψ **by** *fast*
{ assume (~ ψ) $\in \Gamma$
with *assms evil-mem-prv* **have** *list* Γ :⊢ ~ ψ **by** *fast*
with *assms evil-pneg-nded* ψ **have** *False* **by** *blast* **}**
with *assms* **show** *?thesis* **by** *fastsimp*
qed

lemma *evil-Atom-pBB-intro*:
assumes *finite* Φ
and $\Gamma \in$ *Atoms* Φ
and $[-]\ X\ \varphi \in \Gamma$
and $[-]'\ X\ \varphi \in \Phi$
shows $[-]'\ X\ \varphi \in \Gamma$
using *assms*
proof –
from *evil-BB-pBB-eq* [**where** X=X]
evil-eq-weaken evil-eq-symm
have ⊢ $[-]\ X\ \varphi \rightarrow [-]'\ X\ \varphi$ **by** *blast*
with *evil-ClassAx.lift*
have *list* Γ :⊢ $[-]\ X\ \varphi \rightarrow [-]'\ X\ \varphi$ **by** *blast*
with *evil-Atom-mem-intro assms*
show *?thesis* **by** *blast*
qed

lemma *evil-Atom-BB-intro*:
assumes *finite* Φ
and $\Gamma \in$ *Atoms* Φ

    and $[-]'\ X\ \varphi \in \Gamma$
    and $[-]\ X\ \varphi \in \Phi$
  shows $[-]\ X\ \varphi \in \Gamma$
using *assms*
proof –
  from *evil-BB-pBB-eq* [**where** $X{=}X$]
    evil-eq-weaken
  have $\vdash [-]'\ X\ \varphi \rightarrow [-]\ X\ \varphi$ **by** *blast*
  with *evil-ClassAx.lift*
  have *list* $\Gamma :\vdash [-]'\ X\ \varphi \rightarrow [-]\ X\ \varphi$ **by** *blast*
  with *evil-Atom-mem-intro assms*
  show *?thesis* **by** *blast*
qed

lemma *evil-Atom-pBBI-intro*:
  assumes *finite* Φ
    and $\Gamma \in$ *Atoms* Φ
    and $[+]\ X\ \varphi \in \Gamma$
    and $[+]'\ X\ \varphi \in \Phi$
  shows $[+]'\ X\ \varphi \in \Gamma$
using *assms*
proof –
  from *evil-BBI-pBBI-eq* [**where** $X{=}X$]
    evil-eq-weaken evil-eq-symm
  have $\vdash [+]\ X\ \varphi \rightarrow [+]'\ X\ \varphi$ **by** *blast*
  with *evil-ClassAx.lift*
  have *list* $\Gamma :\vdash [+]\ X\ \varphi \rightarrow [+]'\ X\ \varphi$ **by** *blast*
  with *evil-Atom-mem-intro assms*
  show *?thesis* **by** *blast*
qed

lemma *evil-Atom-BBI-intro*:
  assumes *finite* Φ
    and $\Gamma \in$ *Atoms* Φ
    and $[+]'\ X\ \varphi \in \Gamma$
    and $[+]\ X\ \varphi \in \Phi$
  shows $[+]\ X\ \varphi \in \Gamma$
using *assms*
proof –
  from *evil-BBI-pBBI-eq* [**where** $X{=}X$]
    evil-eq-weaken
  have $\vdash [+]'\ X\ \varphi \rightarrow [+]\ X\ \varphi$ **by** *blast*
  with *evil-ClassAx.lift*
  have *list* $\Gamma :\vdash [+]'\ X\ \varphi \rightarrow [+]\ X\ \varphi$ **by** *blast*
  with *evil-Atom-mem-intro assms*

show *?thesis* **by** *blast*
qed

— We now relativize these lemmas to our model we are creating

lemma *evil-FL-mem-prv*:
  assumes $\Phi \in W(\angle\ \varphi)$
    and $\psi \in \Phi$
   shows *list* $\Phi :\vdash \psi$
using *assms*
proof –
  from *finite-dudes finite-evil-FL*
  have *finite* $(\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *blast*
  moreover from *assms*
  have $\Phi \in$ *Atoms* $(\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *fastsimp*
  moreover note *assms evil-mem-prv*
  ultimately show *?thesis* **by** *blast*
qed

