Topological semantics for paraconsistent and paracomplete logics

David Fuenmayor

March 17, 2025

Abstract

We investigate mathematical structures that provide natural semantics for families of (quantified) non-classical logics featuring special unary connectives, known as recovery operators, that allow us to 'recover' the properties of classical logic in a controlled manner. These structures are known as topological Boolean algebras, which are Boolean algebras extended with additional operations subject to specific conditions of a topological nature. In this study we focus on the paradigmatic case of negation. We demonstrate how these algebras are well-suited to provide a semantics for some families of paraconsistent Logics of Formal Inconsistency and paracomplete Logics of Formal Undeterminedness. These logics feature recovery operators used to earmark propositions that behave 'classically' when interacting with non-classical negations. We refer to the companion paper [1] for more information.

Contents

theory *boolean-algebra*
imports *Main*
begin

Technical configuration

declare[[*smt-timeout=30*]]
declare[[*show-types*]]

sledgehammer-params[*isar-proof=false*]
nitpick-params[*assms=true*, *user-axioms=true*, *show-all*, *expect=genuine*, *format=3*, *atoms=a b c d*]

We hide some Isabelle/HOL notation from the libraries (which we don't use) to avoid overloading

hide-const(**open**) *List.list.Nil* **unbundle** *no list-syntax*
hide-const(**open**) *Relation.converse* **unbundle** *no converse-syntax*
hide-const(**open**) *Fun.comp* **no-notation** *Fun.comp* (**infixl** ‹∘› *55*)
hide-const(**open**) *Groups.plus-class.plus* **no-notation** *Groups.plus-class.plus* (**infixl** ‹+› *65*)
hide-const(**open**) *Groups.times-class.times* **no-notation** *Groups.times-class.times* (**infixl** ‹∗› *70*)
hide-const(**open**) *Groups.minus-class.minus* **no-notation** *Groups.minus-class.minus* (**infixl** ‹−› *65*)
hide-const(**open**) *Groups.uminus-class.uminus* **unbundle** *no uminus-syntax*

1 Shallow semantical embedding of (a logic of) Boolean algebras

We encode Boolean algebras via their (Stone) representation as algebras of sets ('fields of sets'). This means that each element of (the carrier of) the algebra will be a set of 'points'. Inspired by the 'propositions as sets of worlds' paradigm from modal logic, we may think of points as being 'worlds', and thus of the elements of our Boolean algebras as 'propositions'. Of course, this is just one among many possible interpretations, and nothing stops us from thinking of points as any other kind of object (e.g., they can be sets, functions, sets of functions, etc.).

We utilize a particular naming convention: The type parameter 'w is employed for the domain/universe of 'points'. We conveniently introduce the (parametric) type-alias $('w)\sigma$ as shorthand for $'w{\Rightarrow}bool$. Hence, the elements of our algebra are objects of type $('w)\sigma$, and thus correspond to (characteristic functions of) sets of 'points'. Set-valued (resp. set-domain) functions are thus functions that have sets as their codomain (resp. domain), they are basically anything with a (parametric) type $'a{\Rightarrow}('w)\sigma$ (resp. $('w)\sigma{\Rightarrow}'a$).

Type for (characteristic functions of) sets (of 'points')

type-synonym $'w\ \sigma = \langle 'w \Rightarrow bool\rangle$

In the sequel, we will (try to) enforce the following naming convention:

(i) Upper-case latin letters (A, B, D, M, P, S, X, etc.) denote arbitrary sets (type $('w)\sigma$). We will employ lower-case letters (p, q, x, w, etc.) to denote variables playing the role of 'points'. In some contexts, the letters S and D will be employed to denote sets/domains of sets (of 'points').

(ii) Greek letters denote arbitrary set-valued functions (type $'a \Rightarrow ('w)\sigma$). We employ the letters φ, ψ *and* η to denote arbitrary unary operations (with type $('w)\sigma \Rightarrow ('w)\sigma$).

(iii) Upper-case calligraphic letters ($\mathcal{B}$, $\mathcal{I}$, $\mathcal{C}$, $\mathcal{F}$, *etc.*) are reserved for unary operations that are intended to act as 'topological operators' in the given context.

1.1 Encoding Boolean operations

Standard inclusion-based order structure on sets.

definition $subset::'w\ \sigma \Rightarrow 'w\ \sigma \Rightarrow bool$ (**infixr** $\langle \leq \rangle$ *45*)
  where $A \leq B \equiv \forall p.\ A\ p \longrightarrow B\ p$
definition $setequ::'w\ \sigma \Rightarrow 'w\ \sigma \Rightarrow bool$ (**infixr** $\langle = \rangle$ *45*)
  where $A = B \equiv \forall p.\ A\ p \longleftrightarrow B\ p$

named-theorems *order*
declare *setequ-def*[*order*] *subset-def*[*order*]

lemma *subset-char1*: $(A \leq B) = (\forall C.\ B \leq C \longrightarrow A \leq C)$ **by** (*metis subset-def*)
lemma *subset-char2*: $(A \leq B) = (\forall C.\ C \leq A \longrightarrow C \leq B)$ **by** (*metis subset-def*)

These (trivial) lemmas are intended to help automated tools.

lemma *setequ-char*: $(A = B) = (A \leq B \wedge B \leq A)$ **unfolding** *order* **by** *blast*
lemma *setequ-ext*: $(A = B) = (A = B)$ **unfolding** *order* **by** *blast*

We now encode connectives for (distributive and complemented) bounded lattices, mostly by reusing their counterpart meta-logical HOL connectives.

definition $meet::'w\ \sigma \Rightarrow 'w\ \sigma \Rightarrow 'w\ \sigma$ (**infixr** $\langle \wedge \rangle$ *54*)
  where $A \wedge B \equiv \lambda p.\ (A\ p) \wedge (B\ p)$ — intersection
definition $join::'w\ \sigma \Rightarrow 'w\ \sigma \Rightarrow 'w\ \sigma$ (**infixr** $\langle \vee \rangle$ *53*)
  where $A \vee B \equiv \lambda p.\ (A\ p) \vee (B\ p)$ — union
definition $top::'w\ \sigma$ ($\langle \top \rangle$)
  where $\top \equiv \lambda w.\ True$ — universe
definition $bottom::'w\ \sigma$ ($\langle \bot \rangle$)
  where $\bot \equiv \lambda w.\ False$ — empty-set

And introduce further operations to obtain a Boolean algebra (of sets).

definition $compl::'w\ \sigma \Rightarrow 'w\ \sigma$ ($\langle -\text{-}\rangle$ [*57*]*58*)
  where $-A \equiv \lambda p.\ \neg(A\ p)$ — (set-)complement
definition $impl::'w\ \sigma \Rightarrow 'w\ \sigma \Rightarrow 'w\ \sigma$ (**infixr** $\langle \rightarrow \rangle$ *51*)
  where $A \rightarrow B \equiv \lambda p.\ (A\ p) \longrightarrow (B\ p)$ — (set-)implication
definition $diff::'w\ \sigma \Rightarrow 'w\ \sigma \Rightarrow 'w\ \sigma$ (**infixr** $\langle \leftharpoonup \rangle$ *51*)
  where $A \leftharpoonup B \equiv \lambda p.\ (A\ p) \wedge \neg(B\ p)$ — (set-)difference
definition $dimpl::'w\ \sigma \Rightarrow 'w\ \sigma \Rightarrow 'w\ \sigma$ (**infixr** $\langle \leftrightarrow \rangle$ *51*)
  where $A \leftrightarrow B \equiv \lambda p.\ (A\ p) = (B\ p)$ — double implication
definition $sdiff::'w\ \sigma \Rightarrow 'w\ \sigma \Rightarrow 'w\ \sigma$ (**infixr** $\langle \triangle \rangle$ *51*)
  where $A \triangle B \equiv \lambda p.\ (A\ p) \neq (B\ p)$ — symmetric difference (aka. xor)

named-theorems *conn*
declare *meet-def*[*conn*] *join-def*[*conn*] *top-def*[*conn*] *bottom-def*[*conn*]
impl-def[*conn*] *dimpl-def*[*conn*] *diff-def*[*conn*] *sdiff-def*[*conn*] *compl-def*[*conn*]

Verify characterization for some connectives.

lemma *compl-char*: $-A = A \rightarrow \bot$ **unfolding** *conn* **by** *simp*
lemma *impl-char*: $A \rightarrow B = -A \vee B$ **unfolding** *conn* **by** *simp*
lemma *dimpl-char*: $A \leftrightarrow B = (A \rightarrow B) \wedge (B \rightarrow A)$ **unfolding** *conn* **by** *blast*
lemma *diff-char1*: $A \leftharpoondown B = A \wedge -B$ **unfolding** *conn* **by** *simp*
lemma *diff-char2*: $A \leftharpoondown B = -(A \rightarrow B)$ **unfolding** *conn* **by** *simp*
lemma *sdiff-char1*: $A \bigtriangleup B = (A \leftharpoondown B) \vee (B \leftharpoondown A)$ **unfolding** *conn* **by** *blast*
lemma *sdiff-char2*: $A \bigtriangleup B = -(A \leftrightarrow B)$ **unfolding** *conn* **by** *simp*

We can verify that (quite trivially) this algebra satisfies some properties of lattices.

lemma *L1*: $A = A \vee A$ **unfolding** *conn order* **by** *simp*
lemma *L2*: $A = A \wedge A$ **unfolding** *conn order* **by** *simp*
lemma *L3*: $A \leq A \vee B$ **unfolding** *conn order* **by** *simp*
lemma *L4*: $A \wedge B \leq A$ **unfolding** *conn order* **by** *simp*
lemma *L5*: $(A \wedge B) \vee B = B$ **unfolding** *setequ-char conn order* **by** *simp*
lemma *L6*: $A \wedge (A \vee B) = A$ **unfolding** *setequ-char conn order* **by** *simp*
lemma *L7*: $A \leq C \wedge B \leq C \longrightarrow A \vee B \leq C$ **unfolding** *conn order* **by** *simp*
lemma *L8*: $C \leq A \wedge C \leq B \longrightarrow C \leq A \wedge B$ **unfolding** *conn order* **by** *simp*
lemma *L9*: $A \leq B \longleftrightarrow (A \vee B) = B$ **unfolding** *setequ-char conn order* **by** *simp*
lemma *L10*: $B \leq A \longleftrightarrow (A \wedge B) = B$ **unfolding** *setequ-char conn order* **by** *simp*
lemma *L11*: $A \leq B \wedge C \leq D \longrightarrow A \vee C \leq B \vee D$ **unfolding** *conn order* **by** *simp*
lemma *L12*: $A \leq B \wedge C \leq D \longrightarrow A \wedge C \leq B \wedge D$ **unfolding** *conn order* **by** *simp*
lemma *L13*: $A \wedge \top = A$ **unfolding** *conn order* **by** *simp*
lemma *L14*: $A \vee \bot = A$ **unfolding** *conn order* **by** *simp*
lemma *L15*: $A \leq B \longleftrightarrow (\forall C.\ C \wedge A \leq C \wedge B)$ **by** (*metis L3 L4 L5 L8 setequ-char subset-char1*)
lemma *L16*: $A \leq B \longleftrightarrow (\forall C.\ C \vee A \leq C \vee B)$ **by** (*metis L11 L4 L5 setequ-char setequ-ext*)

These properties below hold in particular also for Boolean algebras.

lemma *BA-impl*: $A \leq B \longleftrightarrow A \rightarrow B = \top$ **unfolding** *conn order* **by** *simp*
lemma *BA-distr1*: $(A \wedge (B \vee C)) = ((A \wedge B) \vee (A \wedge C))$ **unfolding** *setequ-char conn order* **by** *simp*
lemma *BA-distr2*: $(A \vee (B \wedge C)) = ((A \vee B) \wedge (A \vee C))$ **unfolding** *conn order* **by** *blast*
lemma *BA-cp*: $A \leq B \longleftrightarrow -B \leq -A$ **unfolding** *conn order* **by** *blast*
lemma *BA-deMorgan1*: $-(A \vee B) = (-A \wedge -B)$ **unfolding** *conn order* **by** *simp*
lemma *BA-deMorgan2*: $-(A \wedge B) = (-A \vee -B)$ **unfolding** *conn order* **by** *simp*
lemma *BA-dn*: $--A = A$ **unfolding** *conn order* **by** *simp*
lemma *BA-cmpl-equ*: $(-A = B) = (A = -B)$ **unfolding** *conn order* **by** *blast*

We conveniently introduce these properties of sets of sets (of points).

definition *meet-closed*::$('w\ \sigma)\sigma \Rightarrow$ *bool*
where *meet-closed* $S \equiv\ \forall X\ Y.\ (S\ X \wedge S\ Y) \longrightarrow S(X \wedge Y)$
definition *join-closed*::$('w\ \sigma)\sigma \Rightarrow$ *bool*
where *join-closed* $S \equiv\ \forall X\ Y.\ (S\ X \wedge S\ Y) \longrightarrow S(X \vee Y)$

definition *upwards-closed*::$('w\ \sigma)\sigma \Rightarrow$ *bool*
where *upwards-closed* $S \equiv \forall X\ Y.\ S\ X \wedge X \leq Y \longrightarrow S\ Y$
definition *downwards-closed*::$('w\ \sigma)\sigma \Rightarrow$ *bool*
where *downwards-closed* $S \equiv \forall X\ Y.\ S\ X \wedge Y \leq X \longrightarrow S\ Y$

1.2 Atomicity and primitive equality

We can verify indeed that the algebra is atomic (in three different ways) by relying on the presence of primitive equality in HOL.

lemma *atomic1*: $\forall w.\ \exists Q.\ Q\ w \wedge (\forall P.\ P\ w \longrightarrow Q \leq P)$ **unfolding** *order* **using** *the-sym-eq-trivial* **by** *metis*

definition *atom* $A \equiv \neg(A = \bot) \wedge (\forall P.\ A \leq P \vee A \leq -P)$

lemma *atomic2*: $\forall w.\ \exists Q.\ Q\ w \wedge \mathit{atom}\ Q$ **unfolding** *atom-def order conn* **using** *the-sym-eq-trivial* **by** *metis*
lemma *atomic3*: $\forall P.\ \neg(P = \bot) \longrightarrow (\exists Q.\ \mathit{atom}\ Q \wedge Q \leq P)$ **unfolding** *atom-def order conn* **by** *fastforce*

Now with interactive proof:

lemma $\forall P.\ \neg(P = \bot) \longrightarrow (\exists Q.\ \mathit{atom}\ Q \wedge Q \leq P)$
proof –
  { fix $P{::}'w\ \sigma$
    { assume $\neg(P = \bot)$
      hence $\exists v.\ P\ v$ **unfolding** *conn order* **by** *simp*
      then obtain w **where** $1{:}P\ w$ **by** (*rule exE*)
      let $?Q{=}\lambda v.\ v = w$ — using HOL primitive equality
      have 2: *atom* $?Q$ **unfolding** *atom-def* **unfolding** *conn order* **by** *simp*
      have $\forall v.\ ?Q\ v \longrightarrow P\ v$ **using** 1 **by** *simp*
      hence 3: $?Q \leq P$ **unfolding** *order* **by** *simp*
      from $2\ 3$ **have** $\exists Q.\ \mathit{atom}\ Q \wedge Q \leq P$ **by** *blast*
    } **hence** $\neg(P = \bot) \longrightarrow (\exists Q.\ \mathit{atom}\ Q \wedge Q \leq P)$ **by** (*rule impI*)
  } **thus** *?thesis* **by** (*rule allI*)
qed

1.3 Miscellaneous notions

We add some miscellaneous notions that will be useful later.

abbreviation *isEmpty* $S \equiv \forall x.\ \neg S\ x$
abbreviation *nonEmpty* $S \equiv \exists x.\ S\ x$

Function composition.

definition *fun-comp* :: $('b \Rightarrow 'c) \Rightarrow (\ 'a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'c$ (**infixl** ‹∘› *75*)
  where $\varphi \circ \psi \equiv \lambda x.\ \varphi\ (\psi\ x)$

Inverse projection maps a unary function to a 'projected' binary function wrt. its 1st argument.

abbreviation *inv-proj*::‹$('a \Rightarrow 'c) \Rightarrow ('a \Rightarrow 'b \Rightarrow 'c)$› (‹(-)↿›)
  where $D{\upharpoonleft} \equiv \lambda A\ B.\ D\ A$

Image of a mapping φ, with range as an special case.

definition *image*::$('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow bool) \Rightarrow ('b \Rightarrow bool)$ (‹⟦- -⟧›)
  where $⟦\varphi\ S⟧ \equiv \lambda y.\ \exists x.\ (S\ x) \wedge (\varphi\ x) = y$
definition *range*::$('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow bool)$ (‹⟦-'-⟧›)
  where $⟦\varphi\ \text{-}⟧ \equiv \lambda Y.\ \exists x.\ (\varphi\ x) = Y$
lemma *range-char1*: $⟦\varphi\ \text{-}⟧ = ⟦\varphi\ (\lambda x.\ \mathit{True})⟧$ **by** (*simp add*: *image-def range-def*)
lemma *range-char2*: $⟦\varphi\ \text{-}⟧ = (\lambda X.\ \exists S.\ ⟦\varphi\ S⟧\ X)$ **unfolding** *range-def image-def* **by** *blast*

lemma *image-comp*: $⟦(f \circ g)\ S⟧ = ⟦f\ ⟦g\ S⟧⟧$ **unfolding** *fun-comp-def image-def* **by** *metis*

end
theory *boolean-algebra-operators*
imports *boolean-algebra*
begin

1.4 Operations on set-valued functions

Functions with sets in their codomain will be called here 'set-valued functions'. We conveniently define some (2nd-order) Boolean operations on them.

The 'meet' and 'join' of two set-valued functions.

definition *svfun-meet*::$('a \Rightarrow 'w\ \sigma) \Rightarrow ('a \Rightarrow 'w\ \sigma) \Rightarrow ('a \Rightarrow 'w\ \sigma)$ (**infixr** ‹$\wedge^:$› *62*)
where $\varphi \wedge^: \psi \equiv \lambda x.\ (\varphi\ x) \wedge (\psi\ x)$
definition *svfun-join*::$('a \Rightarrow 'w\ \sigma) \Rightarrow ('a \Rightarrow 'w\ \sigma) \Rightarrow ('a \Rightarrow 'w\ \sigma)$ (**infixr** ‹$\vee^:$› *61*)
where $\varphi \vee^: \psi \equiv \lambda x.\ (\varphi\ x) \vee (\psi\ x)$

Analogously, we can define an 'implication' and a 'complement'.

definition *svfun-impl*::$('a \Rightarrow 'w\ \sigma) \Rightarrow ('a \Rightarrow 'w\ \sigma) \Rightarrow ('a \Rightarrow 'w\ \sigma)$ (**infixr** ‹$\rightarrow^:$› *61*)
where $\psi \rightarrow^: \varphi \equiv \lambda x.\ (\psi\ x) \rightarrow (\varphi\ x)$
definition *svfun-compl*::$('a \Rightarrow 'w\ \sigma) \Rightarrow ('a \Rightarrow 'w\ \sigma)$ (‹(-^-)›)
where $\varphi^- \equiv \lambda x.\ -(\varphi\ x)$

There are two natural 0-ary connectives (aka. constants).

definition *svfun-top*::$'a \Rightarrow 'w\ \sigma$ (‹$\top^:$›)
where $\top^: \equiv \lambda x.\ \top$
definition *svfun-bot*::$'a \Rightarrow 'w\ \sigma$ (‹$\bot^:$›)
where $\bot^: \equiv \lambda x.\ \bot$

named-theorems *conn2*
declare *svfun-meet-def*[*conn2*] *svfun-join-def*[*conn2*] *svfun-impl-def*[*conn2*]
svfun-compl-def[*conn2*] *svfun-top-def*[*conn2*] *svfun-bot-def*[*conn2*]

And, of course, set-valued functions are naturally ordered in the expected way:

definition *svfun-sub*::$('a \Rightarrow 'w\ \sigma) \Rightarrow ('a \Rightarrow 'w\ \sigma) \Rightarrow$ *bool* (**infixr** ‹$\leq^:$› *55*)
where $\psi \leq^: \varphi \equiv \forall x.\ (\psi\ x) \leq (\varphi\ x)$
definition *svfun-equ*::$('a \Rightarrow 'w\ \sigma) \Rightarrow ('a \Rightarrow 'w\ \sigma) \Rightarrow$ *bool* (**infixr** ‹$=^:$› *55*)
where $\psi =^: \varphi \equiv \forall x.\ (\psi\ x) = (\varphi\ x)$

named-theorems *order2*
declare *svfun-sub-def*[*order2*] *svfun-equ-def*[*order2*]

These (trivial) lemmas are intended to help automated tools.

lemma *svfun-sub-char*: $(\psi \leq^: \varphi) = (\psi \rightarrow^: \varphi =^: \top^:)$ **by** (*simp add*: *BA-impl svfun-equ-def svfun-impl-def svfun-sub-def svfun-top-def*)
lemma *svfun-equ-char*: $(\psi =^: \varphi) = (\psi \leq^: \varphi \wedge \varphi \leq^: \psi)$ **unfolding** *order2 setequ-char* **by** *blast*
lemma *svfun-equ-ext*: $(\psi =^: \varphi) = (\psi = \varphi)$ **by** (*meson ext setequ-ext svfun-equ-def*)

Clearly, set-valued functions form a Boolean algebra. We can prove some interesting relationships:

lemma *svfun-compl-char*: $\varphi^- = (\varphi \rightarrow^: \bot^:)$ **unfolding** *conn conn2* **by** *simp*
lemma *svfun-impl-char1*: $(\psi \rightarrow^: \varphi) = (\psi^- \vee^: \varphi)$ **unfolding** *conn conn2* **by** *simp*
lemma *svfun-impl-char2*: $(\psi \rightarrow^: \varphi) = (\psi \wedge^: (\varphi^-))^-$ **unfolding** *conn conn2* **by** *simp*
lemma *svfun-deMorgan1*: $(\psi \wedge^: \varphi)^- = (\psi^-) \vee^: (\varphi^-)$ **unfolding** *conn conn2* **by** *simp*
lemma *svfun-deMorgan2*: $(\psi \vee^: \varphi)^- = (\psi^-) \wedge^: (\varphi^-)$ **unfolding** *conn conn2* **by** *simp*

1.5 Operators and their transformations

Dual to set-valued functions we can have set-domain functions. For them we can define the 'dual-complement'.

definition *sdfun-dcompl*::$('w\ \sigma \Rightarrow 'a) \Rightarrow ('w\ \sigma \Rightarrow 'a)$ (‹(-$^{d-}$)›)
where $\varphi^{d-} \equiv \lambda X.\ \varphi(-X)$
lemma *sdfun-dcompl-char*: $\varphi^{d-} = (\lambda X.\ \exists Y.\ (\varphi\ Y) \wedge (X = -Y))$ **by** (*metis BA-dn setequ-ext sdfun-dcompl-def*)

Operators are a particularly important kind of functions. They are both set-valued and set-domain. Thus our algebra of operators inherits the connectives defined above plus the ones below.

We conveniently define the 'dual' of an operator.

definition *op-dual*::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow ('w\ \sigma \Rightarrow 'w\ \sigma)$ (‹(-^{d})›)
where $\varphi^{d} \equiv \lambda X.\ -(\varphi(-X))$

The following two 0-ary connectives (i.e. operator 'constants') exist already (but somehow implicitly). We just make them explicit by introducing some convenient notation.

definition *id-op*::$'w\ \sigma \Rightarrow 'w\ \sigma$ (‹**e**›)
where $\mathbf{e} \equiv \lambda X.\ X$
definition *compl-op*::$'w\ \sigma \Rightarrow 'w\ \sigma$ (‹**n**›)
where $\mathbf{n} \equiv \lambda X.\ -X$

declare *sdfun-dcompl-def*[*conn2*] *op-dual-def*[*conn2*] *id-op-def*[*conn2*] *compl-op-def*[*conn2*]

We now prove some lemmas (some of them might help provers in their hard work).

lemma *dual-compl-char1*: $\varphi^{d-} = (\varphi^{d})^{-}$ **unfolding** *conn2 conn order* **by** *simp*
lemma *dual-compl-char2*: $\varphi^{d-} = (\varphi^{-})^{d}$ **unfolding** *conn2 conn order* **by** *simp*
lemma *sfun-compl-invol*: $\varphi^{--} = \varphi$ **unfolding** *conn2 conn order* **by** *simp*
lemma *dual-invol*: $\varphi^{dd} = \varphi$ **unfolding** *conn2 conn order* **by** *simp*
lemma *dualcompl-invol*: $(\varphi^{d-})^{d-} = \varphi$ **unfolding** *conn2 conn order* **by** *simp*

lemma *op-prop1*: $\mathbf{e}^{d} = \mathbf{e}$ **unfolding** *conn2 conn* **by** *simp*
lemma *op-prop2*: $\mathbf{n}^{d} = \mathbf{n}$ **unfolding** *conn2 conn* **by** *simp*
lemma *op-prop3*: $\mathbf{e}^{-} = \mathbf{n}$ **unfolding** *conn2 conn* **by** *simp*
lemma *op-prop4*: $(\varphi \vee^{:} \psi)^{d} = (\varphi^{d}) \wedge^{:} (\psi^{d})$ **unfolding** *conn2 conn* **by** *simp*
lemma *op-prop5*: $(\varphi \vee^{:} \psi)^{-} = (\varphi^{-}) \wedge^{:} (\psi^{-})$ **unfolding** *conn2 conn* **by** *simp*
lemma *op-prop6*: $(\varphi \wedge^{:} \psi)^{d} = (\varphi^{d}) \vee^{:} (\psi^{d})$ **unfolding** *conn2 conn* **by** *simp*
lemma *op-prop7*: $(\varphi \wedge^{:} \psi)^{-} = (\varphi^{-}) \vee^{:} (\psi^{-})$ **unfolding** *conn2 conn* **by** *simp*
lemma *op-prop8*: $\top^{:} = \mathbf{n} \vee^{:} \mathbf{e}$ **unfolding** *conn2 conn* **by** *simp*
lemma *op-prop9*: $\bot^{:} = \mathbf{n} \wedge^{:} \mathbf{e}$ **unfolding** *conn2 conn* **by** *simp*

The notion of a fixed-point is fundamental. We speak of sets being fixed-points of operators. We define a function that given an operator returns the set of all its fixed-points.

definition *fixpoints*::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow ('w\ \sigma)\sigma$ (‹*fp*›)
where $fp\ \varphi \equiv \lambda X.\ (\varphi\ X) = X$

We can in fact 'operationalize' the function above, thus obtaining a 'fixed-point operation'.

definition *op-fixpoint*::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow ('w\ \sigma \Rightarrow 'w\ \sigma)$ (‹(-^{fp})›)
where $\varphi^{fp} \equiv \lambda X.\ (\varphi\ X) \leftrightarrow X$

declare *fixpoints-def*[*conn2*] *op-fixpoint-def*[*conn2*]

Interestingly, the fixed-point operation (or transformation) is definable in terms of the others.

lemma *op-fixpoint-char*: $\varphi^{fp} = (\varphi \wedge^{:} \mathbf{e}) \vee^{:} (\varphi^{-} \wedge^{:} \mathbf{n})$ **unfolding** *conn2 order conn* **by** *blast*

Given an operator φ the fixed-points of it's dual is the set of complements of $\varphi's$ fixed-points.

lemma *fp-dual*: *fp* $\varphi^{d} = (\mathit{fp}\ \varphi)^{d-}$ **unfolding** *order conn conn2* **by** *blast*

The fixed-points of $\varphi's$ complement is the set of complements of the fixed-points of $\varphi's$ dual-complement.

lemma *fp-compl*: *fp* $\varphi^{-} = (\mathit{fp}\ (\varphi^{d-}))^{d-}$ **by** (*simp add*: *dual-compl-char2 dualcompl-invol fp-dual*)

The fixed-points of $\varphi's$ dual-complement is the set of complements of the fixed-points of $\varphi's$ complement.

lemma *fp-dcompl*: *fp* $(\varphi^{d-}) = (\mathit{fp}\ \varphi^{-})^{d-}$ **by** (*simp add*: *dualcompl-invol fp-compl*)

The fixed-points function and the fixed-point operation are essentially related.

lemma *fp-rel*: *fp* $\varphi\ A \longleftrightarrow (\varphi^{fp}\ A) = \top$ **unfolding** *conn2 order conn* **by** *simp*
lemma *fp-d-rel*: *fp* $\varphi^{d}\ A \longleftrightarrow \varphi^{fp}(-A) = \top$ **unfolding** *conn2 order conn* **by** *blast*
lemma *fp-c-rel*: *fp* $\varphi^{-}\ A \longleftrightarrow \varphi^{fp}\ A = \bot$ **unfolding** *conn2 order conn* **by** *blast*
lemma *fp-dc-rel*: *fp* $(\varphi^{d-})\ A \longleftrightarrow \varphi^{fp}(-A) = \bot$ **unfolding** *conn2 order conn* **by** *simp*

The fixed-point operation is involutive.

lemma *ofp-invol*: $(\varphi^{fp})^{fp} = \varphi$ **unfolding** *conn2 order conn* **by** *blast*

And commutes the dual with the dual-complement operations.

lemma *ofp-comm-dc1*: $(\varphi^{d})^{fp} = (\varphi^{fp})^{d-}$ **unfolding** *conn2 order conn* **by** *blast*
lemma *ofp-comm-dc2*:$(\varphi^{d-})^{fp} = (\varphi^{fp})^{d}$ **unfolding** *conn2 order conn* **by** *simp*

The fixed-point operation commutes with the complement.

lemma *ofp-comm-compl*: $(\varphi^{-})^{fp} = (\varphi^{fp})^{-}$ **unfolding** *conn2 order conn* **by** *blast*

The above motivates the following alternative definition for a 'complemented-fixed-point' operation.

lemma *ofp-fixpoint-compl-def*: $\varphi^{fp-} = (\lambda X.\ (\varphi\ X) \triangle X)$ **unfolding** *conn2 conn* **by** *simp*

Analogously, the 'complemented fixed-point' operation is also definable in terms of the others.

lemma *op-fixpoint-compl-char*: $\varphi^{fp-} = (\varphi \vee^{:} \mathbf{e}) \wedge^{:} (\varphi^{-} \vee^{:} \mathbf{n})$ **unfolding** *conn2 conn* **by** *blast*

In fact, function composition can be seen as an additional binary connective for operators. We show below some interesting relationships that hold.

lemma *op-prop10*: $\varphi = (\mathbf{e} \circ \varphi)$ **unfolding** *conn2 fun-comp-def* **by** *simp*
lemma *op-prop11*: $\varphi = (\varphi \circ \mathbf{e})$ **unfolding** *conn2 fun-comp-def* **by** *simp*
lemma *op-prop12*: $\mathbf{e} = (\mathbf{n} \circ \mathbf{n})$ **unfolding** *conn2 conn fun-comp-def* **by** *simp*
lemma *op-prop13*: $\varphi^{-} = (\mathbf{n} \circ \varphi)$ **unfolding** *conn2 fun-comp-def* **by** *simp*
lemma *op-prop14*: $\varphi^{d-} = (\varphi \circ \mathbf{n})$ **unfolding** *conn2 fun-comp-def* **by** *simp*
lemma *op-prop15*: $\varphi^{d} = (\mathbf{n} \circ \varphi \circ \mathbf{n})$ **unfolding** *conn2 fun-comp-def* **by** *simp*

There are also some useful properties regarding the images of operators.

lemma *im-prop1*: $[\![\varphi\ D]\!]^{d-} = [\![\varphi^{d}\ D^{d-}]\!]$ **unfolding** *image-def op-dual-def sdfun-dcompl-def* **by** (*metis BA-dn setequ-ext*)
lemma *im-prop2*: $[\![\varphi^{-}\ D]\!]^{d-} = [\![\varphi\ D]\!]$ **unfolding** *image-def svfun-compl-def sdfun-dcompl-def* **by** (*metis BA-dn setequ-ext*)

lemma *im-prop3*: $[\![\varphi^d\ D]\!]^{d-} = [\![\varphi\ D^{d-}]\!]$ **unfolding** *image-def op-dual-def sdfun-dcompl-def* **by** (*metis BA-dn setequ-ext*)
lemma *im-prop4*: $[\![\varphi^{d-}\ D]\!]^{d-} = [\![\varphi^d\ D]\!]$ **unfolding** *image-def op-dual-def sdfun-dcompl-def* **by** (*metis BA-dn setequ-ext*)
lemma *im-prop5*: $[\![\varphi^-\ D^{d-}]\!] = [\![\varphi^{d-}\ D]\!]^{d-}$ **unfolding** *image-def svfun-compl-def sdfun-dcompl-def* **by** (*metis* (*no-types, opaque-lifting*) *BA-dn setequ-ext*)
lemma *im-prop6*: $[\![\varphi^{d-}\ D^{d-}]\!] = [\![\varphi\ D]\!]$ **unfolding** *image-def sdfun-dcompl-def* **by** (*metis BA-dn setequ-ext*)

Observe that all results obtained by assuming fixed-point predicates extend to their associated operators.

lemma $\varphi^{fp}(A) \wedge \Gamma(A) \leq \Delta(A) \longrightarrow (fp\ \varphi)(A) \longrightarrow \Gamma(A) \leq \Delta(A)$
by (*simp add*: *fp-rel meet-def setequ-ext subset-def top-def*)
lemma $\varphi^{fp}(A) \wedge \varphi^{fp}(B) \wedge (\Gamma\ A\ B) \leq (\Delta\ A\ B) \longrightarrow (fp\ \varphi)(A) \wedge (fp\ \varphi)(B) \longrightarrow (\Gamma\ A\ B) \leq (\Delta\ A\ B)$
by (*simp add*: *fp-rel meet-def setequ-ext subset-def top-def*)

end
theory *boolean-algebra-infinitary*
imports *boolean-algebra-operators*
begin

1.6 Encoding infinitary Boolean operations

Our aim is to encode complete Boolean algebras (of sets) which we can later be used to interpret quantified formulas (somewhat in the spirit of Boolean-valued models for set theory).

We start by defining infinite meet (infimum) and infinite join (supremum) operations.

definition *infimum*:: $('w\ \sigma)\sigma \Rightarrow 'w\ \sigma$ (‹$\bigwedge$-›)
where $\bigwedge S \equiv \lambda w.\ \forall X.\ S\ X \longrightarrow X\ w$
definition *supremum*::$('w\ \sigma)\sigma \Rightarrow 'w\ \sigma$ (‹$\bigvee$-›)
where $\bigvee S \equiv \lambda w.\ \exists X.\ S\ X\ \wedge\ X\ w$

declare *infimum-def*[*conn*] *supremum-def*[*conn*]

Infimum and supremum satisfy an infinite variant of the De Morgan laws.

lemma *iDM-a*: $-(\bigwedge S) = \bigvee(S^{d-})$ **unfolding** *order conn conn2* **by** *force*
lemma *iDM-b*: $-(\bigvee S) = \bigwedge(S^{d-})$ **unfolding** *order conn conn2* **by** *force*

We show that our encoded Boolean algebras are lattice-complete. The functions below return the set of upper-/lower-bounds of a set of sets S (wrt. domain D).

definition *upper-bounds*::$('w\ \sigma)\sigma \Rightarrow ('w\ \sigma)\sigma$ (‹*ub*›)
where $ub\ S \equiv \lambda U.\ \forall X.\ S\ X \longrightarrow X \leq U$
definition *upper-bounds-restr*::$('w\ \sigma)\sigma \Rightarrow ('w\ \sigma)\sigma \Rightarrow ('w\ \sigma)\sigma$ (‹ub^-›)
where $ub^D\ S \equiv \lambda U.\ D\ U \wedge (\forall X.\ S\ X \longrightarrow X \leq U)$
definition *lower-bounds*::$('w\ \sigma)\sigma \Rightarrow ('w\ \sigma)\sigma$ (‹*lb*›)
where $lb\ S \equiv \lambda L.\ \forall X.\ S\ X \longrightarrow L \leq X$
definition *lower-bounds-restr*::$('w\ \sigma)\sigma \Rightarrow ('w\ \sigma)\sigma \Rightarrow ('w\ \sigma)\sigma$ (‹lb^-›)
where $lb^D\ S \equiv \lambda L.\ D\ L \wedge (\forall X.\ S\ X \longrightarrow L \leq X)$

lemma *ub-char*: $ub\ S = (let\ D{=}\top\ in\ ub^D\ S)$ **by** (*simp add*: *top-def upper-bounds-def upper-bounds-restr-def*)
lemma *lb-char*: $lb\ S = (let\ D{=}\top\ in\ lb^D\ S)$ **by** (*simp add*: *top-def lower-bounds-def lower-bounds-restr-def*)

Similarly, the functions below return the set of least/greatest upper-/lower-bounds for S (wrt. D).

definition *lub*::$('w\ \sigma)\sigma \Rightarrow ('w\ \sigma)\sigma$ (‹*lub*›)
  where $lub\ S \equiv \lambda U.\ ub\ S\ U \wedge (\forall X.\ ub\ S\ X \longrightarrow U \leq X)$
definition *lub-restr*::$('w\ \sigma)\sigma \Rightarrow ('w\ \sigma)\sigma \Rightarrow ('w\ \sigma)\sigma$ (‹*lub*‾›)
  where $lub^D\ S \equiv \lambda U.\ ub^D\ S\ U \wedge (\forall X.\ ub^D\ S\ X \longrightarrow U \leq X)$
definition *glb*::$('w\ \sigma)\sigma \Rightarrow ('w\ \sigma)\sigma$ (‹*glb*›)
  where $glb\ S \equiv \lambda L.\ lb\ S\ L \wedge (\forall X.\ lb\ S\ X \longrightarrow X \leq L)$
definition *glb-restr*::$('w\ \sigma)\sigma \Rightarrow ('w\ \sigma)\sigma \Rightarrow ('w\ \sigma)\sigma$ (‹*glb*‾›)
  where $glb^D\ S \equiv \lambda L.\ lb^D\ S\ L \wedge (\forall X.\ lb^D\ S\ X \longrightarrow X \leq L)$

Both pairs of definitions above are suitably related. (Note that the term $\top$ below denotes the top element in the algebra of sets of sets (i.e. the powerset).)

lemma *lub-char*: $lub\ S = (let\ D{=}\top\ in\ lub^D\ S)$ **by** (*simp add*: *lub-def lub-restr-def ub-char*)
lemma *glb-char*: $glb\ S = (let\ D{=}\top\ in\ glb^D\ S)$ **by** (*simp add*: *glb-def glb-restr-def lb-char*)

Clearly, the notions of infimum/supremum correspond to least/greatest upper-/lower-bound.

lemma *sup-lub*: $lub\ S\ \bigvee S$ **unfolding** *lub-def upper-bounds-def supremum-def subset-def* **by** *blast*
lemma *sup-exist-unique*: $\forall S.\ \exists! X.\ lub\ S\ X$ **by** (*meson lub-def setequ-char setequ-ext sup-lub*)
lemma *inf-glb*: $glb\ S\ \bigwedge S$ **unfolding** *glb-def lower-bounds-def infimum-def subset-def* **by** *blast*
lemma *inf-exist-unique*: $\forall S.\ \exists! X.\ glb\ S\ X$ **by** (*meson glb-def inf-glb setequ-char setequ-ext*)

The property of being closed under arbitrary (resp. nonempty) supremum/infimum.