thm *evil-mem-prv2*

lemma *evil-FL-mem-prv2*:
  assumes $\Phi \in W(\angle\ \varphi)$
    and $\psi \in \Sigma\ (\delta\ \varphi)\ \varphi$
    and *list* $\Phi :\vdash \psi$
   shows $\psi \in \Phi$
using *assms*
proof –
  from *finite-dudes finite-evil-FL*
  have *finite* $(\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *blast*
  moreover from *assms*
  have $\Phi \in$ *Atoms* $(\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *fastsimp*
  moreover note *assms evil-mem-prv2*
  ultimately show *?thesis* **by** *blast*
qed

lemma *evil-FL-pneg-nded*:
  assumes $\Phi \in W(\angle\ \varphi)$
    and *list* $\Phi :\vdash \psi$
   shows $\tilde{}$ (*list* $\Phi :\vdash \sim \psi$)
using *assms*
proof –
  from *finite-dudes finite-evil-FL*
  have *finite* $(\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *blast*
  moreover from *assms*

  have $\Phi \in$ *Atoms* $(\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *fastsimp*
  moreover note *assms evil-pneg-nded*
  ultimately show *?thesis* **by** *blast*
qed

lemma *evil-FL-mem-intro*:
  assumes $\Phi \in W(\angle\ \varphi)$
    and $\psi \in \Phi$
    and $\chi \in \Sigma\ (\delta\ \varphi)\ \varphi$
    and *list* $\Phi :\vdash \psi \to \chi$
   shows $\chi \in \Phi$
using *assms*
proof –
  from *finite-dudes finite-evil-FL*
  have *finite* $(\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *blast*
  moreover from *assms*
  have $\Phi \in$ *Atoms* $(\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *fastsimp*
  moreover note *evil-Atom-mem-intro assms*
  ultimately show *?thesis* **by** *blast*
qed

lemma *evil-FL-pBB-intro*:
   assumes $\Phi \in W(\angle\ \varphi)$
    and $[-]\ X\ \psi \in \Phi$
   shows $[-]'\ X\ \psi \in \Phi$
using *assms*
proof –
  from *finite-dudes finite-evil-FL*
  have *finite* $(\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *blast*
  moreover from *assms*
  have $\Phi \in$ *Atoms* $(\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *fastsimp*
  moreover
  from *this assms*
   have $[-]\ X\ \psi \in (\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *fastsimp*
  with *evil-FL-BB-to-pBB*
   have $[-]'\ X\ \psi \in (\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *fast*
  moreover note *evil-Atom-pBB-intro* [**where** $X{=}X$]
        assms
  ultimately show *?thesis* **by** *blast*
qed

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 $[-]\ X\ \psi \in \Phi$
using *assms*
proof –
from *finite-dudes finite-evil-FL*
have *finite* $(\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *blast*
moreover from *assms*
have $\Phi \in$ *Atoms* $(\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *fastsimp*
moreover note *evil-Atom-BB-intro* [**where** X=X]
assms
ultimately show *?thesis* **by** *blast*
qed

lemma *evil-FL-pBBI-intro*:
assumes $\Phi \in W(\angle\ \varphi)$
and $[+]\ X\ \psi \in \Phi$
shows $[+]'\ X\ \psi \in \Phi$
using *assms*
proof –
from *finite-dudes finite-evil-FL*
have *finite* $(\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *blast*
moreover from *assms*
have $\Phi \in$ *Atoms* $(\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *fastsimp*
moreover
from *this assms*
have $[+]\ X\ \psi \in (\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *fastsimp*
with *evil-FL-BBI-to-pBBI*
have $[+]'\ X\ \psi \in (\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *fast*
moreover note *evil-Atom-pBBI-intro* [**where** X=X]
assms
ultimately show *?thesis* **by** *blast*
qed