definition *infimum-closed* :: $('w\ \sigma)\sigma \Rightarrow bool$
  where $infimum\text{-}closed\ S \equiv \forall D.\ D \leq S \longrightarrow S(\bigwedge D)$
definition *supremum-closed* :: $('w\ \sigma)\sigma \Rightarrow bool$
  where $supremum\text{-}closed\ S \equiv \forall D.\ D \leq S \longrightarrow S(\bigvee D)$
definition *infimum-closed′* :: $('w\ \sigma)\sigma \Rightarrow bool$
  where$infimum\text{-}closed'\ S \equiv \forall D.\ nonEmpty\ D \wedge D \leq S \longrightarrow S(\bigwedge D)$
definition *supremum-closed′* :: $('w\ \sigma)\sigma \Rightarrow bool$
  where $supremum\text{-}closed'\ S \equiv \forall D.\ nonEmpty\ D \wedge D \leq S \longrightarrow S(\bigvee D)$

lemma *inf-empty*: $isEmpty\ S \Longrightarrow \bigwedge S = \top$ **unfolding** *order conn* **by** *simp*
lemma *sup-empty*: $isEmpty\ S \Longrightarrow \bigvee S = \bot$ **unfolding** *order conn* **by** *simp*

Note that arbitrary infimum- (resp. supremum-) closed sets include the top (resp. bottom) element.

lemma $infimum\text{-}closed\ S \Longrightarrow S\ \top$ **unfolding** *infimum-closed-def conn order* **by** *force*
lemma $supremum\text{-}closed\ S \Longrightarrow S\ \bot$ **unfolding** *supremum-closed-def conn order* **by** *force*

However, the above does not hold for non-empty infimum- (resp. supremum-) closed sets.

lemma $infimum\text{-}closed'\ S \Longrightarrow S\ \top$ **nitpick oops** — countermodel
lemma $supremum\text{-}closed'\ S \Longrightarrow S\ \bot$ **nitpick oops** — countermodel

We have in fact the following characterizations for the notions above.

lemma *inf-closed-char*: $infimum\text{-}closed\ S = (infimum\text{-}closed'\ S \wedge S\ \top)$
 unfolding *infimum-closed′-def infimum-closed-def* **by** (*metis bottom-def infimum-closed-def infimum-def setequ-char setequ-ext subset-def top-def*)
lemma *sup-closed-char*: $supremum\text{-}closed\ S = (supremum\text{-}closed'\ S \wedge S\ \bot)$
  unfolding *supremum-closed′-def supremum-closed-def* **by** (*metis* (*no-types*, *opaque-lifting*) *L14 L9 bottom-def setequ-ext subset-def supremum-def*)

lemma *inf-sup-closed-dc*: $infimum\text{-}closed\ S = supremum\text{-}closed\ S^{d-}$ **by** (*smt* (*verit*) *BA-dn iDM-a iDM-b infimum-closed-def setequ-ext sdfun-dcompl-def subset-def supremum-closed-def*)
lemma *inf-sup-closed-dc′*: $infimum\text{-}closed'\ S = supremum\text{-}closed'\ S^{d-}$ **by** (*smt* (*verit*) *dualcompl-invol iDM-a infimum-closed′-def sdfun-dcompl-def setequ-ext subset-def supremum-closed′-def*)

We check some further properties.

lemma *fp-inf-sup-closed-dual*: *infimum-closed* (*fp* φ) = *supremum-closed* (*fp* φ^d)
  by (*simp add*: *fp-dual inf-sup-closed-dc*)
lemma *fp-inf-sup-closed-dual′*: *infimum-closed′* (*fp* φ) = *supremum-closed′* (*fp* φ^d)
  by (*simp add*: *fp-dual inf-sup-closed-dc′*)

We verify that being infimum-closed' (resp. supremum-closed') entails being meet-closed (resp. join-closed).

lemma *inf-meet-closed*: $\forall S.$ *infimum-closed′* $S \longrightarrow$ *meet-closed* S **proof** −
  { **fix** S::′w $\sigma \Rightarrow$ *bool*
    { **assume** *inf-closed*: *infimum-closed′* S
      hence *meet-closed* S **proof** −
        { **fix** X::′w σ **and** Y::′w σ
          let $?D{=}\lambda Z.\ Z{=}X \vee Z{=}Y$
          { **assume** $S\ X \wedge S\ Y$
            hence $?D \leq S$ **using** *subset-def* **by** *blast*
            moreover have *nonEmpty* $?D$ **by** *auto*
            ultimately have $S(\bigwedge ?D)$ **using** *inf-closed infimum-closed′-def* **by** (*smt* (*z3*))
            hence $S(\lambda w.\ \forall Z.\ (Z{=}X \vee Z{=}Y) \longrightarrow Z\ w)$ **unfolding** *infimum-def* **by** *simp*
            moreover have $(\lambda w.\ \forall Z.\ (Z{=}X \vee Z{=}Y) \longrightarrow Z\ w) = (\lambda w.\ X\ w \wedge Y\ w)$ **by** *auto*
            ultimately have $S(\lambda w.\ X\ w \wedge Y\ w)$ **by** *simp*
          } **hence** $(S\ X \wedge S\ Y) \longrightarrow S(X \wedge Y)$ **unfolding** *conn* **by** (*rule impI*)
        } **thus** *?thesis* **unfolding** *meet-closed-def* **by** *simp* **qed**
    } **hence** *infimum-closed′* $S \longrightarrow$ *meet-closed* S **by** *simp*
  } **thus** *?thesis* **by** (*rule allI*)
qed
lemma *sup-join-closed*: $\forall P.$ *supremum-closed′* $P \longrightarrow$ *join-closed* P **proof** −
  { **fix** S::′w $\sigma \Rightarrow$ *bool*
    { **assume** *sup-closed*: *supremum-closed′* S
      hence *join-closed* S **proof** −
        { **fix** X::′w σ **and** Y::′w σ
          let $?D{=}\lambda Z.\ Z{=}X \vee Z{=}Y$
          { **assume** $S\ X \wedge S\ Y$
            hence $?D \leq S$ **using** *subset-def* **by** *blast*
            moreover have *nonEmpty* $?D$ **by** *auto*
            ultimately have $S(\bigvee ?D)$ **using** *sup-closed supremum-closed′-def* **by** (*smt* (*z3*))
            hence $S(\lambda w.\ \exists Z.\ (Z{=}X \vee Z{=}Y) \wedge Z\ w)$ **unfolding** *supremum-def* **by** *simp*
            moreover have $(\lambda w.\ \exists Z.\ (Z{=}X \vee Z{=}Y) \wedge Z\ w) = (\lambda w.\ X\ w \vee Y\ w)$ **by** *auto*
            ultimately have $S(\lambda w.\ X\ w \vee Y\ w)$ **by** *simp*
          } **hence** $(S\ X \wedge S\ Y) \longrightarrow S(X \vee Y)$ **unfolding** *conn* **by** (*rule impI*)
        } **thus** *?thesis* **unfolding** *join-closed-def* **by** *simp* **qed**
    } **hence** *supremum-closed′* $S \longrightarrow$ *join-closed* S **by** *simp*
  } **thus** *?thesis* **by** (*rule allI*)
qed

end
theory *conditions-positive*
  imports *boolean-algebra-operators*
begin

2 Topological Conditions

We define and interrelate some useful axiomatic conditions on unary operations (operators) having a 'w-parametric type $('w)\sigma{\Rightarrow}('w)\sigma$. Boolean algebras extended with such operators give us different sorts of topological Boolean algebras.

2.1 Positive Conditions

Monotonicity (MONO).

definition $MONO$::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹MONO›)
where $MONO\ \varphi \equiv \forall A\ B.\ A \leq B \longrightarrow \varphi\ A \leq \varphi\ B$

named-theorems *cond*
declare *MONO-def*[*cond*]

MONO is self-dual.

lemma *MONO-dual*: $MONO\ \varphi = MONO\ \varphi^d$ **by** (*smt* (*verit*) *BA-cp MONO-def dual-invol op-dual-def*)

Expansive/extensive (EXPN) and its dual contractive (CNTR).

definition $EXPN$::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹EXPN›)
where $EXPN\ \varphi\ \equiv \forall A.\ A \leq \varphi\ A$
definition $CNTR$::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹CNTR›)
where $CNTR\ \varphi \equiv \forall A.\ \varphi\ A \leq A$

declare *EXPN-def*[*cond*] *CNTR-def*[*cond*]

EXPN and CNTR are dual to each other.

lemma *EXPN-CNTR-dual1*: $EXPN\ \varphi = CNTR\ \varphi^d$ **unfolding** *cond* **by** (*metis BA-cp BA-dn op-dual-def setequ-ext*)
lemma *EXPN-CNTR-dual2*: $CNTR\ \varphi = EXPN\ \varphi^d$ **by** (*simp add*: *EXPN-CNTR-dual1 dual-invol*)

Normality (NORM) and its dual (DNRM).

definition $NORM$::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹NORM›)
where $NORM\ \varphi\ \equiv (\varphi\ \bot) = \bot$
definition $DNRM$::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹DNRM›)
where $DNRM\ \varphi \equiv (\varphi\ \top) = \top$

declare *NORM-def*[*cond*] *DNRM-def*[*cond*]

NORM and DNRM are dual to each other.

lemma *NOR-dual1*: $NORM\ \varphi = DNRM\ \varphi^d$ **unfolding** *cond* **by** (*simp add*: *bottom-def compl-def op-dual-def setequ-def top-def*)
lemma *NOR-dual2*: $DNRM\ \varphi = NORM\ \varphi^d$ **by** (*simp add*: *NOR-dual1 dual-invol*)

EXPN (CNTR) entails DNRM (NORM).

lemma *EXPN-impl-DNRM*: $EXPN\ \varphi \longrightarrow DNRM\ \varphi$ **unfolding** *cond* **by** (*simp add*: *setequ-def subset-def top-def*)
lemma *CNTR-impl-NORM*: $CNTR\ \varphi \longrightarrow NORM\ \varphi$ **by** (*simp add*: *EXPN-CNTR-dual2 EXPN-impl-DNRM NOR-dual1 dual-invol*)

Idempotence (IDEM).

definition $IDEM$::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹IDEM›)
where $IDEM\ \varphi\ \equiv \forall A.\ \varphi(\varphi\ A) = (\varphi\ A)$
definition $IDEM\text{-}a$::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹$IDEM^a$›)
where $IDEM^a\ \varphi \equiv \forall A.\ \varphi(\varphi\ A) \leq (\varphi\ A)$
definition $IDEM\text{-}b$::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹$IDEM^b$›)

where $IDEM^b\ \varphi \equiv \forall A.\ \ (\varphi\ A) \leq \varphi(\varphi\ A)$

declare *IDEM-def*[*cond*] *IDEM-a-def*[*cond*] *IDEM-b-def*[*cond*]

IDEM-a and IDEM-b are dual to each other.

lemma *IDEM-dual1*: $IDEM^a\ \varphi = IDEM^b\ \varphi^d$ **unfolding** *cond* **by** (*metis* (*mono-tags*, *opaque-lifting*) *BA-cp BA-dn op-dual-def setequ-ext*)
lemma *IDEM-dual2*: $IDEM^b\ \varphi = IDEM^a\ \varphi^d$ **by** (*simp add*: *IDEM-dual1 dual-invol*)

lemma *IDEM-char*: $IDEM\ \varphi = (IDEM^a\ \varphi \wedge IDEM^b\ \varphi)$ **unfolding** *cond setequ-char* **by** *blast*
lemma *IDEM-dual*: $IDEM\ \varphi = IDEM\ \varphi^d$ **using** *IDEM-char IDEM-dual1 IDEM-dual2* **by** *blast*

EXPN (CNTR) entail IDEM-b (IDEM-a).

lemma *EXPN-impl-IDEM-b*: $EXPN\ \varphi \longrightarrow IDEM^b\ \varphi$ **by** (*simp add*: *EXPN-def IDEM-b-def*)
lemma *CNTR-impl-IDEM-a*: $CNTR\ \varphi \longrightarrow IDEM^a\ \varphi$ **by** (*simp add*: *CNTR-def IDEM-a-def*)

Moreover, IDEM has some other interesting characterizations. For example, via function composition:

lemma *IDEM-fun-comp-char*: $IDEM\ \varphi = (\varphi = \varphi \circ \varphi)$ **unfolding** *cond fun-comp-def* **by** (*metis setequ-ext*)

Or having the property of collapsing the range and the set of fixed-points of an operator:

lemma *IDEM-range-fp-char*: $IDEM\ \varphi = ([\![\varphi\ \text{-}]\!] = fp\ \varphi)$ **unfolding** *cond range-def fixpoints-def* **by** (*metis setequ-ext*)

Distribution over joins or additivity (ADDI).

definition $ADDI{::}('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹*ADDI*›)
  where $ADDI\ \varphi \quad \equiv \forall A\ B.\ \varphi(A \vee B) = (\varphi\ A) \vee (\varphi\ B)$
definition $ADDI\text{-}a{::}('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹$ADDI^a$›)
  where $ADDI^a\ \varphi \equiv \forall A\ B.\ \varphi(A \vee B) \leq (\varphi\ A) \vee (\varphi\ B)$
definition $ADDI\text{-}b{::}('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹$ADDI^b$›)
  where $ADDI^b\ \varphi \equiv \forall A\ B.\ \ (\varphi\ A) \vee (\varphi\ B) \leq \varphi(A \vee B)$

Distribution over meets or multiplicativity (MULT).

definition $MULT{::}('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹*MULT*›)
  where $MULT\ \varphi \quad \equiv \forall A\ B.\ \varphi(A \wedge B) = (\varphi\ A) \wedge (\varphi\ B)$
definition $MULT\text{-}a{::}('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹$MULT^a$›)
  where $MULT^a\ \varphi \equiv \forall A\ B.\ \varphi(A \wedge B) \leq (\varphi\ A) \wedge (\varphi\ B)$
definition $MULT\text{-}b{::}('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹$MULT^b$›)
  where $MULT^b\ \varphi \equiv \forall A\ B.\ (\varphi\ A) \wedge (\varphi\ B) \leq \varphi(A \wedge B)$

declare *ADDI-def*[*cond*] *ADDI-a-def*[*cond*] *ADDI-b-def*[*cond*]
    MULT-def[*cond*] *MULT-a-def*[*cond*] *MULT-b-def*[*cond*]

lemma *ADDI-char*: $ADDI\ \varphi = (ADDI^a\ \varphi \wedge ADDI^b\ \varphi)$ **unfolding** *cond* **using** *setequ-char* **by** *blast*
lemma *MULT-char*: $MULT\ \varphi = (MULT^a\ \varphi \wedge MULT^b\ \varphi)$ **unfolding** *cond* **using** *setequ-char* **by** *blast*

MONO, MULT-a and ADDI-b are equivalent.

lemma *MONO-MULTa*: $MULT^a\ \varphi = MONO\ \varphi$ **unfolding** *cond* **by** (*metis L10 L3 L4 L5 L8 setequ-char setequ-ext*)
lemma *MONO-ADDIb*: $ADDI^b\ \varphi = MONO\ \varphi$ **unfolding** *cond* **by** (*metis* (*mono-tags*, *lifting*) *L7 L9 join-def setequ-ext subset-def*)

Below we prove several duality relationships between ADDI(a/b) and MULT(a/b).

Duality between MULT-a and ADDI-b (an easy corollary from the self-duality of MONO).

lemma *MULTa-ADDIb-dual1*: $MULT^a\ \varphi = ADDI^b\ \varphi^d$ **by** (*metis MONO-ADDIb MONO-MULTa MONO-dual*)
lemma *MULTa-ADDIb-dual2*: $ADDI^b\ \varphi = MULT^a\ \varphi^d$ **by** (*simp add*: *MULTa-ADDIb-dual1 dual-invol*)

Duality between ADDI-a and MULT-b.

lemma *ADDIa-MULTb-dual1*: $ADDI^a\ \varphi = MULT^b\ \varphi^d$ **unfolding** *cond op-dual-def* **by** (*metis BA-cp BA-deMorgan1 BA-dn setequ-ext*)
lemma *ADDIa-MULTb-dual2*: $MULT^b\ \varphi = ADDI^a\ \varphi^d$ **by** (*simp add*: *ADDIa-MULTb-dual1 dual-invol*)

Duality between ADDI and MULT.

lemma *ADDI-MULT-dual1*: $ADDI\ \varphi = MULT\ \varphi^d$ **using** *ADDI-char ADDIa-MULTb-dual1 MULT-char MULTa-ADDIb-dual2* **by** *blast*
lemma *ADDI-MULT-dual2*: $MULT\ \varphi = ADDI\ \varphi^d$ **by** (*simp add*: *ADDI-MULT-dual1 dual-invol*)

We verify properties regarding closure over meets/joins for fixed-points.

MULT implies meet-closedness of the set of fixed-points (the converse requires additional assumptions).

lemma *MULT-meetclosed*: $MULT\ \varphi \Longrightarrow$ *meet-closed* (*fp* φ) **by** (*simp add*: *MULT-def fixpoints-def meet-closed-def setequ-ext*)
lemma *meet-closed* (*fp* φ) $\Longrightarrow MULT\ \varphi$ **nitpick oops** — countermodel found: needs further assumptions.
lemma *meetclosed-MULT*: $MONO\ \varphi \Longrightarrow CNTR\ \varphi \Longrightarrow IDEM^b\ \varphi \Longrightarrow$ *meet-closed* (*fp* φ) $\Longrightarrow MULT\ \varphi$ **by** (*smt* (*z3*) *CNTR-def IDEM-b-def MONO-MULTa MONO-def MULT-a-def MULT-def fixpoints-def meet-closed-def meet-def setequ-char setequ-ext subset-def*)

ADDI implies join-closedness of the set of fixed-points (the converse requires additional assumptions).

lemma *ADDI-joinclosed*: $ADDI\ \varphi \Longrightarrow$ *join-closed* (*fp* φ) **by** (*simp add*: *ADDI-def fixpoints-def join-closed-def setequ-ext*)
lemma *join-closed* (*fp* φ) $\Longrightarrow ADDI\ \varphi$ **nitpick oops** — countermodel found: needs further assumptions
lemma *joinclosed-ADDI*: $MONO\ \varphi \Longrightarrow EXPN\ \varphi \Longrightarrow IDEM^a\ \varphi \Longrightarrow$ *join-closed* (*fp* φ) $\Longrightarrow ADDI\ \varphi$ **by** (*smt* (*verit*, *ccfv-threshold*) *ADDI-MULT-dual1 BA-deMorgan2 EXPN-CNTR-dual1 IDEM-dual1 MONO-dual fp-dual join-closed-def meet-closed-def meetclosed-MULT sdfun-dcompl-def setequ-ext*)

Assuming MONO, we have that EXPN (CNTR) implies meet-closed (join-closed) for the set of fixed-points.

lemma *EXPN-meetclosed*: $MONO\ \varphi \Longrightarrow EXPN\ \varphi \Longrightarrow$ *meet-closed* (*fp* φ) **by** (*smt* (*verit*) *EXPN-def MONO-MULTa MULT-a-def fixpoints-def meet-closed-def setequ-char setequ-ext*)
lemma *CNTR-joinclosed*: $MONO\ \varphi \Longrightarrow CNTR\ \varphi \Longrightarrow$ *join-closed* (*fp* φ) **by** (*smt* (*verit*, *best*) *ADDI-b-def CNTR-def MONO-ADDIb fixpoints-def join-closed-def setequ-char setequ-ext*)

Further assuming IDEM the above results can be stated to the whole range of an operator.

lemma $MONO\ \varphi \Longrightarrow EXPN\ \varphi \Longrightarrow IDEM\ \varphi \Longrightarrow$ *meet-closed* ($[\![\varphi\ \text{-}]\!]$) **by** (*simp add*: *EXPN-meetclosed IDEM-range-fp-char*)
lemma $MONO\ \varphi \Longrightarrow CNTR\ \varphi \Longrightarrow IDEM\ \varphi \Longrightarrow$ *join-closed* ($[\![\varphi\ \text{-}]\!]$) **by** (*simp add*: *CNTR-joinclosed IDEM-range-fp-char*)

end
theory *conditions-positive-infinitary*
imports *conditions-positive boolean-algebra-infinitary*
begin

2.2 Infinitary Positive Conditions

We define and interrelate infinitary variants for some previously introduced axiomatic conditions on operators.

Distribution over infinite joins (suprema) or infinite additivity (iADDI).

definition *iADDI*::($'w\ \sigma \Rightarrow 'w\ \sigma$) $\Rightarrow$ *bool* (‹*iADDI*›)
  where *iADDI* $\varphi \quad \equiv \forall S.\ \varphi(\bigvee S) = \bigvee [\![\varphi\ S]\!]$
definition *iADDI-a*::($'w\ \sigma \Rightarrow 'w\ \sigma$) $\Rightarrow$ *bool* (‹$iADDI^a$›)
  where $iADDI^a\ \varphi \equiv \forall S.\ \varphi(\bigvee S) \leq \bigvee [\![\varphi\ S]\!]$
definition *iADDI-b*::($'w\ \sigma \Rightarrow 'w\ \sigma$) $\Rightarrow$ *bool* (‹$iADDI^b$›)
  where $iADDI^b\ \varphi \equiv \forall S.\ \bigvee [\![\varphi\ S]\!] \leq \varphi(\bigvee S)$

Distribution over infinite meets (infima) or infinite multiplicativity (iMULT).

definition *iMULT*::($'w\ \sigma \Rightarrow 'w\ \sigma$) $\Rightarrow$ *bool* (‹*iMULT*›)
  where *iMULT* $\varphi \quad \equiv \forall S.\ \varphi(\bigwedge S) = \bigwedge [\![\varphi\ S]\!]$
definition *iMULT-a*::($'w\ \sigma \Rightarrow 'w\ \sigma$) $\Rightarrow$ *bool* (‹$iMULT^a$›)
  where $iMULT^a\ \varphi \equiv \forall S.\ \varphi(\bigwedge S) \leq \bigwedge [\![\varphi\ S]\!]$
definition *iMULT-b*::($'w\ \sigma \Rightarrow 'w\ \sigma$) $\Rightarrow$ *bool* (‹$iMULT^b$›)
  where $iMULT^b\ \varphi \equiv \forall S.\ \bigwedge [\![\varphi\ S]\!] \leq \varphi(\bigwedge S)$

declare *iADDI-def*[*cond*] *iADDI-a-def*[*cond*] *iADDI-b-def*[*cond*]
    iMULT-def[*cond*] *iMULT-a-def*[*cond*] *iMULT-b-def*[*cond*]

lemma *iADDI-char*: *iADDI* $\varphi = (iADDI^a\ \varphi \wedge iADDI^b\ \varphi)$ **unfolding** *cond* **using** *setequ-char* **by** *blast*
lemma *iMULT-char*: *iMULT* $\varphi = (iMULT^a\ \varphi \wedge iMULT^b\ \varphi)$ **unfolding** *cond* **using** *setequ-char* **by** *blast*

ADDI-b and iADDI-b are in fact equivalent.

lemma *iADDIb-equ*: $iADDI^b\ \varphi = ADDI^b\ \varphi$ **proof** −
  have *lr*: $iADDI^b\ \varphi \Longrightarrow ADDI^b\ \varphi$ **proof** −
  assume *iaddib*: $iADDI^b\ \varphi$
  { fix $A::'a\ \sigma$ **and** $B::'a\ \sigma$
    let $?S{=}\lambda Z.\ Z{=}A \vee Z{=}B$
    have $\bigvee ?S = A \vee B$ **unfolding** *supremum-def join-def* **by** *blast*
    hence *p1*: $\varphi(\bigvee ?S) = \varphi(A \vee B)$ **by** *simp*
    have $[\![\varphi\ ?S]\!] = (\lambda Z.\ Z{=}(\varphi\ A) \vee Z{=}(\varphi\ B))$ **unfolding** *image-def* **by** *metis*
    hence *p2*: $\bigvee [\![\varphi\ ?S]\!] = (\varphi\ A) \vee (\varphi\ B)$ **unfolding** *supremum-def join-def* **by** *auto*
    have $\bigvee [\![\varphi\ ?S]\!] \leq \varphi(\bigvee ?S)$ **using** *iaddib iADDI-b-def* **by** *blast*
    hence $(\varphi\ A) \vee (\varphi\ B) \leq \varphi(A \vee B)$ **using** *p1 p2* **by** *simp*
  } thus *?thesis* **by** (*simp add*: *ADDI-b-def*) **qed**
  have *rl*: $ADDI^b\ \varphi \Longrightarrow iADDI^b\ \varphi$ **unfolding** *iADDI-b-def* **by** (*smt* (*verit*) *MONO-ADDIb MONO-def lub-def image-def sup-lub upper-bounds-def*)
  from *lr rl* **show** *?thesis* **by** *auto*
qed

MULT-a and iMULT-a are also equivalent.

lemma *iMULTa-equ*: $iMULT^a\ \varphi = MULT^a\ \varphi$ **proof** −
  have *lr*: $iMULT^a\ \varphi \Longrightarrow MULT^a\ \varphi$ **proof** −
  assume *imulta*: $iMULT^a\ \varphi$
  { fix $A::'a\ \sigma$ **and** $B::'a\ \sigma$
    let $?S{=}\lambda Z.\ Z{=}A \vee Z{=}B$
    have $\bigwedge ?S = A \wedge B$ **unfolding** *infimum-def meet-def* **by** *blast*
    hence *p1*: $\varphi(\bigwedge ?S) = \varphi(A \wedge B)$ **by** *simp*
    have $[\![\varphi\ ?S]\!] = (\lambda Z.\ Z{=}(\varphi\ A) \vee Z{=}(\varphi\ B))$ **unfolding** *image-def* **by** *metis*

hence *p2*: $\bigwedge[\![\varphi\ ?S]\!] = (\varphi\ A) \wedge (\varphi\ B)$ **unfolding** *infimum-def meet-def* **by** *auto*
have $\varphi(\bigwedge ?S) \leq \bigwedge[\![\varphi\ ?S]\!]$ **using** *imulta iMULT-a-def* **by** *blast*
hence $\varphi(A \wedge B) \leq (\varphi\ A) \wedge (\varphi\ B)$ **using** *p1 p2* **by** *simp*
} **thus** *?thesis* **by** (*simp add*: *MULT-a-def*) **qed**
have *rl*: $MULT^a\ \varphi \Longrightarrow iMULT^a\ \varphi$ **by** (*smt* (*verit*) *MONO-MULTa MONO-def glb-def iMULT-a-def inf-glb lower-bounds-def image-def*)
from *lr rl* **show** *?thesis* **by** *blast*
qed

Thus we have that MONO, ADDI-b/iADDI-b and MULT-a/iMULT-a are all equivalent.

lemma *MONO-iADDIb*: $iADDI^b\ \varphi = MONO\ \varphi$ **unfolding** *MONO-ADDIb iADDIb-equ* **by** *simp*
lemma *MONO-iMULTa*: $iMULT^a\ \varphi = MONO\ \varphi$ **unfolding** *MONO-MULTa iMULTa-equ* **by** *simp*

Below we prove several duality relationships between iADDI(a/b) and iMULT(a/b).

Duality between iMULT-a and iADDI-b (an easy corollary from the previous equivalence).

lemma *iMULTa-iADDIb-dual1*: $iMULT^a\ \varphi = iADDI^b\ \varphi^d$ **by** (*simp add*: *MULTa-ADDIb-dual1 iADDIb-equ iMULTa-equ*)
lemma *iMULTa-iADDIb-dual2*: $iADDI^b\ \varphi = iMULT^a\ \varphi^d$ **by** (*simp add*: *MULTa-ADDIb-dual2 iADDIb-equ iMULTa-equ*)

Duality between iADDI-a and iMULT-b.

lemma *iADDIa-iMULTb-dual1*: $iADDI^a\ \varphi = iMULT^b\ \varphi^d$ **by** (*smt* (*z3*) *BA-cmpl-equ BA-cp dual-compl-invol iADDI-a-def iDM-a iMULT-b-def im-prop1 op-dual-def setequ-ext*)
lemma *iADDIa-iMULTb-dual2*: $iMULT^b\ \varphi = iADDI^a\ \varphi^d$ **by** (*simp add*: *dual-invol iADDIa-iMULTb-dual1*)

Duality between iADDI and iMULT.

lemma *iADDI-iMULT-dual1*: $iADDI\ \varphi = iMULT\ \varphi^d$ **using** *iADDI-char iADDIa-iMULTb-dual1 iMULT-char iMULTa-iADDIb-dual2* **by** *blast*
lemma *iADDI-iMULT-dual2*: $iMULT\ \varphi = iADDI\ \varphi^d$ **by** (*simp add*: *dual-invol iADDI-iMULT-dual1*)

In fact, infinite additivity (multiplicativity) entails (dual) normality:

lemma *iADDI-NORM*: $iADDI\ \varphi \longrightarrow NORM\ \varphi$ **unfolding** *cond* **by** (*metis bottom-def image-def setequ-ext sup-empty*)
lemma *iMULT-DNRM*: $iMULT\ \varphi \longrightarrow DNRM\ \varphi$ **by** (*simp add*: *NOR-dual2 iADDI-NORM iADDI-iMULT-dual2*)

Suitable conditions on an operation can make the set of its fixed-points closed under infinite meets/joins.

lemma *fp-sup-closed-cond1*: $iADDI\ \varphi \longrightarrow$ *supremum-closed* (*fp* φ) **unfolding** *cond order supremum-closed-def fixpoints-def image-def* **by** (*smt* (*verit*) *supremum-def*)
lemma *fp-sup-closed-cond2*: $iADDI^a\ \varphi \wedge EXPN\ \varphi \longrightarrow$ *supremum-closed* (*fp* φ) **unfolding** *cond* **by** (*smt* (*z3*) *fixpoints-def image-def setequ-char subset-def supremum-closed-def supremum-def*)
lemma *fp-sup-closed-cond3*: $MONO\ \varphi \wedge CNTR\ \varphi \longrightarrow$ *supremum-closed* (*fp* φ) **unfolding** *cond* **by** (*smt* (*verit*, *del-insts*) *fixpoints-def lub-def setequ-char setequ-ext subset-def sup-lub supremum-closed-def upper-bounds-def*)

lemma *fp-inf-closed-cond1*: $iMULT\ \varphi \longrightarrow$ *infimum-closed* (*fp* φ) **by** (*metis fp-dual fp-sup-closed-cond1 iADDI-iMULT-dual2 inf-sup-closed-dc*)
lemma *fp-inf-closed-cond2*: $iMULT^b\ \varphi \wedge CNTR\ \varphi \longrightarrow$ *infimum-closed* (*fp* φ) **by** (*metis EXPN-CNTR-dual2 fp-dual fp-sup-closed-cond2 iADDIa-iMULTb-dual2 inf-sup-closed-dc*)
lemma *fp-inf-closed-cond3*: $MONO\ \varphi \wedge EXPN\ \varphi \longrightarrow$ *infimum-closed* (*fp* φ) **by** (*metis EXPN-CNTR-dual1 MONO-dual fp-dual fp-sup-closed-cond3 inf-sup-closed-dc*)

OTOH, the converse conjectures have (finite) countermodels. They need additional assumptions.

lemma *infimum-closed* (*fp* φ) $\longrightarrow$ *iMULT* φ **nitpick oops** — countermodel found: needs further assumptions
lemma *supremum-closed* (*fp* φ) $\longrightarrow$ *iADDI* φ **nitpick oops** — countermodel found: needs further assumptions
lemma *fp-inf-closed-iMULT*: *MONO* $\varphi \Longrightarrow$ *CNTR* $\varphi \Longrightarrow$ *IDEM*^{b} $\varphi \Longrightarrow$ *infimum-closed* (*fp* φ) $\longrightarrow$ *iMULT* φ
proof –
assume *mono*: *MONO* φ **and** *cntr*: *CNTR* φ **and** *idem*:*IDEM*^{b} φ **{**
 assume *ic*:*infimum-closed* (*fp* φ) **{**
  fix S
  from *ic* **have** $\forall D.\ D \leq (fp\ \varphi) \longrightarrow (fp\ \varphi)(\bigwedge D)$ **unfolding** *infimum-closed-def* **by** *simp*
  hence *let* D=$[\![\varphi\ S]\!]$ *in* $(\forall X.\ D\ X \longrightarrow (X = \varphi\ X)) \longrightarrow \bigwedge D = \varphi \bigwedge D$ **by** (*simp add*: *fixpoints-def setequ-ext subset-def*)
  moreover from *idem* **have** $(\forall X.\ [\![\varphi\ S]\!]\ X \longrightarrow (X = \varphi\ X))$ **by** (*metis* (*mono-tags*, *lifting*) *CNTR-def IDEM-b-def cntr image-def setequ-char*)
  ultimately have *aux*: $\bigwedge([\![\varphi\ S]\!]) = \varphi(\bigwedge[\![\varphi\ S]\!])$ **by** *meson*
  from *cntr* **have** $\bigwedge[\![\varphi\ S]\!] \leq \bigwedge S$ **by** (*smt* (*verit*, *best*) *CNTR-def image-def infimum-def subset-def*)
  hence $\varphi(\bigwedge[\![\varphi\ S]\!]) \leq \varphi(\bigwedge S)$ **using** *mono* **by** (*simp add*: *MONO-def*)
  from *this aux* **have** $\bigwedge[\![\varphi\ S]\!] \leq \varphi(\bigwedge S)$ **by** (*simp add*: *setequ-ext*)
 } **hence** *infimum-closed* (*fp* φ) $\longrightarrow$ *iMULT* φ **by** (*simp add*: *MONO-iMULTa iMULT-b-def iMULT-char mono*)
} **thus** *?thesis* **by** *simp*
qed
lemma *fp-sup-closed-iADDI*: *MONO* $\varphi \Longrightarrow$ *EXPN* $\varphi \Longrightarrow$ *IDEM*^{a} $\varphi \Longrightarrow$ *supremum-closed* (*fp* φ) $\longrightarrow$ *iADDI* φ

proof –
assume *mono*: *MONO* φ **and** *expn*: *EXPN* φ **and** *idem*:*IDEM*^{a} φ **{**
 assume *sc*:*supremum-closed* (*fp* φ) **{**
  fix S
  from *sc* **have** $\forall D.\ D \leq (fp\ \varphi) \longrightarrow (fp\ \varphi)(\bigvee D)$ **unfolding** *supremum-closed-def* **by** *simp*
  hence *let* D=$[\![\varphi\ S]\!]$ *in* $(\forall X.\ D\ X \longrightarrow (X = \varphi\ X)) \longrightarrow \bigvee D = \varphi \bigvee D$ **by** (*simp add*: *fixpoints-def setequ-ext subset-def*)
  moreover have $(\forall X.\ [\![\varphi\ S]\!]\ X \longrightarrow (X = \varphi\ X))$ **by** (*metis* (*mono-tags*, *lifting*) *EXPN-def IDEM-a-def expn idem image-def setequ-char*)
  ultimately have *aux*: $\bigvee([\![\varphi\ S]\!]) = \varphi(\bigvee[\![\varphi\ S]\!])$ **by** *meson*
  have $\bigvee S \leq \bigvee[\![\varphi\ S]\!]$ **by** (*metis EXPN-CNTR-dual1 EXPN-def IDEM-dual1 MONO-dual dual-invol expn fp-inf-closed-iMULT fp-inf-sup-closed-dual iADDI-def iADDI-iMULT-dual1 idem mono sc setequ-ext*)
  hence $\varphi(\bigvee S) \leq \varphi(\bigvee[\![\varphi\ S]\!])$ **using** *mono* **by** (*simp add*: *MONO-def*)
  from *this aux* **have** $\varphi(\bigvee S) \leq \bigvee[\![\varphi\ S]\!]$ **by** (*metis setequ-ext*)
 } **hence** *supremum-closed* (*fp* φ) $\longrightarrow$ *iADDI* φ **by** (*simp add*: *MONO-ADDIb iADDI-a-def iADDI-char iADDIb-equ mono*)
} **thus** *?thesis* **by** *simp*
qed

3 Alexandrov topologies and (generalized) specialization preorders

A topology is called 'Alexandrov' (after the Russian mathematician Pavel Alexandrov) if the intersection (resp. union) of any (finite or infinite) family of open (resp. closed) sets is open (resp. closed); in algebraic terms, this means that the set of fixed points of the interior (closure) operation is closed under infinite meets (joins). Another common algebraic formulation requires the closure (interior) operation to satisfy the infinitary variants of additivity (multiplicativity), i.e. iADDI (iMULT) as introduced before.

In the literature, the well-known Kuratowski conditions for the closure (resp. interior) operation are assumed, namely: ADDI, EXPN, NORM, IDEM (resp. MULT, CNTR, DNRM, IDEM). This makes both formulations equivalent. However, this is not the case in general if those conditions become negotiable.

Alexandrov topologies have interesting properties relating them to the semantics of modal logic. Assuming Kuratowski conditions, Alexandrov topological operations defined on subsets of S are in one-to-one correspondence with preorders on S; in topological terms, Alexandrov topologies are uniquely determined by their specialization preorders. Since we do not presuppose any Kuratowski conditions to begin with, the preorders in question are in general not even transitive. Here we just call them 'reachability relations'. We will still call (generalized) closure/interior-like operations as such (for lack of a better name). We explore minimal conditions under which some relevant results for the semantics of modal logic obtain.

Closure/interior(-like) operators can be derived from an arbitrary relation (as in modal logic).

definition *Cl-rel*::$('w \Rightarrow 'w \Rightarrow bool) \Rightarrow ('w\ \sigma \Rightarrow 'w\ \sigma)$ (‹$\mathcal{C}$[-]›)
  where $\mathcal{C}[R] \equiv \lambda A.\ \lambda w.\ \exists v.\ R\ w\ v \wedge A\ v$
definition *Int-rel*::$('w \Rightarrow 'w \Rightarrow bool) \Rightarrow ('w\ \sigma \Rightarrow 'w\ \sigma)$ (‹$\mathcal{I}$[-]›)
  where $\mathcal{I}[R] \equiv \lambda A.\ \lambda w.\ \forall v.\ R\ w\ v \longrightarrow A\ v$

Duality between interior and closure follows directly:

lemma *dual-CI*: $\mathcal{C}[R] = \mathcal{I}[R]^d$ **by** (*simp add*: *Cl-rel-def Int-rel-def compl-def op-dual-def setequ-char*)

We explore minimal conditions of the reachability relation under which some operation's conditions obtain.