lemma *evil-push*:
assumes *finite* A
and *list* $(\{\psi\} \cup A) :\vdash \varphi$
shows *list* $A :\vdash \psi \rightarrow \varphi$
using *assms*
proof –
from *assms* **have** *finite* $(\{\psi\} \cup A)$ **by** *fastsimp*
with *set-list* [**where** A=$\{\psi\} \cup A$] **have**
eq1: *set* (*list* $(\{\psi\} \cup A)$) = $\{\psi\} \cup A$ **..**
from *assms set-list* [**where** A=A]
have *set* (*list* A) = A **by** *fast*
hence *eq2*: *set* $(\psi\ \#\ ($*list* $A))$ = $\{\psi\} \cup A$ **by** *simp*
with *eq1 eq2* **have**

```
  set (list ({ψ} ∪ A)) = set (ψ # (list A)) by fast
  with assms evil-ClassAx.lift-eq have
    ψ # (list A) :⊢ φ by blast
  with evil-ClassAx.undisch show ?thesis by blast
qed

lemma evil-push-dneg:
  assumes finite A
      and list ({∼ ψ} ∪ A) :⊢ ⊥
    shows list A :⊢ ψ
using assms
proof –
    from assms evil-push
    have list A :⊢ ¬ ∼ ψ by blast
    moreover
    from evil-pdneg-eq
         evil-eq-weaken
         evil-ClassAx.lift [where Γ=list A]
    have list A :⊢ ¬ ∼ ψ → ψ by blast
    moreover note evil-ClassAx.lift-mp
    ultimately show ?thesis by fast
qed

lemma map-to-comp:
   assumes set L = S
   shows set (map f L) = {f x | x. x ∈ S}
using assms
by (induct L,fastsimp+)

lemma image-of-comp:
  f ‘ {g χ | χ . P(χ)} = {f (g χ) | χ . P(χ)}
by fastsimp

lemma evil-unions-to-appends:
  assumes finite A
      and finite B
    shows (list (A ∪ B) @ Δ :⊢ ψ)
         = (list A @ list B @ Δ :⊢ ψ)
using assms
proof –
   let ?ASM1 = list (A ∪ B) @ Δ
   and ?ASM2 = list A @ list B @ Δ
   from assms have finite (A ∪ B) by fast
   with set-list [where A=A]
        set-list [where A=B]
```

set-list [**where** A=$A \cup B$]
have A: *set* (*list* A) = A
and B: *set* (*list* B) = B
and *AuB*: *set* (*list* ($A \cup B$)) = $A \cup B$
by *fastsimp*+
{ **fix** A B **have** *set* (A @ B) = *set* A ∪ *set* B
by (*induct* A, *fastsimp*+) }
note *union* = *this*
from *AuB* *union* **have**
eq1: *set*(*?ASM1*) = $A \cup B \cup$ *set* Δ
by *fastsimp*
from A B *union* **have**
eq2: *set*(*?ASM2*) = $A \cup B \cup$ *set* Δ
by *fastsimp*
from *eq1* *eq2* **have** *set ?ASM1* = *set ?ASM2* **by** *blast*
with *evil-ClassAx.lift-eq* **show** *?thesis* **by** *blast*
qed

— With these lemmas behind us, we may proceed forward (literally)!
— We shall prove the foward direction for each box

lemma *evil-B-forward*:
assumes *H1*: $\Box\ X\ \psi \in \Phi$
and *H2*: $(\Phi,\Psi) \in RB(\angle\ \varphi)\ X$
shows $\psi \in \Psi$
using *assms*
by *fastsimp*

lemma *evil-BB-forward*:
assumes *H1*: $[-]\ X\ \psi \in \Phi$
and *H2*: $(\Phi,\Psi) \in RBB(\angle\ \varphi)\ X$
shows $\psi \in \Psi$
using *assms*
proof –
from *H2* **have** $\Phi \in W(\angle\ \varphi)$ **by** *fastsimp*
with *H1* *evil-FL-pBB-intro*
have $[-]'\ X\ \psi \in \Phi$ **by** *fast*
with *H2* **show** *?thesis* **by** *fastsimp*
qed

lemma *evil-BBI-forward*:
assumes *H1*: $[+]\ X\ \psi \in \Phi$
and *H2*: $(\Phi,\Psi) \in RBBI(\angle\ \varphi)\ X$
shows $\psi \in \Psi$
using *assms*

proof –
  from *H2* **have** $\Phi \in W(\angle\ \varphi)$ **by** *fastsimp*
  with *H1 evil-FL-pBBI-intro*
  have $[+]'\ X\ \psi \in \Phi$ **by** *fast*
  with *H2* **show** *?thesis* **by** *fastsimp*
qed

— With the forward directions out the way, we move backward
— These are all non-trivial lemmas

lemma *evil-B-back*:
  assumes $\Phi \in W(\angle\ \varphi)$
    and $\Box\ X\ \psi \notin \Phi$
    and $\Box\ X\ \psi \in \Sigma\ (\delta\ \varphi)\ \varphi$
    shows $\exists\ \Psi.\ (\Phi,\Psi) \in RB(\angle\ \varphi)\ X \wedge (\sim\ \psi) \in \Psi$
using *assms*
proof –
  let $?s1 = \{\chi \mid \chi.\ \Box\ X\ \chi \in \Phi\}$
  let $?s2 = \{\Box\ X\ \chi \mid \chi.\ \Box\ X\ \chi \in \Phi\}$