Define some relevant properties of relations:

abbreviation *serial* $R \equiv \forall x.\ \exists y.\ R\ x\ y$
abbreviation *reflexive* $R \equiv \forall x.\ R\ x\ x$
abbreviation *transitive* $R \equiv \forall x\ y\ z.\ R\ x\ y \wedge R\ y\ z \longrightarrow R\ x\ z$
abbreviation *antisymmetric* $R \equiv \forall x\ y.\ R\ x\ y \wedge R\ y\ x \longrightarrow x = y$
abbreviation *symmetric* $R \equiv \forall x\ y.\ R\ x\ y \longrightarrow R\ y\ x$

lemma *rC1*: *iADDI* $\mathcal{C}[R]$ **unfolding** *iADDI-def Cl-rel-def image-def supremum-def* **using** *setequ-def* **by** *fastforce*
lemma *rC2*: *reflexive* R = *EXPN* $\mathcal{C}[R]$ **by** (*metis* (*full-types*) *CNTR-def EXPN-CNTR-dual2 Int-rel-def dual-CI subset-def*)
lemma *rC3*: *NORM* $\mathcal{C}[R]$ **by** (*simp add*: *iADDI-NORM rC1*)
lemma *rC4*: *transitive* R = $IDEM^a$ $\mathcal{C}[R]$ **proof** −
  have *l2r*: *transitive* $R \longrightarrow IDEM^a$ $\mathcal{C}[R]$ **by** (*smt* (*verit*, *best*) *Cl-rel-def IDEM-a-def subset-def*)
  have *r2l*: $IDEM^a$ $\mathcal{C}[R] \longrightarrow$ *transitive* R **unfolding** *Cl-rel-def IDEM-a-def subset-def* **using** *compl-def* **by** *force*
  from *l2r r2l* **show** *?thesis* **by** *blast*
qed

A reachability (specialization) relation (preorder) can be derived from a given operation (intended as a closure-like operation).

definition $\mathcal{R}$::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow ('w \Rightarrow 'w \Rightarrow bool)$ (‹$\mathcal{R}$[-]›)
  where $\mathcal{R}[\varphi] \equiv \lambda w\ v.\ \varphi\ (\lambda x.\ x = v)\ w$

Preorder properties of the reachability relation follow from the corresponding operation's conditions.

lemma *rel-refl*: $EXPN\ \varphi \longrightarrow reflexive\ \mathcal{R}[\varphi]$ **by** (*simp add*: *EXPN-def* $\mathcal{R}$*-def subset-def*)
lemma *rel-trans*: $MONO\ \varphi \wedge IDEM^a\ \varphi \longrightarrow transitive\ \mathcal{R}[\varphi]$ **by** (*smt* (*verit*, *best*) *IDEM-a-def MONO-def* $\mathcal{R}$*-def subset-def*)
lemma $IDEM^a\ \varphi \longrightarrow transitive\ \mathcal{R}[\varphi]$ **nitpick oops** — counterexample

lemma $reflexive\ \mathcal{R}[\varphi] \longrightarrow EXPN\ \varphi$ **nitpick oops** — counterexample
lemma $transitive\ \mathcal{R}[\varphi] \longrightarrow IDEM^a\ \varphi$ **nitpick oops** — counterexample
lemma $transitive\ \mathcal{R}[\varphi] \longrightarrow MONO\ \varphi$ **nitpick oops** — counterexample

However, we can obtain finite countermodels for antisymmetry and symmetry given all relevant conditions. We will revisit this issue later and examine their relation with the topological separation axioms T0 and T1 resp.

lemma $iADDI\ \varphi \Longrightarrow EXPN\ \varphi \Longrightarrow IDEM^a\ \varphi \Longrightarrow antisymmetric\ \mathcal{R}[\varphi]$ **nitpick oops** — counterexample
lemma $iADDI\ \varphi \Longrightarrow EXPN\ \varphi \Longrightarrow IDEM^a\ \varphi \Longrightarrow symmetric\ \mathcal{R}[\varphi]$ **nitpick oops** — counterexample

As mentioned previously, Alexandrov closure (and by duality interior) operations correspond to specialization orderings (reachability relations). It is worth mentioning that in Alexandrov topologies every point has a minimal/smallest neighborhood, namely the set of points related to it by the specialization preorder (reachability relation). We examine below minimal conditions under which these relations obtain.

lemma *Cl-rel′-a*: $MONO\ \varphi \longrightarrow (\forall A.\ \mathcal{C}[\mathcal{R}[\varphi]]\ A \leq \varphi\ A)$ **unfolding** *Cl-rel-def MONO-def* $\mathcal{R}$*-def* **by** (*smt* (*verit*, *ccfv-SIG*) *subset-def*)
lemma *Cl-rel′-b*: $iADDI^a\ \varphi \longrightarrow (\forall A.\ \varphi\ A \leq \mathcal{C}[\mathcal{R}[\varphi]]\ A)$ **proof** −
{ assume *iaddia*: $iADDI^a\ \varphi$
 { fix $A::'a\ \sigma$
  let $?S=\lambda B.\ \exists w.\ A\ w \wedge B=(\lambda u.\ u=w)$
  have $A = (\bigvee ?S)$ **unfolding** *supremum-def setequ-def* **by** *auto*
  hence $\varphi(A) = \varphi(\bigvee ?S)$ **by** (*simp add*: *setequ-ext*)
  moreover have $\bigvee[\![\varphi\ ?S]\!] = \mathcal{C}[\mathcal{R}[\varphi]]\ A$ **by** (*smt* (*verit*) *Cl-rel-def* $\mathcal{R}$*-def image-def setequ-def supremum-def*)
  moreover from *iaddia* **have** $\varphi(\bigvee ?S) \leq \bigvee[\![\varphi\ ?S]\!]$ **unfolding** *iADDI-a-def* **by** *simp*
  ultimately have $\varphi\ A \leq \mathcal{C}[\mathcal{R}[\varphi]]\ A$ **by** (*simp add*: *setequ-ext*)
} } thus *?thesis* **by** *simp*
qed
lemma *Cl-rel′*: $iADDI\ \varphi \longrightarrow \varphi =^{:} \mathcal{C}[\mathcal{R}[\varphi]]$ **by** (*simp add*: *MONO-iADDIb iADDI-char setequ-char Cl-rel′-a Cl-rel′-b svfun-equ-def*)
lemma *Cl-rel*: $iADDI\ \varphi \longleftrightarrow \varphi = \mathcal{C}[\mathcal{R}[\varphi]]$ **using** *Cl-rel′* **by** (*metis rC1 svfun-equ-ext*)

It is instructive to expand the definitions in the above result:

lemma $iADDI\ \varphi \longleftrightarrow (\forall A.\ \forall w.\ (\varphi\ A)\ w \longleftrightarrow (\exists v.\ A\ v \wedge \varphi\ (\lambda x.\ x{=}v)\ w))$ **oops**

Closure (interior) operations derived from relations are thus closed under infinite joins (meets).

lemma *supremum-closed* $(fp\ \mathcal{C}[R])$ **by** (*simp add*: *fp-sup-closed-cond1 rC1*)
lemma *infimum-closed* $(fp\ \mathcal{I}[R])$ **by** (*metis dual-CI fp-inf-closed-cond1 iADDI-iMULT-dual2 rC1*)

We can now revisit the relationship between (anti)symmetry and the separation axioms T1 and T0.

T0: any two distinct points in the space can be separated by a closed (or open) set (i.e. containing one point and not the other).

abbreviation $T0\ \mathcal{C} \equiv (\forall p\ q.\ p \neq q \longrightarrow (\exists G.\ (fp\ \mathcal{C})\ G \wedge \neg(G\ p \longleftrightarrow G\ q)))$

T1: any two distinct points can be separated by (two not necessarily disjoint) closed (or open) sets.

abbreviation $T1\ \mathcal{C} \equiv (\forall\, p\ q.\ p \neq q \longrightarrow (\exists\, G\ H.\ (fp\ \mathcal{C})\ G \wedge (fp\ \mathcal{C})\ H \wedge G\ p \wedge \neg G\ q \wedge H\ q \wedge \neg H\ p))$

We can (sanity) check that T1 entails T0 but not viceversa.

lemma $T1\ \mathcal{C} \Longrightarrow T0\ \mathcal{C}$ **by** *meson*
lemma $T0\ \mathcal{C} \Longrightarrow T1\ \mathcal{C}$ **nitpick oops** — counterexample

Under appropriate conditions, T0-separation corresponds to antisymmetry of the specialization relation (here an ordering).

lemma $T0\ \mathcal{C} \longleftrightarrow antisymmetric\ \mathcal{R}[\mathcal{C}]$ **nitpick oops**
lemma *T0-antisymm-a*: $MONO\ \mathcal{C} \Longrightarrow T0\ \mathcal{C} \longrightarrow antisymmetric\ \mathcal{R}[\mathcal{C}]$ **by** (*smt* (*verit*, *best*) *Cl-rel′-a Cl-rel-def fixpoints-def setequ-ext subset-def*)
lemma *T0-antisymm-b*: $EXPN\ \mathcal{C} \Longrightarrow IDEM^a\ \mathcal{C} \Longrightarrow antisymmetric\ \mathcal{R}[\mathcal{C}] \longrightarrow T0\ \mathcal{C}$ **by** (*metis EXPN-impl-IDEM-b IDEM-char IDEM-def* $\mathcal{R}$*-def fixpoints-def rel-refl*)
lemma *T0-antisymm*: $MONO\ \mathcal{C} \Longrightarrow EXPN\ \mathcal{C} \Longrightarrow IDEM^a\ \mathcal{C} \Longrightarrow T0\ \mathcal{C} = antisymmetric\ \mathcal{R}[\mathcal{C}]$ **using** *T0-antisymm-a T0-antisymm-b* **by** *fastforce*

Also, under the appropriate conditions, T1-separation corresponds to symmetry of the specialization relation.

lemma *T1-symm-a*: $MONO\ \mathcal{C} \Longrightarrow T1\ \mathcal{C} \longrightarrow symmetric\ \mathcal{R}[\mathcal{C}]$ **by** (*metis* (*mono-tags*, *opaque-lifting*) *Cl-rel′-a Cl-rel-def fixpoints-def setequ-ext subset-def*)
lemma *T1-symm-b*: $MONO\ \mathcal{C} \Longrightarrow EXPN\ \mathcal{C} \Longrightarrow T0\ \mathcal{C} \Longrightarrow symmetric\ \mathcal{R}[\mathcal{C}] \longrightarrow T1\ \mathcal{C}$ **by** (*smt* (*verit*, *ccfv-SIG*) *T0-antisymm-a* $\mathcal{R}$*-def fixpoints-def rel-refl setequ-def*)
lemma *T1-symm*: $MONO\ \mathcal{C} \Longrightarrow EXPN\ \mathcal{C} \Longrightarrow T0\ \mathcal{C} \Longrightarrow symmetric\ \mathcal{R}[\mathcal{C}] = T1\ \mathcal{C}$ **using** *T1-symm-a T1-symm-b* **by** (*smt* (*verit*, *ccfv-threshold*))

end
theory *conditions-negative*
  imports *conditions-positive*
begin

3.1 Negative Conditions

We continue defining and interrelating axiomatic conditions on unary operations (operators). We now move to conditions commonly satisfied by negation-like logical operations.

Anti-tonicity (ANTI).

definition $ANTI::('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹$ANTI$›)
  where $ANTI\ \varphi \equiv \forall\, A\ B.\ A \leq B \longrightarrow \varphi\ B \leq \varphi\ A$

declare *ANTI-def*[*cond*]

ANTI is self-dual.

lemma *ANTI-dual*: $ANTI\ \varphi = ANTI\ \varphi^d$ **by** (*smt* (*verit*) *BA-cp ANTI-def dual-invol op-dual-def*)

ANTI is the 'complement' of MONO.

lemma *ANTI-MONO*: $MONO\ \varphi = ANTI\ \varphi^-$ **by** (*metis ANTI-def BA-cp MONO-def svfun-compl-def*)

Anti-expansive/extensive (nEXPN) and its dual anti-contractive (nCNTR).

definition $nEXPN::('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹$nEXPN$›)
  where $nEXPN\ \varphi \ \equiv \forall\, A.\ \varphi\ A \leq -A$
definition $nCNTR::('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹$nCNTR$›)
  where $nCNTR\ \varphi \equiv \forall\, A.\ -A \leq \varphi\ A$

declare *nEXPN-def*[*cond*] *nCNTR-def*[*cond*]

nEXPN and nCNTR are dual to each other.

lemma *nEXPN-nCNTR-dual1*: *nEXPN* φ = *nCNTR* φ^d **unfolding** *cond* **by** (*metis BA-cp BA-dn op-dual-def setequ-ext*)
lemma *nEXPN-nCNTR-dual2*: *nCNTR* φ = *nEXPN* φ^d **by** (*simp add*: *dual-invol nEXPN-nCNTR-dual1*)

nEXPN and nCNTR are the 'complements' of EXPN and CNTR respectively.

lemma *nEXPN-CNTR-compl*: *EXPN* φ = *nEXPN* φ^- **by** (*metis BA-cp EXPN-def nEXPN-def sv-fun-compl-def*)
lemma *nCNTR-EXPN-compl*: *CNTR* φ = *nCNTR* φ^- **by** (*metis EXPN-CNTR-dual2 dual-compl-char1 dual-compl-char2 nEXPN-CNTR-compl nEXPN-nCNTR-dual2*)

Anti-Normality (nNORM) and its dual (nDNRM).

definition *nNORM*::($'w\ \sigma \Rightarrow 'w\ \sigma$) $\Rightarrow$ *bool* (‹*nNORM*›)
  where *nNORM* $\varphi \equiv (\varphi\ \bot) = \top$
definition *nDNRM*::($'w\ \sigma \Rightarrow 'w\ \sigma$) $\Rightarrow$ *bool* (‹*nDNRM*›)
  where *nDNRM* $\varphi \equiv (\varphi\ \top) = \bot$

declare *nNORM-def*[*cond*] *nDNRM-def*[*cond*]

nNORM and nDNRM are dual to each other.

lemma *nNOR-dual1*: *nNORM* φ = *nDNRM* φ^d **unfolding** *cond* **by** (*simp add*: *bottom-def compl-def op-dual-def setequ-def top-def*)
lemma *nNOR-dual2*: *nDNRM* φ = *nNORM* φ^d **by** (*simp add*: *dual-invol nNOR-dual1*)

nNORM and nDNRM are the 'complements' of NORM and DNRM respectively.

lemma *nNORM-NORM-compl*: *NORM* φ = *nNORM* φ^- **by** (*simp add*: *NORM-def bottom-def compl-def nNORM-def setequ-def svfun-compl-def top-def*)
lemma *nDNRM-DNRM-compl*: *DNRM* φ = *nDNRM* φ^- **by** (*simp add*: *DNRM-def bottom-def compl-def nDNRM-def setequ-def svfun-compl-def top-def*)

nEXPN (nCNTR) entail nDNRM (nNORM).

lemma *nEXPN-impl-nDNRM*: *nEXPN* $\varphi \longrightarrow$ *nDNRM* φ **unfolding** *cond* **by** (*metis bottom-def compl-def setequ-def subset-def top-def*)
lemma *nCNTR-impl-nNORM*: *nCNTR* $\varphi \longrightarrow$ *nNORM* φ **by** (*simp add*: *nEXPN-impl-nDNRM nEXPN-nCNTR-dual2 nNOR-dual1*)

Anti-Idempotence (nIDEM).

definition *nIDEM*::($'w\ \sigma \Rightarrow 'w\ \sigma$) $\Rightarrow$ *bool* (‹*nIDEM*›)
  where *nIDEM* $\varphi \equiv \forall A.\ \varphi(-(\varphi\ A)) = (\varphi\ A)$
definition *nIDEM-a*::($'w\ \sigma \Rightarrow 'w\ \sigma$) $\Rightarrow$ *bool* (‹$nIDEM^a$›)
  where *nIDEM-a* $\varphi \equiv \forall A.\ (\varphi\ A) \preceq \varphi(-(\varphi\ A))$
definition *nIDEM-b*::($'w\ \sigma \Rightarrow 'w\ \sigma$) $\Rightarrow$ *bool* (‹$nIDEM^b$›)
  where *nIDEM-b* $\varphi \equiv \forall A.\ \varphi(-(\varphi\ A)) \preceq (\varphi\ A)$

declare *nIDEM-def*[*cond*] *nIDEM-a-def*[*cond*] *nIDEM-b-def*[*cond*]

nIDEM-a and nIDEM-b are dual to each other.

lemma *nIDEM-dual1*: $nIDEM^a\ \varphi = nIDEM^b\ \varphi^d$ **unfolding** *cond* **by** (*metis BA-cp BA-dn op-dual-def setequ-ext*)
lemma *nIDEM-dual2*: $nIDEM^b\ \varphi = nIDEM^a\ \varphi^d$ **by** (*simp add*: *dual-invol nIDEM-dual1*)

lemma *nIDEM-char*: $nIDEM\ \varphi = (nIDEM^a\ \varphi \wedge nIDEM^b\ \varphi)$ **unfolding** *cond setequ-char* **by** *blast*
lemma *nIDEM-dual*: $nIDEM\ \varphi = nIDEM\ \varphi^d$ **using** *nIDEM-char nIDEM-dual1 nIDEM-dual2* **by** *blast*

nIDEM(a/b) and IDEM(a/b) are the 'complements' each other.

lemma *nIDEM-a-compl*: $IDEM^a\ \varphi = nIDEM^a\ \varphi^-$ **by** (*metis* (*no-types*, *lifting*) *BA-cp IDEM-a-def nIDEM-a-def sfun-compl-invol svfun-compl-def*)
lemma *nIDEM-b-compl*: $IDEM^b\ \varphi = nIDEM^b\ \varphi^-$ **by** (*metis IDEM-dual2 dual-compl-char1 dual-compl-char2 nIDEM-a-compl nIDEM-dual2*)
lemma *nIDEM-compl*: $nIDEM\ \varphi = IDEM\ \varphi^-$ **by** (*simp add*: *IDEM-char nIDEM-a-compl nIDEM-b-compl nIDEM-char sfun-compl-invol*)

nEXPN (nCNTR) entail nIDEM-a (nIDEM-b).

lemma *nEXPN-impl-nIDEM-a*: $nEXPN\ \varphi \longrightarrow nIDEM^b\ \varphi$ **by** (*metis nEXPN-def nIDEM-b-def sfun-compl-invol svfun-compl-def*)
lemma *nCNTR-impl-nIDEM-b*: $nCNTR\ \varphi \longrightarrow nIDEM^a\ \varphi$ **by** (*simp add*: *nEXPN-impl-nIDEM-a nEXPN-nCNTR-dual2 nIDEM-dual1*)

Anti-distribution over joins or anti-additivity (nADDI) and its dual.

definition $nADDI$::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ $(\langle nADDI\rangle)$
  where $nADDI\ \varphi\ \equiv \forall A\ B.\ \varphi(A \vee B) = (\varphi\ A) \wedge (\varphi\ B)$
definition $nADDI\text{-}a$::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ $(\langle nADDI^a\rangle)$
  where $nADDI^a\ \varphi \equiv \forall A\ B.\ (\varphi\ A) \wedge (\varphi\ B) \leq \varphi(A \vee B)$
definition $nADDI\text{-}b$::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ $(\langle nADDI^b\rangle)$
  where $nADDI^b\ \varphi \equiv \forall A\ B.\ \varphi(A \vee B) \leq (\varphi\ A) \wedge (\varphi\ B)$

Anti-distribution over meets or anti-multiplicativity (nMULT).

definition $nMULT$::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ $(\langle nMULT\rangle)$
  where $nMULT\ \varphi\ \equiv \forall A\ B.\ \varphi(A \wedge B) = (\varphi\ A) \vee (\varphi\ B)$
definition $nMULT\text{-}a$::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ $(\langle nMULT^a\rangle)$
  where $nMULT^a\ \varphi \equiv \forall A\ B.\ (\varphi\ A) \vee (\varphi\ B) \leq \varphi(A \wedge B)$
definition $nMULT\text{-}b$::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ $(\langle nMULT^b\rangle)$
  where $nMULT^b\ \varphi \equiv \forall A\ B.\ \varphi(A \wedge B) \leq (\varphi\ A) \vee (\varphi\ B)$

declare *nADDI-def*[*cond*] *nADDI-a-def*[*cond*] *nADDI-b-def*[*cond*]
    nMULT-def[*cond*] *nMULT-a-def*[*cond*] *nMULT-b-def*[*cond*]

lemma *nADDI-char*: $nADDI\ \varphi = (nADDI^a\ \varphi \wedge nADDI^b\ \varphi)$ **unfolding** *cond* **using** *setequ-char* **by** *blast*
lemma *nMULT-char*: $nMULT\ \varphi = (nMULT^a\ \varphi \wedge nMULT^b\ \varphi)$ **unfolding** *cond* **using** *setequ-char* **by** *blast*

ANTI, nMULT-a and nADDI-b are equivalent.

lemma *ANTI-nMULTa*: $nMULT^a\ \varphi = ANTI\ \varphi$ **unfolding** *cond* **by** (*smt* (*z3*) *L10 L7 join-def meet-def setequ-ext subset-def*)
lemma *ANTI-nADDIb*: $nADDI^b\ \varphi = ANTI\ \varphi$ **unfolding** *cond* **by** (*smt* (*verit*) *BA-cp BA-deMorgan1 L10 L3 L5 L8 L9 setequ-char setequ-ext*)

Below we prove several duality relationships between nADDI(a/b) and nMULT(a/b).

Duality between nMULT-a and nADDI-b (an easy corollary from the self-duality of ANTI).

lemma *nMULTa-nADDIb-dual1*: $nMULT^a\ \varphi = nADDI^b\ \varphi^d$ **using** *ANTI-nADDIb ANTI-nMULTa ANTI-dual* **by** *blast*
lemma *nMULTa-nADDIb-dual2*: $nADDI^b\ \varphi = nMULT^a\ \varphi^d$ **by** (*simp add*: *dual-invol nMULTa-nADDIb-dual1*)

Duality between nADDI-a and nMULT-b.

lemma *nADDIa-nMULTb-dual1*: *nADDI*^{a} φ = *nMULT*^{b} φ^d **unfolding** *cond* **by** (*metis* (*no-types*, *lifting*) *BA-cp BA-deMorgan1 BA-dn op-dual-def setequ-ext*)
lemma *nADDIa-nMULTb-dual2*: *nMULT*^{b} φ = *nADDI*^{a} φ^d **by** (*simp add*: *dual-invol nADDIa-nMULTb-dual1*)

Duality between ADDI and MULT.

lemma *nADDI-nMULT-dual1*: *nADDI* φ = *nMULT* φ^d **using** *nADDI-char nADDIa-nMULTb-dual1 nMULT-char nMULTa-nADDIb-dual2* **by** *blast*
lemma *nADDI-nMULT-dual2*: *nMULT* φ = *nADDI* φ^d **by** (*simp add*: *dual-invol nADDI-nMULT-dual1*)

nADDI and nMULT are the 'complements' of ADDI and MULT respectively.

lemma *nADDIa-compl*: *ADDI*^{a} φ = *nADDI*^{a} φ^- **by** (*metis ADDI-a-def BA-cp BA-deMorgan1 nADDI-a-def setequ-ext svfun-compl-def*)
lemma *nADDIb-compl*: *ADDI*^{b} φ = *nADDI*^{b} φ^- **by** (*simp add*: *ANTI-nADDIb ANTI-MONO MONO-ADDIb sfun-compl-invol*)
lemma *nADDI-compl*: *ADDI* φ = *nADDI* φ^- **by** (*simp add*: *ADDI-char nADDI-char nADDIa-compl nADDIb-compl*)
lemma *nMULTa-compl*: *MULT*^{a} φ = *nMULT*^{a} φ^- **by** (*simp add*: *ANTI-MONO ANTI-nMULTa MONO-MULTa sfun-compl-invol*)
lemma *nMULTb-compl*: *MULT*^{b} φ = *nMULT*^{b} φ^- **by** (*metis BA-cp BA-deMorgan2 MULT-b-def nMULT-b-def setequ-ext svfun-compl-def*)
lemma *nMULT-compl*: *MULT* φ = *nMULT* φ^- **by** (*simp add*: *MULT-char nMULT-char nMULTa-compl nMULTb-compl*)

We verify properties regarding closure over meets/joins for fixed-points.

nMULT for an operator implies join-closedness of the set of fixed-points of its dual-complement.

lemma *nMULT-joinclosed*: *nMULT* $\varphi \Longrightarrow$ *join-closed* (*fp* (φ^{d-})) **by** (*smt* (*verit*, *del-insts*) *ADDI-MULT-dual2 ADDI-joinclosed BA-deMorgan1 MULT-def dual-compl-char2 nMULT-def setequ-ext svfun-compl-def*)
lemma *join-closed* (*fp* (φ^{d-})) $\Longrightarrow$ *nMULT* φ **nitpick oops** — countermodel found: needs further assumptions
lemma *joinclosed-nMULT*: *ANTI* $\varphi \Longrightarrow$ *nCNTR* $\varphi \Longrightarrow$ *nIDEM*^{b} $\varphi \Longrightarrow$ *join-closed* (*fp* (φ^{d-})) $\Longrightarrow$ *nMULT* φ **by** (*metis ANTI-MONO ANTI-dual IDEM-char IDEM-dual dual-compl-char1 dual-compl-char2 joinclosed-ADDI nADDI-compl nADDI-nMULT-dual2 nCNTR-impl-nIDEM-b nEXPN-CNTR-compl nEXPN-nCNTR-du nIDEM-char nIDEM-compl sfun-compl-invol*)

nADDI for an operator implies meet-closedness of the set of fixed-points of its dual-complement.

lemma *nADDI-meetclosed*: *nADDI* $\varphi \Longrightarrow$ *meet-closed* (*fp* (φ^{d-})) **by** (*smt* (*verit*, *ccfv-threshold*) *ADDI-MULT-dual1 ADDI-def BA-deMorgan2 MULT-meetclosed dual-compl-char2 nADDI-def setequ-ext svfun-compl-def*)
lemma *meet-closed* (*fp* (φ^{d-})) $\Longrightarrow$ *nADDI* φ **nitpick oops** — countermodel found: needs further assumptions
lemma *meetclosed-nADDI*: *ANTI* $\varphi \Longrightarrow$ *nEXPN* $\varphi \Longrightarrow$ *nIDEM*^{a} $\varphi \Longrightarrow$ *meet-closed* (*fp* (φ^{d-})) $\Longrightarrow$ *nADDI* φ **by** (*metis ADDI-MULT-dual2 ADDI-joinclosed ANTI-MONO ANTI-dual dual-compl-char1 dual-compl-char2 joinclosed-nMULT meetclosed-MULT nADDI-nMULT-dual1 nCNTR-EXPN-compl nEXPN-nCNTR-du nIDEM-b-compl nIDEM-dual1 sfun-compl-invol*)

Assuming ANTI, we have that nEXPN (nCNTR) implies meet-closed (join-closed) for the set of fixed-points.

lemma *nEXPN-meetclosed*: *ANTI* $\varphi \Longrightarrow$ *nEXPN* $\varphi \Longrightarrow$ *meet-closed* (*fp* φ) **by** (*metis* (*full-types*) *L10 compl-def fixpoints-def meet-closed-def nEXPN-def setequ-ext subset-def*)
lemma *nCNTR-joinclosed*: *ANTI* $\varphi \Longrightarrow$ *nCNTR* $\varphi \Longrightarrow$ *join-closed* (*fp* φ) **by** (*smt* (*verit*, *ccfv-threshold*) *BA-impl L9 fixpoints-def impl-char join-closed-def nCNTR-def setequ-char setequ-ext*)

end
theory *conditions-negative-infinitary*
imports *conditions-negative conditions-positive-infinitary*
begin

3.2 Infinitary Negative Conditions

We define and interrelate infinitary variants for some previously introduced axiomatic conditions on operators.

Anti-distribution over infinite joins (suprema) or infinite anti-additivity (inADDI).

definition *inADDI*::($'w\ \sigma \Rightarrow 'w\ \sigma$) $\Rightarrow$ *bool* (‹*inADDI*›)
  where *inADDI* $\varphi \ \equiv \forall S.\ \varphi(\bigvee S) = \bigwedge [\![\varphi\ S]\!]$
definition *inADDI-a*::($'w\ \sigma \Rightarrow 'w\ \sigma$) $\Rightarrow$ *bool* (‹*inADDI*^{a}›)
  where *inADDI*^{a} $\varphi \equiv \forall S.\ \bigwedge [\![\varphi\ S]\!] \leq \varphi(\bigvee S)$
definition *inADDI-b*::($'w\ \sigma \Rightarrow 'w\ \sigma$) $\Rightarrow$ *bool* (‹*inADDI*^{b}›)
  where *inADDI*^{b} $\varphi \equiv \forall S.\ \varphi(\bigvee S) \leq \bigwedge [\![\varphi\ S]\!]$

Anti-distribution over infinite meets (infima) or infinite anti-multiplicativity (inMULT).

definition *inMULT*::($'w\ \sigma \Rightarrow 'w\ \sigma$) $\Rightarrow$ *bool* (‹*inMULT*›)
  where *inMULT* $\varphi \ \equiv \forall S.\ \varphi(\bigwedge S) = \bigvee [\![\varphi\ S]\!]$
definition *inMULT-a*::($'w\ \sigma \Rightarrow 'w\ \sigma$) $\Rightarrow$ *bool* (‹*inMULT*^{a}›)
  where *inMULT*^{a} $\varphi \equiv \forall S.\ \bigvee [\![\varphi\ S]\!] \leq \varphi(\bigwedge S)$
definition *inMULT-b*::($'w\ \sigma \Rightarrow 'w\ \sigma$) $\Rightarrow$ *bool* (‹*inMULT*^{b}›)
  where *inMULT*^{b} $\varphi \equiv \forall S.\ \varphi(\bigwedge S) \leq \bigvee [\![\varphi\ S]\!]$

declare *inADDI-def*[*cond*] *inADDI-a-def*[*cond*] *inADDI-b-def*[*cond*]
    inMULT-def[*cond*] *inMULT-a-def*[*cond*] *inMULT-b-def*[*cond*]

lemma *inADDI-char*: *inADDI* φ = (*inADDI*^{a} $\varphi \wedge$ *inADDI*^{b} φ) **unfolding** *cond* **using** *setequ-char* **by** *blast*
lemma *inMULT-char*: *inMULT* φ = (*inMULT*^{a} $\varphi \wedge$ *inMULT*^{b} φ) **unfolding** *cond* **using** *setequ-char* **by** *blast*

nADDI-b and inADDI-b are in fact equivalent.

lemma *inADDIb-equ*: *inADDI*^{b} φ = *nADDI*^{b} φ **proof** −
  have *lr*: *inADDI*^{b} $\varphi \Longrightarrow$ *nADDI*^{b} φ **proof** −
  assume *inaddib*: *inADDI*^{b} φ
  { fix *A*::$'a\ \sigma$ **and** *B*::$'a\ \sigma$
    let $?S{=}\lambda Z.\ Z{=}A \vee Z{=}B$
    have $\bigvee ?S = A \vee B$ **unfolding** *supremum-def join-def* **by** *blast*
    hence *p1*: $\varphi(\bigvee ?S) = \varphi(A \vee B)$ **by** *simp*
    have $[\![\varphi\ ?S]\!] = (\lambda Z.\ Z{=}(\varphi\ A) \vee Z{=}(\varphi\ B))$ **unfolding** *image-def* **by** *metis*
    hence *p2*: $\bigwedge [\![\varphi\ ?S]\!] = (\varphi\ A) \wedge (\varphi\ B)$ **unfolding** *infimum-def meet-def* **by** *auto*
    have $\varphi(\bigvee ?S) \leq \bigwedge [\![\varphi\ ?S]\!]$ **using** *inaddib inADDI-b-def* **by** *blast*
    hence $\varphi(A \vee B) \leq (\varphi\ A) \wedge (\varphi\ B)$ **using** *p1 p2* **by** *simp*
  } thus *?thesis* **by** (*simp add*: *nADDI-b-def*) **qed**
  have *rl*: *nADDI*^{b} $\varphi \Longrightarrow$ *inADDI*^{b} φ **unfolding** *inADDI-b-def ANTI-nADDIb ANTI-def image-def*
    by (*smt* (*verit*) *glb-def inf-glb lower-bounds-def lub-def sup-lub upper-bounds-def*)
  from *lr rl* **show** *?thesis* **by** *auto*
qed

nMULT-a and inMULT-a are also equivalent.

lemma *inMULTa-equ*: *inMULT*^{a} φ = *nMULT*^{a} φ **proof** −
  have *lr*: *inMULT*^{a} $\varphi \Longrightarrow$ *nMULT*^{a} φ **proof** −
  assume *inmulta*: *inMULT*^{a} φ
  { fix *A*::$'a\ \sigma$ **and** *B*::$'a\ \sigma$
    let $?S{=}\lambda Z.\ Z{=}A \vee Z{=}B$
    have $\bigwedge ?S = A \wedge B$ **unfolding** *infimum-def meet-def* **by** *blast*
    hence *p1*: $\varphi(\bigwedge ?S) = \varphi(A \wedge B)$ **by** *simp*

have $[\![\varphi\ ?S]\!] = (\lambda Z.\ Z{=}(\varphi\ A) \lor Z{=}(\varphi\ B))$ **unfolding** *image-def* **by** *metis*
hence *p2*: $\bigvee[\![\varphi\ ?S]\!] = (\varphi\ A) \lor (\varphi\ B)$ **unfolding** *supremum-def join-def* **by** *auto*
have $\bigvee[\![\varphi\ ?S]\!] \leq \varphi(\bigwedge ?S)$ **using** *inmulta inMULT-a-def* **by** *blast*
hence $(\varphi\ A) \lor (\varphi\ B) \leq \varphi(A \land B)$ **using** *p1 p2* **by** *simp*
} **thus** *?thesis* **by** (*simp add*: *nMULT-a-def*) **qed**
have *rl*: $nMULT^a\ \varphi \Longrightarrow inMULT^a\ \varphi$ **unfolding** *inMULT-a-def ANTI-nMULTa ANTI-def image-def*
by (*smt* (*verit*) *glb-def inf-glb lower-bounds-def lub-def sup-lub upper-bounds-def*)
from *lr rl* **show** *?thesis* **by** *blast*
qed

Thus we have that ANTI, nADDI-b/inADDI-b and nMULT-a/inMULT-a are all equivalent.

lemma *ANTI-inADDIb*: $inADDI^b\ \varphi = ANTI\ \varphi$ **unfolding** *ANTI-nADDIb inADDIb-equ* **by** *simp*
lemma *ANTI-inMULTa*: $inMULT^a\ \varphi = ANTI\ \varphi$ **unfolding** *ANTI-nMULTa inMULTa-equ* **by** *simp*

Below we prove several duality relationships between inADDI(a/b) and inMULT(a/b).

Duality between inMULT-a and inADDI-b (an easy corollary from the previous equivalence).

lemma *inMULTa-inADDIb-dual1*: $inMULT^a\ \varphi = inADDI^b\ \varphi^d$ **by** (*simp add*: *nMULTa-nADDIb-dual1 inADDIb-equ inMULTa-equ*)
lemma *inMULTa-inADDIb-dual2*: $inADDI^b\ \varphi = inMULT^a\ \varphi^d$ **by** (*simp add*: *nMULTa-nADDIb-dual2 inADDIb-equ inMULTa-equ*)

Duality between inADDI-a and inMULT-b.

lemma *inADDIa-inMULTb-dual1*: $inADDI^a\ \varphi = inMULT^b\ \varphi^d$ **by** (*smt* (*z3*) *BA-cmpl-equ BA-cp dualcompl-invol inADDI-a-def iDM-a inMULT-b-def im-prop1 op-dual-def setequ-ext*)
lemma *inADDIa-inMULTb-dual2*: $inMULT^b\ \varphi = inADDI^a\ \varphi^d$ **by** (*simp add*: *dual-invol inADDIa-inMULTb-dual1*)

Duality between inADDI and inMULT.

lemma *inADDI-inMULT-dual1*: $inADDI\ \varphi = inMULT\ \varphi^d$ **using** *inADDI-char inADDIa-inMULTb-dual1 inMULT-char inMULTa-inADDIb-dual2* **by** *blast*
lemma *inADDI-inMULT-dual2*: $inMULT\ \varphi = inADDI\ \varphi^d$ **by** (*simp add*: *dual-invol inADDI-inMULT-dual1*)

inADDI and inMULT are the 'complements' of iADDI and iMULT respectively.

lemma *inADDIa-compl*: $iADDI^a\ \varphi = inADDI^a\ \varphi^-$ **by** (*metis BA-cmpl-equ BA-cp iADDI-a-def iDM-a im-prop2 inADDI-a-def setequ-ext svfun-compl-def*)
lemma *inADDIb-compl*: $iADDI^b\ \varphi = inADDI^b\ \varphi^-$ **by** (*simp add*: *ANTI-MONO ANTI-inADDIb MONO-iADDIb*)
lemma *inADDI-compl*: $iADDI\ \varphi = inADDI\ \varphi^-$ **by** (*simp add*: *iADDI-char inADDI-char inADDIa-compl inADDIb-compl*)
lemma *inMULTa-compl*: $iMULT^a\ \varphi = inMULT^a\ \varphi^-$ **by** (*simp add*: *ANTI-MONO ANTI-inMULTa MONO-iMULTa*)
lemma *inMULTb-compl*: $iMULT^b\ \varphi = inMULT^b\ \varphi^-$ **by** (*metis dual-compl-char1 dual-compl-char2 iADDIa-iMULTb-dual2 inADDIa-compl inADDIa-inMULTb-dual2*)
lemma *inMULT-compl*: $iMULT\ \varphi = inMULT\ \varphi^-$ **by** (*simp add*: *iMULT-char inMULT-char inMULTa-compl inMULTb-compl*)

In fact, infinite anti-additivity (anti-multiplicativity) entails (dual) anti-normality:

lemma *inADDI-nNORM*: $inADDI\ \varphi \longrightarrow nNORM\ \varphi$ **by** (*metis bottom-def inADDI-def inf-empty image-def nNORM-def setequ-ext sup-empty*)
lemma *inMULT-nDNRM*: $inMULT\ \varphi \longrightarrow nDNRM\ \varphi$ **by** (*simp add*: *inADDI-inMULT-dual2 inADDI-nNORM nNOR-dual2*)

end
theory *conditions-relativized*
imports *conditions-negative*
begin

3.3 Relativized Conditions

We continue defining and interrelating axiomatic conditions on unary operations (operators). This time we consider their 'relativized' variants.