  — We have a bunch of facts to establish
  from *finite-dudes finite-evil-FL*
  have *fin-*$\Sigma\delta\varphi\varphi$: *finite* $(\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *blast*
  moreover from *assms*
  have Φ*-atom*: $\Phi \in Atoms\ (\Sigma\ (\delta\ \varphi)\ \varphi)$ **by** *fastsimp*
  moreover note *finite-subset*
  ultimately have *fin-*Φ: *finite* Φ
    and *s2-sub*: $?s2 \subseteq \Phi$
    by *fastsimp+*
  with *finite-subset*
  have *fin-s2*: *finite ?s2*
    by *fastsimp*
  hence *finite* $(\surd\ `\ ?s2)$ **by** *simp*
  with *image-of-comp* [**where** $g = \lambda\ x.\ \Box\ X\ x$
    and $P = \lambda\ x.\ \Box\ X\ x \in \Phi$
    and $f = \surd$]
  have *fin-s1*: *finite ?s1* **by** *fastsimp*
  from *fin-s1 fin-s2 fin-*Φ
    set-list [**where** $A = ?s1$]
    set-list [**where** $A = ?s2$]
    set-list [**where** $A = \Phi$]
  have *eq1*: *set* (*list ?s1*) = *?s1*
   and *eq2*: *set* (*list ?s2*) = *?s2*
   and *eq3*: *set* (*list* Φ) = Φ **by** *blast+*
  from *eq1 eq2 map-to-comp*

```
have eq4:
  set (map (λ φ. □ X φ) (list ?s1)) = set (list ?s2)
      by fastsimp
from s2-sub eq2 eq3
have s2-sub2: set (list ?s2) ⊆ set (list Φ)
      by fastsimp

— Now reductio ad absurdem...
{ assume list ({~ ψ} ∪ ?s1) :⊢ ⊥
   with fin-s1 evil-push-dneg
   have list ?s1 :⊢ ψ by fastsimp
   with evil-B-lift-map [where X=X]
   have (map (λ φ. □ X φ) (list ?s1)) :⊢ □ X ψ by blast
   with eq4
    evil-ClassAx.lift-eq
       [where Γ=map (λ φ. □ X φ) (list ?s1)
          and Ψ=list ?s2]
   have list ?s2 :⊢ □ X ψ by fast
   with s2-sub2 evil-ClassAx.lift-mono
   have list Φ :⊢ □ X ψ by blast
   with assms evil-FL-mem-prv2 have False by fast }
hence ~(list ({~ ψ} ∪ ?s1) :⊢ ⊥) by blast
note con = this

{ fix χ assume (□ X χ) ∈ Σ (δ φ) φ
  with evil-FL-subforms
  have ⇓ (□ X χ) ⊆  Σ (δ φ) φ
     by fast
   hence χ ∈ Σ (δ φ) φ by fastsimp }
note mem = this

from assms mem have ψ ∈ Σ (δ φ) φ by blast
with pneg-evil-FL have (~ ψ) ∈ Σ (δ φ) φ by fast
moreover with s2-sub Φ-atom
  have ?s2 ⊆ Σ (δ φ) φ by fastsimp
with mem have ?s1 ⊆ Σ (δ φ) φ by fast
ultimately have {~ ψ} ∪ ?s1 ⊆ Σ (δ φ) φ by fast
with fin-Σδφφ con
     pneg-evil-FL [where φ=φ and Δ=δ φ]
     evil-ClassAx.little-lindy [where Φ=Σ (δ φ) φ
                                and Γ={~ ψ} ∪ ?s1
                                and ψ=⊥]
   obtain Ψ where A: Ψ ∈ Atoms (Σ (δ φ) φ)
             and B: {~ ψ} ∪ ?s1 ⊆ Ψ
   by (simp add: mem-def, fast+)
```

with Φ-*atom* **have** $(\Phi,\Psi) \in RB(\angle\ \varphi)\ X$ **by** *fastsimp*
with B **show** *?thesis* **by** *blast*
qed

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 $\Box$ 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 $\Diamond$ 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 $'a$ $(P\#'\ \text{-})$
| *DE-Top* $(\top)$
| *DE-Conj* $('a,'b)$ *devil-form*
  $('a,'b)$ *devil-form* (**infixr** $\wedge$ *30*)
| *DE-Neg* $('a,'b)$ *devil-form* $(\neg\ \text{-}\ [40]\ 40)$
| *DE-D* $'b$ $('a,'b)$ *devil-form* $(\Diamond)$
| *DE-PP* $'b$ $(\odot')$
| *DE-DD* $'b$ $('a,'b)$ *devil-form* $(\langle - \rangle)$
| *DE-DDI* $'b$ $('a,'b)$ *devil-form* $(\langle + \rangle)$

fun *devil-eval* :: $('a,'b)$ *evil-world set*
  $\Rightarrow ('a,'b)$ *evil-world*
  $\Rightarrow ('a,'b)$ *devil-form*
  $\Rightarrow$ *bool* $(\text{-},\text{-}\ \|\!\models\ \text{-}\ 50)$ **where**