Relativized order and equality relations.

definition *subset-in*::‹$'p\ \sigma \Rightarrow\ 'p\ \sigma \Rightarrow\ 'p\ \sigma \Rightarrow bool$› (‹$\text{-}\leq_{\text{-}}\text{-}$›)
  where ‹$A \leq_U B \equiv \forall x.\ U\ x \longrightarrow (A\ x \longrightarrow B\ x)$›
definition *subset-out*::‹$'p\ \sigma \Rightarrow\ 'p\ \sigma \Rightarrow\ 'p\ \sigma \Rightarrow bool$› (‹$\text{-}\leq^{\text{-}}\text{-}$›)
  where ‹$A \leq^U B \equiv \forall x.\ \neg U\ x \longrightarrow (A\ x \longrightarrow B\ x)$›
definition *setequ-in*::‹$'p\ \sigma \Rightarrow\ 'p\ \sigma \Rightarrow\ 'p\ \sigma \Rightarrow bool$› (‹$\text{-}=_{\text{-}}\text{-}$›)
  where ‹$A =_U B \equiv \forall x.\ U\ x \longrightarrow (A\ x \longleftrightarrow B\ x)$›
definition *setequ-out*::‹$'p\ \sigma \Rightarrow\ 'p\ \sigma \Rightarrow\ 'p\ \sigma \Rightarrow bool$› (‹$\text{-}=^{\text{-}}\text{-}$›)
  where ‹$A =^U B \equiv \forall x.\ \neg U\ x \longrightarrow (A\ x \longleftrightarrow B\ x)$›

declare *subset-in-def*[*order*] *subset-out-def*[*order*] *setequ-in-def*[*order*] *setequ-out-def*[*order*]

lemma *subset-in-out*: ($let\ U{=}C\ in\ (A \leq_U B)) = (let\ U{=}{-}C\ in\ (A \leq^U B)$) **by** (*simp add*: *compl-def subset-in-def subset-out-def*)
lemma *setequ-in-out*: ($let\ U{=}C\ in\ (A =_U B)) = (let\ U{=}{-}C\ in\ (A =^U B)$) **by** (*simp add*: *compl-def setequ-in-def setequ-out-def*)

lemma *subset-in-char*: $(A \leq_U B) = (U \wedge A \leq U \wedge B)$ **unfolding** *order conn* **by** *blast*
lemma *subset-out-char*: $(A \leq^U B) = (U \vee A \leq U \vee B)$ **unfolding** *order conn* **by** *blast*
lemma *setequ-in-char*: $(A =_U B) = (U \wedge A = U \wedge B)$ **unfolding** *order conn* **by** *blast*
lemma *setequ-out-char*: $(A =^U B) = (U \vee A = U \vee B)$ **unfolding** *order conn* **by** *blast*

Relativization cannot be meaningfully applied to conditions (n)NORM or (n)DNRM.

lemma $NORM\ \varphi\ = (let\ U = \top\ in\ ((\varphi\ \bot) =_U \bot))$ **by** (*simp add*: *NORM-def setequ-def setequ-in-def top-def*)
lemma $(let\ U = \bot\ in\ ((\varphi\ \bot) =_U \bot))$ **by** (*simp add*: *bottom-def setequ-in-def*)

Relativization ('in' resp. 'out') leaves (n)EXPN/(n)CNTR unchanged or trivializes them.

lemma $EXPN\ \varphi\ = (\forall A.\ A \leq_A \varphi\ A)$ **by** (*simp add*: *EXPN-def subset-def subset-in-def*)
lemma $CNTR\ \varphi\ = (\forall A.\ (\varphi\ A) \leq^A A)$ **by** (*metis* (*mono-tags*, *lifting*) *CNTR-def subset-def subset-out-def*)
lemma $\forall A.\ A \leq^A \varphi\ A$ **by** (*simp add*: *subset-out-def*)
lemma $\forall A.\ (\varphi\ A) \leq_A A$ **by** (*simp add*: *subset-in-def*)

Relativized ADDI variants.

definition *ADDIr*::$('w\ \sigma \Rightarrow\ 'w\ \sigma) \Rightarrow bool$ (‹$ADDIr$›)
  where $ADDIr\ \varphi\ \equiv \forall A\ B.\ let\ U = (A \vee B)\ in\ \ (\varphi(A \vee B) =^U (\varphi\ A) \vee (\varphi\ B))$
definition *ADDIr-a*::$('w\ \sigma \Rightarrow\ 'w\ \sigma) \Rightarrow bool$ (‹$ADDIr^a$›)
  where $ADDIr^a\ \varphi \equiv \forall A\ B.\ let\ U = (A \vee B)\ in\ \ (\varphi(A \vee B) \leq^U (\varphi\ A) \vee (\varphi\ B))$
definition *ADDIr-b*::$('w\ \sigma \Rightarrow\ 'w\ \sigma) \Rightarrow bool$ (‹$ADDIr^b$›)
  where $ADDIr^b\ \varphi \equiv \forall A\ B.\ let\ U = (A \vee B)\ in\ ((\varphi\ A) \vee (\varphi\ B) \leq^U \varphi(A \vee B))$

declare *ADDIr-def*[*cond*] *ADDIr-a-def*[*cond*] *ADDIr-b-def*[*cond*]

lemma *ADDIr-char*: $ADDIr\ \varphi = (ADDIr^a\ \varphi \wedge ADDIr^b\ \varphi)$ **unfolding** *cond* **by** (*meson setequ-char setequ-out-char subset-out-char*)

lemma *ADDIr-a-impl*: $ADDI^a\ \varphi \longrightarrow ADDIr^a\ \varphi$ **by** (*simp add*: *ADDI-a-def ADDIr-a-def subset-def subset-out-def*)

lemma *ADDIr-a-equ*: $EXPN\ \varphi \Longrightarrow ADDIr^a\ \varphi = ADDI^a\ \varphi$ **unfolding** *cond* **by** (*smt* (*verit*, *del-insts*) *join-def subset-def subset-out-def*)
lemma *ADDIr-a-equ′*:$nEXPN\ \varphi \Longrightarrow ADDIr^a\ \varphi = ADDI^a\ \varphi$ **unfolding** *cond* **by** (*smt* (*verit*, *ccfv-threshold*) *compl-def subset-def subset-out-def*)

lemma *ADDIr-b-impl*: $ADDI^b\ \varphi \longrightarrow ADDIr^b\ \varphi$ **by** (*simp add*: *ADDI-b-def ADDIr-b-def subset-def subset-out-def*)
lemma $nEXPN\ \varphi \Longrightarrow ADDIr^b\ \varphi \longrightarrow ADDI^b\ \varphi$ **nitpick oops** — countermodel
lemma *ADDIr-b-equ*: $EXPN\ \varphi \Longrightarrow ADDIr^b\ \varphi = ADDI^b\ \varphi$ **unfolding** *cond* **by** (*smt* (*z3*) *subset-def subset-out-def*)

Relativized MULT variants.

definition *MULTr*::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹*MULTr*›)
where $MULTr\ \varphi\ \equiv \forall A\ B.\ let\ U = (A \wedge B)\ in\ \ (\varphi(A \wedge B) =_U (\varphi\ A) \wedge (\varphi\ B))$
definition *MULTr-a*::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹$MULTr^a$›)
where $MULTr^a\ \varphi \equiv \forall A\ B.\ let\ U = (A \wedge B)\ in\ \ (\varphi(A \wedge B) \leq_U (\varphi\ A) \wedge (\varphi\ B))$
definition *MULTr-b*::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹$MULTr^b$›)
where $MULTr^b\ \varphi \equiv \forall A\ B.\ let\ U = (A \wedge B)\ in\ ((\varphi\ A) \wedge (\varphi\ B) \leq_U \varphi(A \wedge B))$

declare *MULTr-def*[*cond*] *MULTr-a-def*[*cond*] *MULTr-b-def*[*cond*]

lemma *MULTr-char*: $MULTr\ \varphi = (MULTr^a\ \varphi \wedge MULTr^b\ \varphi)$ **unfolding** *cond* **by** (*meson setequ-char setequ-in-char subset-in-char*)

lemma *MULTr-a-impl*: $MULT^a\ \varphi \longrightarrow MULTr^a\ \varphi$ **by** (*simp add*: *MULT-a-def MULTr-a-def subset-def subset-in-def*)
lemma $nCNTR\ \varphi \Longrightarrow MULTr^a\ \varphi \longrightarrow MULT^a\ \varphi$ **nitpick oops** — countermodel
lemma *MULTr-a-equ*: $CNTR\ \varphi \Longrightarrow MULTr^a\ \varphi = MULT^a\ \varphi$ **unfolding** *cond* **by** (*smt* (*verit*, *del-insts*) *subset-def subset-in-def*)

lemma *MULTr-b-impl*: $MULT^b\ \varphi \longrightarrow MULTr^b\ \varphi$ **by** (*simp add*: *MULT-b-def MULTr-b-def subset-def subset-in-def*)
lemma $MULTr^b\ \varphi \longrightarrow MULT^b\ \varphi$ **nitpick oops** — countermodel
lemma *MULTr-b-equ*: $CNTR\ \varphi \Longrightarrow MULTr^b\ \varphi = MULT^b\ \varphi$ **unfolding** *cond* **by** (*smt* (*verit*, *del-insts*) *meet-def subset-def subset-in-def*)
lemma *MULTr-b-equ′*:$nCNTR\ \varphi \Longrightarrow MULTr^b\ \varphi = MULT^b\ \varphi$ **unfolding** *cond* **by** (*smt* (*z3*) *compl-def subset-def subset-in-def*)

Weak variants of monotonicity.

definition *MONOw1*::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹$MONOw^1$›)
where $MONOw^1\ \varphi \equiv \forall A\ B.\ A \leq B \longrightarrow (\varphi\ A) \leq B \vee (\varphi\ B)$
definition *MONOw2*::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹$MONOw^2$›)
where $MONOw^2\ \varphi \equiv \forall A\ B.\ A \leq B \longrightarrow A \wedge (\varphi\ A) \leq (\varphi\ B)$

declare *MONOw1-def*[*cond*] *MONOw2-def*[*cond*]

lemma *MONOw1-ADDIr-b*: $MONOw^1\ \varphi = ADDIr^b\ \varphi$ **proof** −
have *l2r*: $MONOw^1\ \varphi \longrightarrow ADDIr^b\ \varphi$ **unfolding** *cond subset-out-char* **by** (*metis* (*mono-tags*, *opaque-lifting*) *L7 join-def subset-def*)
have *r2l*: $ADDIr^b\ \varphi \longrightarrow MONOw^1\ \varphi$ **unfolding** *cond subset-out-char* **by** (*metis* (*full-types*) *L9 join-def setequ-ext subset-def*)
show *?thesis* **using** *l2r r2l* **by** *blast*
qed
lemma *MONOw2-MULTr-a*: $MONOw^2\ \varphi = MULTr^a\ \varphi$ **proof** −
have *l2r*: $MONOw^2\ \varphi \longrightarrow MULTr^a\ \varphi$ **unfolding** *cond subset-in-char* **by** (*meson L4 L5 L8 L9*)

have *r2l*:$MULTr^a$ $\varphi \longrightarrow MONOw^2$ φ **unfolding** *cond subset-in-char* **by** (*metis BA-distr1 L2 L5 L6 L9 setequ-ext*)
show *?thesis* **using** *l2r r2l* **by** *blast*
qed

lemma *MONOw1-impl*: $MONO\ \varphi \longrightarrow MONOw^1\ \varphi$ **by** (*simp add*: *ADDIr-b-impl MONO-ADDIb MONOw1-ADDIr-b*)
lemma $MONOw^1\ \varphi \longrightarrow MONO\ \varphi$ **nitpick oops** — countermodel
lemma *MONOw2-impl*: $MONO\ \varphi \longrightarrow MONOw^2\ \varphi$ **by** (*simp add*: *MONO-MULTa MONOw2-MULTr-a MULTr-a-impl*)
lemma $MONOw^2\ \varphi \longrightarrow MONO\ \varphi$ **nitpick oops** — countermodel

We have in fact that (n)CNTR (resp. (n)EXPN) implies MONOw-1/ADDIr-b (resp. MONOw-2/MULTr-a).

lemma *CNTR-MONOw1-impl*: $CNTR\ \varphi \longrightarrow MONOw^1\ \varphi$ **by** (*metis CNTR-def L3 MONOw1-def subset-char1*)
lemma *nCNTR-MONOw1-impl*: $nCNTR\ \varphi \longrightarrow MONOw^1\ \varphi$ **by** (*smt* (*verit*, *ccfv-threshold*) *MONOw1-def compl-def join-def nCNTR-def subset-def*)
lemma *EXPN-MONOw2-impl*: $EXPN\ \varphi \longrightarrow MONOw^2\ \varphi$ **by** (*metis EXPN-def L4 MONOw2-def subset-char1*)
lemma *nEXPN-MONOw2-impl*: $nEXPN\ \varphi \longrightarrow MONOw^2\ \varphi$ **by** (*smt* (*verit*) *MONOw2-def compl-def meet-def nEXPN-def subset-def*)

Relativized nADDI variants.

definition *nADDIr*::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹*nADDIr*›)
where $nADDIr\ \varphi \equiv \forall A\ B.\ let\ U = (A \vee B)\ in\ \ (\varphi(A \vee B) =^U (\varphi\ A) \wedge (\varphi\ B))$
definition *nADDIr-a*::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹$nADDIr^a$›)
where $nADDIr^a\ \varphi \equiv \forall A\ B.\ let\ U = (A \vee B)\ in\ ((\varphi\ A) \wedge (\varphi\ B) \leq^U \varphi(A \vee B))$
definition *nADDIr-b*::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹$nADDIr^b$›)
where $nADDIr^b\ \varphi \equiv \forall A\ B.\ let\ U = (A \vee B)\ in\ \ (\varphi(A \vee B) \leq^U (\varphi\ A) \wedge (\varphi\ B))$

declare *nADDIr-def*[*cond*] *nADDIr-a-def*[*cond*] *nADDIr-b-def*[*cond*]

lemma *nADDIr-char*: $nADDIr\ \varphi = (nADDIr^a\ \varphi \wedge nADDIr^b\ \varphi)$ **unfolding** *cond* **by** (*meson setequ-char setequ-out-char subset-out-char*)

lemma *nADDIr-a-impl*: $nADDI^a\ \varphi \longrightarrow nADDIr^a\ \varphi$ **unfolding** *cond* **by** (*simp add*: *subset-def subset-out-def*)
lemma $nADDIr^a\ \varphi \longrightarrow nADDI^a\ \varphi$ **nitpick oops** — countermodel
lemma *nADDIr-a-equ*: $EXPN\ \varphi \Longrightarrow nADDIr^a\ \varphi = nADDI^a\ \varphi$ **unfolding** *cond* **by** (*smt* (*z3*) *subset-def subset-out-def*)
lemma *nADDIr-a-equ′*:$nEXPN\ \varphi \Longrightarrow nADDIr^a\ \varphi = nADDI^a\ \varphi$ **unfolding** *cond* **by** (*smt* (*z3*) *compl-def join-def meet-def subset-def subset-out-def*)

lemma *nADDIr-b-impl*: $nADDI^b\ \varphi \longrightarrow nADDIr^b\ \varphi$ **by** (*simp add*: *nADDI-b-def nADDIr-b-def subset-def subset-out-def*)
lemma $EXPN\ \varphi \Longrightarrow nADDIr^b\ \varphi \longrightarrow nADDI^b\ \varphi$ **nitpick oops** — countermodel
lemma *nADDIr-b-equ*: $nEXPN\ \varphi \Longrightarrow nADDIr^b\ \varphi = nADDI^b\ \varphi$ **unfolding** *cond* **by** (*smt* (*z3*) *compl-def subset-def subset-out-def*)

Relativized nMULT variants.

definition *nMULTr*::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹*nMULTr*›)
where $nMULTr\ \varphi \equiv \forall A\ B.\ let\ U = (A \wedge B)\ in\ \ (\varphi(A \wedge B) =_U (\varphi\ A) \vee (\varphi\ B))$
definition *nMULTr-a*::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹$nMULTr^a$›)
where $nMULTr^a\ \varphi \equiv \forall A\ B.\ let\ U = (A \wedge B)\ in\ ((\varphi\ A) \vee (\varphi\ B) \leq_U \varphi(A \wedge B))$
definition *nMULTr-b*::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ (‹$nMULTr^b$›)

where $nMULTr^b\ \varphi \equiv \forall\, A\ B.\ let\ U = (A \wedge B)\ in \quad (\varphi(A \wedge B) \leq_U (\varphi\ A) \vee (\varphi\ B))$

declare *nMULTr-def*[*cond*] *nMULTr-a-def*[*cond*] *nMULTr-b-def*[*cond*]

lemma *nMULTr-char*: $nMULTr\ \varphi = (nMULTr^a\ \varphi \wedge nMULTr^b\ \varphi)$ **unfolding** *cond* **by** (*meson setequ-char setequ-in-char subset-in-char*)

lemma *nMULTr-a-impl*: $nMULT^a\ \varphi \longrightarrow nMULTr^a\ \varphi$ **by** (*simp add*: *nMULT-a-def nMULTr-a-def subset-def subset-in-def*)
lemma $CNTR\ \varphi \Longrightarrow nMULTr^a\ \varphi \longrightarrow nMULT^a\ \varphi$ **nitpick oops** — countermodel
lemma *nMULTr-a-equ*: $nCNTR\ \varphi \Longrightarrow nMULTr^a\ \varphi = nMULT^a\ \varphi$ **unfolding** *cond* **by** (*smt* (*z3*) *compl-def subset-def subset-in-def*)

lemma *nMULTr-b-impl*: $nMULT^b\ \varphi \longrightarrow nMULTr^b\ \varphi$ **by** (*simp add*: *nMULT-b-def nMULTr-b-def subset-def subset-in-def*)
lemma $nMULTr^b\ \varphi \longrightarrow nMULT^b\ \varphi$ **nitpick oops** — countermodel
lemma *nMULTr-b-equ*: $CNTR\ \varphi \Longrightarrow nMULTr^b\ \varphi = nMULT^b\ \varphi$ **unfolding** *cond* **by** (*smt* (*z3*) *compl-def join-def meet-def subset-def subset-in-def*)
lemma *nMULTr-b-equ'*:$nCNTR\ \varphi \Longrightarrow nMULTr^b\ \varphi = nMULT^b\ \varphi$ **unfolding** *cond* **by** (*smt* (*z3*) *compl-def join-def meet-def subset-def subset-in-def*)

Weak variants of antitonicity.

definition *ANTIw1*::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ ($\langle ANTIw^1 \rangle$)
where $ANTIw^1\ \varphi \equiv \forall\, A\ B.\ A \leq B \longrightarrow (\varphi\ B) \leq B \vee (\varphi\ A)$
definition *ANTIw2*::$('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ ($\langle ANTIw^2 \rangle$)
where $ANTIw^2\ \varphi \equiv \forall\, A\ B.\ A \leq B \longrightarrow A \wedge (\varphi\ B) \leq (\varphi\ A)$

declare *ANTIw1-def*[*cond*] *ANTIw2-def*[*cond*]

lemma *ANTIw1-nADDIr-b*: $ANTIw^1\ \varphi = nADDIr^b\ \varphi$ **proof** −
have *l2r*: $ANTIw^1\ \varphi \longrightarrow nADDIr^b\ \varphi$ **unfolding** *cond subset-out-char* **by** (*smt* (*verit*, *ccfv-SIG*) *BA-distr2 L8 join-def setequ-ext subset-def*)
have *r2l*: $nADDIr^b\ \varphi \longrightarrow ANTIw^1\ \varphi$ **unfolding** *cond subset-out-def* **by** (*metis* (*full-types*) *L9 join-def meet-def setequ-ext subset-def*)
show *?thesis* **using** *l2r r2l* **by** *blast*
qed
lemma *ANTIw2-nMULTr-a*: $ANTIw^2\ \varphi = nMULTr^a\ \varphi$ **proof** −
have *l2r*: $ANTIw^2\ \varphi \longrightarrow nMULTr^a\ \varphi$ **unfolding** *cond subset-in-char* **by** (*metis BA-distr1 L3 L4 L5 L7 L8 setequ-ext*)
have *r2l*: $nMULTr^a\ \varphi \longrightarrow ANTIw^2\ \varphi$ **unfolding** *cond subset-in-def* **by** (*metis* (*full-types*) *L10 join-def meet-def setequ-ext subset-def*)
show *?thesis* **using** *l2r r2l* **by** *blast*
qed

lemma $ANTI\ \varphi \longrightarrow ANTIw^1\ \varphi$ **by** (*simp add*: *ANTI-nADDIb ANTIw1-nADDIr-b nADDIr-b-impl*)
lemma $ANTIw^1\ \varphi \longrightarrow ANTI\ \varphi$ **nitpick oops** — countermodel
lemma $ANTI\ \varphi \longrightarrow ANTIw^2\ \varphi$ **by** (*simp add*: *ANTI-nMULTa ANTIw2-nMULTr-a nMULTr-a-impl*)
lemma $ANTIw^2\ \varphi \longrightarrow ANTI\ \varphi$ **nitpick oops** — countermodel

We have in fact that (n)CNTR (resp. (n)EXPN) implies ANTIw-1/nADDIr-b (resp. ANTIw-2/nMULTr-a).

lemma *CNTR-ANTIw1-impl*: $CNTR\ \varphi \longrightarrow ANTIw^1\ \varphi$ **unfolding** *cond* **using** *L3 subset-char1* **by** *blast*
lemma *nCNTR-ANTIw1-impl*: $nCNTR\ \varphi \longrightarrow ANTIw^1\ \varphi$ **unfolding** *cond* **by** (*metis* (*full-types*) *compl-def join-def subset-def*)

lemma *EXPN-ANTIw2-impl*: *EXPN* $\varphi \longrightarrow ANTIw^2\ \varphi$ **unfolding** *cond* **using** *L4 subset-char1* **by** *blast*
lemma *nEXPN-ANTIw2-impl*: *nEXPN* $\varphi \longrightarrow ANTIw^2\ \varphi$ **unfolding** *cond* **by** (*metis* (*full-types*) *compl-def meet-def subset-def*)