$(\text{-},(a,\text{-}) \Vdash P\#'\ p) = (p \in a)$
$\mid\ (\text{-},\text{-} \Vdash \top) = \mathit{True}$
$\mid\ (\Omega,(a,A) \Vdash \varphi \wedge \psi) =$
$((\Omega,(a,A) \Vdash \varphi) \wedge (\Omega,(a,A) \Vdash \psi))$
$\mid\ (\Omega,(a,A) \Vdash \neg\ \varphi) = ({\sim}\ (\Omega,(a,A) \Vdash \varphi))$
$\mid\ (\Omega,(a,A) \Vdash \Diamond\ X\ \varphi) =$
$(\exists (b,B) \in \Omega.\ (\forall \chi \in A(X).\ b \models \chi)$
$\wedge\ \ \Omega,(b,B) \Vdash \varphi)$
$\mid\ (\Omega,(a,A) \Vdash \odot'\ X) = (\forall \chi \in A(X).\ a \models \chi)$
$\mid\ (\Omega,(a,A) \Vdash \langle - \rangle\ X\ \varphi) = (\exists (b,B) \in \Omega.\ a = b$
$\wedge\ B(X) \subseteq A(X)$
$\wedge\ \ \Omega,(b,B) \Vdash \varphi)$
$\mid\ (\Omega,(a,A) \Vdash \langle + \rangle\ X\ \varphi) =$
$(\exists (b,B) \in \Omega.\ a = b \wedge B(X) \supseteq A(X) \wedge\ \ \Omega,(b,B) \Vdash \varphi)$

fun *devil-modal-eval* :: $('w,'a,'b)$ *evil-kripke*
$\Rightarrow\ 'w$
$\Rightarrow ('a,'b)$ *devil-form*
$\Rightarrow$ *bool* $(\text{-},\text{-} \Vdash \text{-}\ \mathit{50})$ **where**
$(M,w \Vdash P\#'\ p) = (p \in V(M)(w))$
$\mid\ (\text{-},\text{-} \Vdash \top) = \mathit{True}$
$\mid\ (M,w \Vdash \varphi \wedge \psi) =$
$((M,w \Vdash \varphi) \wedge (M,w \Vdash \psi))$
$\mid\ (M,w \Vdash \neg\ \varphi) = ({\sim}\ (M,w \Vdash \varphi))$
$\mid\ (M,w \Vdash\ \ \Diamond\ X\ \varphi) =$
$(\exists\, v \in W(M).\ (w,v) \in RB(M)(X) \wedge M,v \Vdash \varphi)$
$\mid\ (M,w \Vdash \odot'\ X) = (w \in PP(M)(X))$
$\mid\ (M,w \Vdash \langle - \rangle\ X\ \varphi) =$
$(\exists\, v \in W(M).\ (w,v) \in RBB(M)(X) \wedge M,v \Vdash \varphi)$
$\mid\ (M,w \Vdash \langle + \rangle\ X\ \varphi) =$
$(\exists\, v \in W(M).\ (w,v) \in RBBI(M)(X) \wedge M,v \Vdash \varphi)$

primrec *devil* :: $('a,'b)$ *evil-form*
$\Rightarrow ('a,'b)$ *devil-form* **where**
devil $P\#\ p = P\#'\ p$
$\mid$ *devil* $\bot = (\neg\ \top)$
$\mid$ *devil* $(\varphi \rightarrow \psi) = (\neg\ ((\mathit{devil}\ \varphi) \wedge \neg\ (\mathit{devil}\ \psi)))$
$\mid$ *devil* $(\Box\ X\ \varphi) = (\neg\ (\Diamond\ X\ \ (\neg\ (\mathit{devil}\ \varphi))))$
$\mid$ *devil* $(\odot\ X) = \odot'\ X$
$\mid$ *devil* $([-]\ X\ \varphi) = (\neg\ (\langle - \rangle\ X\ \ (\neg\ (\mathit{devil}\ \varphi))))$
$\mid$ *devil* $([+]\ X\ \varphi) = (\neg\ (\langle + \rangle\ X\ \ (\neg\ (\mathit{devil}\ \varphi))))$

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

lemma *evil-devil1*:

$\forall M.\ \forall w.\ (M,w \models \varphi) = (M,w \Vdash\models \textit{devil}\ \varphi)$
by (*induct* φ, *fastsimp+*)

lemma *evil-devil2*:
$\forall M.\ \forall w.\ (M,w \vdash \varphi) = (M,w \Vdash\vdash \textit{devil}\ \varphi)$
by (*induct* φ, *fastsimp+*)

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
$\downarrow(P\#'\ p) = \{P\#'\ p\}$
$\mid \downarrow(\top) = \{\top\}$
$\mid \downarrow(\neg\ \varphi) = \{\neg\ \varphi\} \cup \downarrow(\varphi)$
$\mid \downarrow(\varphi \wedge \psi) = \{\varphi \wedge \psi\} \cup \downarrow(\varphi) \cup \downarrow(\psi)$
$\mid \downarrow(\Diamond\ X\ \varphi) = \{\Diamond\ X\ \varphi\} \cup \downarrow(\varphi)$
$\mid \downarrow(\odot'\ X) = \{\odot'\ X\}$
$\mid \downarrow(\langle - \rangle\ X\ \varphi) = \{\langle - \rangle\ X\ \varphi\} \cup \downarrow(\varphi)$
$\mid \downarrow(\langle + \rangle\ X\ \varphi) = \{\langle + \rangle\ X\ \varphi\} \cup \downarrow(\varphi)$

lemma *finite-devil-subforms*:
finite $(\downarrow \varphi)$
by (*induct* φ, *simp-all*)

lemma *subform-refl* [*simp*]:
$\varphi \in \downarrow\varphi$
by (*induct* φ, *simp-all*)

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*
$\wedge$ *range g* $\subseteq (L - \{p.\ (P\#'\ p) \in (\downarrow \varphi)\})$

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 –
  have *finite* $\{p.\ (P\#'\ p) \in (\downarrow \varphi)\}$
    by (*induct* φ) *simp-all*
  with *infi-L Diff-infinite-finite*
  have *infinite* $(L - \{p.\ (P\#'\ p) \in (\downarrow \varphi)\})$
    by *blast*
  with *fini-Ws* **have** $\exists\ g.$ *inj-on g Ws*
      $\wedge$ *range* $g \subseteq (L - \{p.\ (P\#'\ p) \in (\downarrow \varphi)\})$
   by (*fastsimp intro*!: *fin-inj-on-infi*)
  with *ϱ-def*
   and *someI-ex* [**where** *P*=% *g*. *inj-on g Ws*
        $\wedge$ *range* $g \subseteq (L - \{p.\ (P\#'\ p) \in (\downarrow \varphi)\})$]
  show *?thesis* **by** *fastsimp*
qed

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 $P\#'\ \varrho(w) \notin \downarrow\varphi$
using *ϱ-works rangeI*
  by *fastsimp*

lemma (**in** *EviL-ϱ*) *ϱ-eq*:
shows $\{w,v\} \subseteq Ws \Longrightarrow (w = v) = (\varrho(w) = \varrho(v))$
using *ϱ-works*
 by (*auto*, *unfold inj-on-def*, *blast*)

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 ==
    ((⋃X. RBB(M)(X) ∪ (RBB(M)(X))^-1)^*) `` {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 ((⋃i∈S. (r i) ∪ (r i)^-1)^*)
using sym-Un-converse
      sym-UNION [where r=% i. (r i) ∪ (r i)^-1]
      refl-rtrancl [where r=⋃i∈S. (r i) ∪ (r i)^-1]
      sym-rtrancl [where r=⋃i∈S. (r i) ∪ (r i)^-1]
      trans-rtrancl [where r=⋃i∈S. (r i) ∪ (r i)^-1]
  by (unfold equiv-def, blast)
```

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 ∈ col M w
using crazy-Un-equiv [where S=UNIV and r=RBB(M)]
  and equiv-class-self [where A=UNIV
                        and r=⋃X. RBB(M)(X) ∪ (RBB(M)(X))^-1]
by (unfold col-def,simp)

lemma col-mem-eq:
  (v ∈ col M w) = (col M v = col M w)
proof
  let ?R = (⋃X. RBB(M)(X) ∪ (RBB(M)(X))^-1)^*
    assume v ∈ col M w
    with crazy-Un-equiv [where S=UNIV and r=RBB(M)]
         eq-equiv-class-iff [where A=UNIV
                             and r=?R]
    show col M v = col M w by (unfold col-def, blast)
  next assume col M v = col M w
    with col-refl show v ∈ col M w by fast
qed
```

The previous lemma weakens to the following equality:

lemma *weak-col-mem-eq*:
  $(v \in col\ M\ w) = (w \in col\ M\ v)$
proof –
  from *col-mem-eq*
    have $(v \in col\ M\ w) = (col\ M\ v = col\ M\ w)$ **.**
  moreover from *col-mem-eq*
    have $(w \in col\ M\ v) = (col\ M\ w = col\ M\ v)$ **.**
  ultimately show *?thesis* **by** *auto*
qed

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 –
  from *col-refl* **have**
    $col\ M\ w \subseteq W(M) \Longrightarrow w \in W(M)$ **by** *fastsimp*
  moreover
  { assume $\heartsuit$: $w \in W(M)$
    fix p **let** $?R = \bigcup X.\ RBB(M)(X) \cup (RBB(M)(X))\hat{}{-1}$
    — The idea is to pick an arbitrary element of the column
    assume $p \in col\ M\ w$
    hence $(w,p) \in (?R)\hat{}*$ **by** (*simp add*: *col-def*)
    — And show set membership:
    hence $p \in W(M)$
    proof(*induct rule*: *rtrancl-induct*)
    — We proceed by induction...
      case *base*
        from $\heartsuit$ **show** *?case* **by** *simp*
      next case (*step p z*)
        with *prop0* **show** $z \in W(M)$ **by** *fast*
    qed }
  ultimately show *?thesis* **by** *fast*
qed

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 –
  from *col-refl* **have** $V(M)(w) \subseteq \bigcup\ V(M)\ `(col\ M\ w)$
    by *fastsimp*

```
  moreover
  { fix p assume p ∈ ⋃ V(M) `(col M w)
    hence p ∈ V(M)(w)
    proof(unfold col-def,unfold Image-def,clarify)
      — After clarification, this is what we need to prove:
      let ?R = ⋃X. RBB(M)(X) ∪ (RBB(M)(X))^-1
      fix v assume (w,v) ∈ ?R^* and p ∈ V(M)(v)
      thus p ∈ V(M)(w)
      proof (rule converse-rtrancl-induct)
        — The trick here is to use converse induction
        — converse-rtrancl-induct states:
        — ⟦(?a, ?b) ∈ ?r*; ?P ?b; ⋀y z. ⟦(y, z) ∈ ?r; (z, ?b) ∈ ?r*; ?P z⟧ ⟹ ?P
y⟧ ⟹ ?P ?a
        — we shall focus on the inductive step
        fix y z assume p ∈ V(M)(z)
                  and (y,z) ∈ ?R
        moreover from prop5 have
        ∀ X. (y,z) ∈ (RBB(M)(X))^-1
            ⟶ V(M)(z) = V(M)(y)
        and
        ∀ X. (y,z) ∈ (RBB(M)(X))
            ⟶ V(M)(z) = V(M)(y)
        by (blast)+
        ultimately show p ∈ V(M)(y) by fast
      qed
    qed
  }
  ultimately show ?thesis by fast
qed

lemma (in partly-EviL) col-V-eq:
  assumes v ∈ col M w
  shows V(M)(w) = V(M)(v)
using assms
proof -
  from assms col-mem-eq [where M=M]
    have col M v = col M w by auto
  moreover from col-V-eqp
  have V(M)(w) = ⋃ V(M) `(col M w)
   and V(M)(v) = ⋃ V(M) `(col M v) by blast+
  ultimately show ?thesis by simp
qed
```

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) ∈ RB(M)(X) = (∀ u ∈ col M v. (w,u) ∈ RB(M)(X))
proof –
  from col-refl
  have ∀ u ∈ col M v. (w,u) ∈ RB(M)(X) ⟹ (w,v) ∈ RB(M)(X)
    by fastsimp
  moreover
  { fix u let ?R = ⋃X. RBB(M)(X) ∪ (RBB(M)(X))^-1
    assume u ∈ (col M v)
    hence (v,u) ∈ ?R^* by (unfold col-def, simp)
    moreover assume (w,v) ∈ RB(M)(X)
    ultimately have (w,u) ∈ RB(M)(X)
    proof (rule rtrancl-induct)
      — This time, the proof proceeds by ordinary induction
      — As usual, we focus on the inductive step
      fix y z assume (w,y) ∈ RB(M)(X) and (y,z) ∈ ?R
      moreover with prop7 have
        ∀ Y. (y,z) ∈ (RBB(M)(Y))^-1
              ⟶ ((w,z) ∈ RB(M)(X))
        and
        ∀ Y. (y,z) ∈ (RBB(M)(Y))
              ⟶ ((w,z) ∈ RB(M)(X))
        by blast+
      ultimately show (w,z) ∈ RB(M)(X) by fast
    qed
  }
  ultimately show ?thesis by fast
qed

lemma (in partly-EviL) col-RB-eq:
assumes v ∈ col M u
  shows (w,v) ∈ RB(M)(X) = ((w,u) ∈ RB(M)(X))
using assms
proof –
  from assms col-mem-eq [where M=M]
    have col M v = col M u by auto
  moreover
  from col-RB-eqp
  have (w,v) ∈ RB(M)(X)
        = (∀ u ∈ col M v. (w,u) ∈ RB(M)(X))
   and (w,u) ∈ RB(M)(X)
        = (∀ v ∈ col M u. (w,v) ∈ RB(M)(X))
   by blast+
```

ultimately show *?thesis* **by** *simp*
qed

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 $(\exists\, u \in col\ M\ v.\ u \in W(M))$
    $= (\forall\, u \in col\ M\ v.\ u \in W(M))$
proof –
  from *col-refl*
  have $\forall\, u \in col\ M\ v.\ u \in W(M)$
   $\Longrightarrow \exists\, u \in col\ M\ v.\ u \in W(M)$
   by *fastsimp*
  moreover
  { **assume** $\exists\, u \in col\ M\ v.\ u \in W(M)$
   from *this* **obtain** u **where**
     $u \in col\ M\ v$ **and** ♡: $u \in W(M)$
      by *fastsimp*
   with *col-mem-eq* [**where** w=v] **have**
     $col\ M\ u = col\ M\ v$ **by** *auto*
   moreover from *col-mem-eq* [**where** w=u] **have**
     $\forall\, t \in col\ M\ u.\ col\ M\ t = col\ M\ u$ **by** *auto*
   ultimately have
     $\forall\, t \in col\ M\ v.\ col\ M\ t = col\ M\ u$ **by** *blast*
   moreover from ♡ *mem-col-subseteq* **have**
     $col\ M\ u \subseteq W(M)$ **by** *auto*
   moreover note *col-refl*
   ultimately have $\forall\, t \in col\ M\ v.\ t \in W(M)$ **by** *fastsimp*
  }
  ultimately show *?thesis* **by** *fast*
qed

lemma (**in** *partly-EviL*) *col-V-irr*:
  shows $(\exists\, u \in col\ M\ v.\ V(M)(u)(p))$
    $= (\forall\, u \in col\ M\ v.\ V(M)(u)(p))$
proof –
  from *col-refl*
  have $\forall\, u \in col\ M\ v.\ V(M)(u)(p)$
   $\Longrightarrow \exists\, u \in col\ M\ v.\ V(M)(u)(p)$
   by *fastsimp*
  moreover
  { **assume** $\exists\, u \in col\ M\ v.\ V(M)(u)(p)$
   from *this* **obtain** u **where**
     $u \in col\ M\ v$ **and** ♡: $V(M)(u)(p)$
      by *fastsimp*

```
    with col-mem-eq [where w=v] have
        col M u = col M v by auto
    moreover from col-mem-eq [where w=u] have
        ∀ t ∈ col M u. col M t = col M u by auto
    ultimately have
        ∀ t ∈ col M v. col M t = col M u by blast
    moreover from col-V-eqp have
        ∀ t ∈ col M v. V(M)(t) = ⋃ V(M) ` col M t
    and V(M)(u) = ⋃ V(M) ` col(M)(u)
        by blast+
    moreover note ♡
    ultimately have ∀ t ∈ col M v. V(M)(t)(p) by fastsimp
  }
  ultimately show ?thesis by fast
qed

lemma (in partly-EviL) col-RB-irr:
  shows (∃ u ∈ col M v. (w,u) ∈ RB(M)(X))
       = (∀ u ∈ col M v. (w,u) ∈ RB(M)(X))
proof -
  from col-refl
  have ∀ u ∈ col M v. (w,u) ∈ RB(M)(X)
    ⟹ ∃ u ∈ col M v. (w,u) ∈ RB(M)(X)
    by fastsimp
  moreover
  { assume ∃ u ∈ col M v. (w,u) ∈ RB(M)(X)
    from this obtain u where
      u ∈ col M v and ♡:(w,u) ∈ RB(M)(X)
        by fastsimp
    with col-mem-eq [where M=M]
      have col M v = col M u by fastsimp
    moreover
    from ♡ col-RB-eqp
    have ∀ v ∈ col M u. (w, v) ∈ RB(M)(X) by fast
    ultimately
    have ∀ u ∈ col M v. (w, u) ∈ RB(M)(X) by fast
  }
  ultimately show ?thesis by fast
qed

end
```