Dual interrelations.

lemma *ADDIr-dual1*: $ADDIr^a\ \varphi = MULTr^b\ \varphi^d$ **unfolding** *cond subset-in-char subset-out-char* **by** (*smt* (*z3*) *BA-cp BA-deMorgan1 BA-dn op-dual-def setequ-ext*)
lemma *ADDIr-dual2*: $ADDIr^b\ \varphi = MULTr^a\ \varphi^d$ **unfolding** *cond subset-in-char subset-out-char* **by** (*smt* (*verit*, *ccfv-threshold*) *BA-cp BA-deMorgan1 BA-dn op-dual-def setequ-ext*)
lemma *ADDIr-dual*: $ADDIr\ \varphi = MULTr\ \varphi^d$ **using** *ADDIr-char ADDIr-dual1 ADDIr-dual2 MULTr-char* **by** *blast*

lemma *nADDIr-dual1*: $nADDIr^a\ \varphi = nMULTr^b\ \varphi^d$ **unfolding** *cond subset-in-char subset-out-char* **by** (*smt* (*verit*, *del-insts*) *BA-cp BA-deMorgan1 BA-dn op-dual-def setequ-ext*)
lemma *nADDIr-dual2*: $nADDIr^b\ \varphi = nMULTr^a\ \varphi^d$ **by** (*smt* (*z3*) *BA-deMorgan1 BA-dn compl-def nADDIr-b-def nMULTr-a-def op-dual-def setequ-ext subset-in-def subset-out-def*)
lemma *nADDIr-dual*: $nADDIr\ \varphi = nMULTr\ \varphi^d$ **using** *nADDIr-char nADDIr-dual1 nADDIr-dual2 nMULTr-char* **by** *blast*

Complement interrelations.

lemma *ADDIr-a-cmpl*: $ADDIr^a\ \varphi = nADDIr^a\ \varphi^-$ **unfolding** *cond* **by** (*smt* (*verit*, *del-insts*) *BA-deMorgan1 compl-def setequ-ext subset-out-def svfun-compl-def*)
lemma *ADDIr-b-cmpl*: $ADDIr^b\ \varphi = nADDIr^b\ \varphi^-$ **unfolding** *cond* **by** (*smt* (*verit*, *del-insts*) *BA-deMorgan1 compl-def setequ-ext subset-out-def svfun-compl-def*)
lemma *ADDIr-cmpl*: $ADDIr\ \varphi = nADDIr\ \varphi^-$ **by** (*simp add*: *ADDIr-a-cmpl ADDIr-b-cmpl ADDIr-char nADDIr-char*)
lemma *MULTr-a-cmpl*: $MULTr^a\ \varphi = nMULTr^a\ \varphi^-$ **unfolding** *cond* **by** (*smt* (*verit*, *del-insts*) *BA-deMorgan2 compl-def setequ-ext subset-in-def svfun-compl-def*)
lemma *MULTr-b-cmpl*: $MULTr^b\ \varphi = nMULTr^b\ \varphi^-$ **unfolding** *cond* **by** (*smt* (*verit*, *ccfv-threshold*) *BA-deMorgan2 compl-def setequ-ext subset-in-def svfun-compl-def*)
lemma *MULTr-cmpl*: $MULTr\ \varphi = nMULTr\ \varphi^-$ **by** (*simp add*: *MULTr-a-cmpl MULTr-b-cmpl MULTr-char nMULTr-char*)

Fixed-point interrelations.

lemma *EXPN-fp*: $EXPN\ \varphi = EXPN\ \varphi^{fp}$ **by** (*simp add*: *EXPN-def dimpl-def op-fixpoint-def subset-def*)
lemma *EXPN-fpc*: $EXPN\ \varphi = nEXPN\ \varphi^{fp-}$ **using** *EXPN-fp nEXPN-CNTR-compl* **by** *blast*
lemma *CNTR-fp*: $CNTR\ \varphi = nCNTR\ \varphi^{fp}$ **by** (*metis EXPN-CNTR-dual1 EXPN-fp dual-compl-char2 dual-invol nCNTR-EXPN-compl ofp-comm-dc1 sfun-compl-invol*)
lemma *CNTR-fpc*: $CNTR\ \varphi = CNTR\ \varphi^{fp-}$ **by** (*metis CNTR-fp nCNTR-EXPN-compl ofp-comm-compl ofp-invol*)

lemma *nNORM-fp*: $NORM\ \varphi = nNORM\ \varphi^{fp}$ **by** (*metis NORM-def fixpoints-def fp-rel nNORM-def*)
lemma *NORM-fpc*: $NORM\ \varphi = NORM\ \varphi^{fp-}$ **by** (*simp add*: *NORM-def bottom-def ofp-fixpoint-compl-def sdiff-def*)
lemma *DNRM-fp*: $DNRM\ \varphi = DNRM\ \varphi^{fp}$ **by** (*simp add*: *DNRM-def dimpl-def op-fixpoint-def top-def*)
lemma *DNRM-fpc*: $DNRM\ \varphi = nDNRM\ \varphi^{fp-}$ **using** *DNRM-fp nDNRM-DNRM-compl* **by** *blast*

lemma *ADDIr-a-fpc*: $ADDIr^a\ \varphi = ADDIr^a\ \varphi^{fp-}$ **unfolding** *cond subset-out-def* **by** (*simp add*: *join-def ofp-fixpoint-compl-def sdiff-def*)
lemma *ADDIr-a-fp*: $ADDIr^a\ \varphi = nADDIr^a\ \varphi^{fp}$ **by** (*metis ADDIr-a-cmpl ADDIr-a-fpc sfun-compl-invol*)
lemma *ADDIr-b-fpc*: $ADDIr^b\ \varphi = ADDIr^b\ \varphi^{fp-}$ **unfolding** *cond subset-out-def* **by** (*simp add*: *join-def ofp-fixpoint-compl-def sdiff-def*)
lemma *ADDIr-b-fp*: $ADDIr^b\ \varphi = nADDIr^b\ \varphi^{fp}$ **by** (*metis ADDIr-b-cmpl ADDIr-b-fpc sfun-compl-invol*)

lemma *MULTr-a-fp*: $MULTr^a\ \varphi = MULTr^a\ \varphi^{fp}$ **unfolding** *cond subset-in-def* **by** (*simp add*: *dimpl-def meet-def op-fixpoint-def*)
lemma *MULTr-a-fpc*: $MULTr^a\ \varphi = nMULTr^a\ \varphi^{fp-}$ **using** *MULTr-a-cmpl MULTr-a-fp* **by** *blast*
lemma *MULTr-b-fp*: $MULTr^b\ \varphi = MULTr^b\ \varphi^{fp}$ **unfolding** *cond subset-in-def* **by** (*simp add*: *dimpl-def meet-def op-fixpoint-def*)
lemma *MULTr-b-fpc*: $MULTr^b\ \varphi = nMULTr^b\ \varphi^{fp-}$ **using** *MULTr-b-cmpl MULTr-b-fp* **by** *blast*

Relativized IDEM variants.

definition *IDEMr-a*::$('w\ \sigma \Rightarrow\ 'w\ \sigma) \Rightarrow bool$ ($\langle IDEMr^a\rangle$)
  where $IDEMr^a\ \varphi \equiv \forall A.\ \varphi(A \vee \varphi\ A) \leq^A (\varphi\ A)$
definition *IDEMr-b*::$('w\ \sigma \Rightarrow\ 'w\ \sigma) \Rightarrow bool$ ($\langle IDEMr^b\rangle$)
  where $IDEMr^b\ \varphi \equiv \forall A.\ (\varphi\ A) \leq_A \varphi(A \wedge \varphi\ A)$
declare *IDEMr-a-def*[*cond*] *IDEMr-b-def*[*cond*]

Relativized nIDEM variants.

definition *nIDEMr-a*::$('w\ \sigma \Rightarrow\ 'w\ \sigma) \Rightarrow bool$ ($\langle nIDEMr^a\rangle$)
  where $nIDEMr^a\ \varphi \equiv \forall A.\ (\varphi\ A) \leq^A \varphi(A \vee -(\varphi\ A))$
definition *nIDEMr-b*::$('w\ \sigma \Rightarrow\ 'w\ \sigma) \Rightarrow bool$ ($\langle nIDEMr^b\rangle$)
  where $nIDEMr^b\ \varphi \equiv \forall A.\ \varphi(A \wedge -(\varphi\ A)) \leq_A (\varphi\ A)$

declare *nIDEMr-a-def*[*cond*] *nIDEMr-b-def*[*cond*]

Complement interrelations.

lemma *IDEMr-a-cmpl*: $IDEMr^a\ \varphi = nIDEMr^a\ \varphi^-$ **unfolding** *cond subset-in-def subset-out-def* **by** (*metis compl-def sfun-compl-invol svfun-compl-def*)
lemma *IDEMr-b-cmpl*: $IDEMr^b\ \varphi = nIDEMr^b\ \varphi^-$ **unfolding** *cond subset-in-def subset-out-def* **by** (*metis compl-def sfun-compl-invol svfun-compl-def*)

Dual interrelation.

lemma *IDEMr-dual*: $IDEMr^a\ \varphi = IDEMr^b\ \varphi^d$ **unfolding** *cond subset-in-def subset-out-def op-dual-def* **by** (*metis* (*mono-tags*, *opaque-lifting*) *BA-dn compl-def diff-char1 diff-char2 impl-char setequ-ext*)
lemma *nIDEMr-dual*: $nIDEMr^a\ \varphi = nIDEMr^b\ \varphi^d$ **by** (*metis IDEMr-dual IDEMr-a-cmpl IDEMr-b-cmpl dual-compl-char1 dual-compl-char2 sfun-compl-invol*)

Fixed-point interrelations.

lemma *nIDEMr-a-fpc*: $nIDEMr^a\ \varphi = nIDEMr^a\ \varphi^{fp-}$ **proof** −
  have $\forall A.\ (\lambda p.\ A\ p \vee \neg\varphi\ A\ p) = (\lambda p.\ A\ p \vee \varphi\ A\ p = A\ p)$ **by** *blast*
  thus *?thesis* **unfolding** *cond subset-out-def ofp-fixpoint-compl-def conn order* **by** *simp*
qed
lemma *IDEMr-a-fp*: $IDEMr^a\ \varphi = nIDEMr^a\ \varphi^{fp}$ **by** (*metis IDEMr-a-cmpl nIDEMr-a-fpc ofp-invol*)
lemma *IDEMr-a-fpc*: $IDEMr^a\ \varphi = IDEMr^a\ \varphi^{fp-}$ **by** (*metis IDEMr-a-cmpl nIDEMr-a-fpc ofp-comm-compl*)
lemma *IDEMr-b-fp*: $IDEMr^b\ \varphi = IDEMr^b\ \varphi^{fp}$ **by** (*metis IDEMr-a-fpc IDEMr-dual dual-compl-char1 dual-invol ofp-comm-compl ofp-comm-dc2*)
lemma *IDEMr-b-fpc*: $IDEMr^b\ \varphi = nIDEMr^b\ \varphi^{fp-}$ **using** *IDEMr-b-fp IDEMr-b-cmpl* **by** *blast*

The original border condition B1' (by Zarycki) is equivalent to the conjuntion of nMULTr and CNTR.

abbreviation $B1'\ \varphi \equiv \forall A\ B.\ \varphi(A \wedge B) = (A \wedge \varphi\ B) \vee (\varphi\ A \wedge B)$

lemma $B1'\ \varphi = (nMULTr\ \varphi \wedge CNTR\ \varphi)$ **proof** −
  have *l2ra*: $B1'\ \varphi \longrightarrow nMULTr\ \varphi$ **unfolding** *cond* **by** (*smt* (*z3*) *join-def meet-def setequ-ext setequ-in-def*)
  have *l2rb*: $B1'\ \varphi \longrightarrow CNTR\ \varphi$ **unfolding** *cond* **by** (*metis L2 L4 L5 L7 L9 setequ-ext*)

have *r2l*: $(nMULTr\ \varphi \wedge CNTR\ \varphi) \longrightarrow B1'\ \varphi$ **unfolding** *cond* **by** (*smt* (*z3*) *L10 join-def meet-def setequ-def setequ-in-char*)
from *l2ra l2rb r2l* **show** *?thesis* **by** *blast*
qed

Modulo conditions nMULTr and CNTR the border condition B4 is equivalent to nIDEMr-b.

abbreviation $B4\ \varphi \equiv \forall A.\ \varphi(-\varphi(-A)) \leq A$

lemma $nMULTr\ \varphi \Longrightarrow CNTR\ \varphi \Longrightarrow B4\ \varphi = nIDEMr^b\ \varphi$ **proof** −
assume *a1*: $nMULTr\ \varphi$ **and** *a2*: $CNTR\ \varphi$
have *l2r*: $nMULTr^b\ \varphi \Longrightarrow B4\ \varphi \longrightarrow nIDEMr^b\ \varphi$ **unfolding** *cond subset-in-char subset-def* **by** (*metis BA-deMorgan1 BA-dn compl-def meet-def setequ-ext*)
have *r2l*: $nMULTr^a\ \varphi \Longrightarrow CNTR\ \varphi \Longrightarrow nIDEMr^b\ \varphi \longrightarrow B4\ \varphi$ **unfolding** *cond* **by** (*smt* (*verit*) *compl-def join-def meet-def subset-def subset-in-def*)
from *l2r r2l* **show** *?thesis* **using** *a1 a2 nMULTr-char* **by** *blast*
qed

end
theory *conditions-relativized-infinitary*
imports *conditions-relativized conditions-negative-infinitary*
begin

3.4 Infinitary Relativized Conditions

We define and interrelate infinitary variants for some previously introduced axiomatic conditions on operators.

definition $iADDIr::('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ ($\langle iADDIr \rangle$)
where $iADDIr\ \varphi \equiv \forall S.\ let\ U{=}\bigvee S\ in\ (\varphi(\bigvee S) =^U \bigvee[\![\varphi\ S]\!])$
definition $iADDIr\text{-}a::('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ ($\langle iADDIr^a \rangle$)
where $iADDIr^a\ \varphi \equiv \forall S.\ let\ U{=}\bigvee S\ in\ (\varphi(\bigvee S) \leq^U \bigvee[\![\varphi\ S]\!])$
definition $iADDIr\text{-}b::('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ ($\langle iADDIr^b \rangle$)
where $iADDIr^b\ \varphi \equiv \forall S.\ let\ U{=}\bigvee S\ in\ (\bigvee[\![\varphi\ S]\!] \leq^U \varphi(\bigvee S))$

definition $inADDIr::('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ ($\langle inADDIr \rangle$)
where $inADDIr\ \varphi \equiv \forall S.\ let\ U{=}\bigvee S\ in\ (\varphi(\bigvee S) =^U \bigwedge[\![\varphi\ S]\!])$
definition $inADDIr\text{-}a::('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ ($\langle inADDIr^a \rangle$)
where $inADDIr^a\ \varphi \equiv \forall S.\ let\ U{=}\bigvee S\ in\ (\bigwedge[\![\varphi\ S]\!] \leq^U \varphi(\bigvee S))$
definition $inADDIr\text{-}b::('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ ($\langle inADDIr^b \rangle$)
where $inADDIr^b\ \varphi \equiv \forall S.\ let\ U{=}\bigvee S\ in\ (\varphi(\bigvee S) \leq^U \bigwedge[\![\varphi\ S]\!])$

declare *iADDIr-def*[*cond*] *iADDIr-a-def*[*cond*] *iADDIr-b-def*[*cond*]
inADDIr-def[*cond*] *inADDIr-a-def*[*cond*] *inADDIr-b-def*[*cond*]

definition $iMULTr::('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ ($\langle iMULTr \rangle$)
where $iMULTr\ \varphi \equiv \forall S.\ let\ U{=}\bigwedge S\ in\ (\varphi(\bigwedge S) =_U \bigwedge[\![\varphi\ S]\!])$
definition $iMULTr\text{-}a::('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ ($\langle iMULTr^a \rangle$)
where $iMULTr^a\ \varphi \equiv \forall S.\ let\ U{=}\bigwedge S\ in\ (\varphi(\bigwedge S) \leq_U \bigwedge[\![\varphi\ S]\!])$
definition $iMULTr\text{-}b::('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ ($\langle iMULTr^b \rangle$)
where $iMULTr^b\ \varphi \equiv \forall S.\ let\ U{=}\bigwedge S\ in\ (\bigwedge[\![\varphi\ S]\!] \leq_U \varphi(\bigwedge S))$

definition $inMULTr::('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ ($\langle inMULTr \rangle$)
where $inMULTr\ \varphi \equiv \forall S.\ let\ U{=}\bigwedge S\ in\ (\varphi(\bigwedge S) =_U \bigvee[\![\varphi\ S]\!])$
definition $inMULTr\text{-}a::('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ ($\langle inMULTr^a \rangle$)
where $inMULTr^a\ \varphi \equiv \forall S.\ let\ U{=}\bigwedge S\ in\ (\bigvee[\![\varphi\ S]\!] \leq_U \varphi(\bigwedge S))$
definition $inMULTr\text{-}b::('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow bool$ ($\langle inMULTr^b \rangle$)

where *inMULTr*^{b} $\varphi \equiv \forall S.$ *let* $U{=}\bigwedge S$ *in* $(\varphi(\bigwedge S) \leq_U \bigvee[\![\varphi\ S]\!])$

declare *iMULTr-def*[*cond*] *iMULTr-a-def*[*cond*] *iMULTr-b-def*[*cond*] *inMULTr-def*[*cond*] *inMULTr-a-def*[*cond*] *inMULTr-b-def*[*cond*]

lemma *iADDIr-char*: *iADDIr* φ = (*iADDIr*^{a} $\varphi \wedge$ *iADDIr*^{b} φ) **unfolding** *cond setequ-char setequ-out-char subset-out-char* **by** (*meson setequ-char*)
lemma *iMULTr-char*: *iMULTr* φ = (*iMULTr*^{a} $\varphi \wedge$ *iMULTr*^{b} φ) **unfolding** *cond setequ-char setequ-in-char subset-in-char* **by** (*meson setequ-char*)

lemma *inADDIr-char*: *inADDIr* φ = (*inADDIr*^{a} $\varphi \wedge$ *inADDIr*^{b} φ) **unfolding** *cond setequ-char setequ-out-char subset-out-char* **by** (*meson setequ-char*)
lemma *inMULTr-char*: *inMULTr* φ = (*inMULTr*^{a} $\varphi \wedge$ *inMULTr*^{b} φ) **unfolding** *cond setequ-char setequ-in-char subset-in-char* **by** (*meson setequ-char*)

Dual interrelations.

lemma *iADDIr-dual1*: *iADDIr*^{a} φ = *iMULTr*^{b} φ^d **unfolding** *cond* **by** (*smt* (*z3*) *BA-cmpl-equ BA-cp BA-deMorgan2 dual-invol iDM-a iDM-b im-prop1 op-dual-def setequ-ext subset-in-char subset-out-char*)
lemma *iADDIr-dual2*: *iADDIr*^{b} φ = *iMULTr*^{a} φ^d **unfolding** *cond* **by** (*smt* (*z3*) *BA-cmpl-equ BA-cp BA-deMorgan2 dual-invol iDM-a iDM-b im-prop1 op-dual-def setequ-ext subset-in-char subset-out-char*)
lemma *iADDIr-dual*: *iADDIr* φ = *iMULTr* φ^d **using** *iADDIr-char iADDIr-dual1 iADDIr-dual2 iMULTr-char* **by** *blast*

lemma *inADDIr-dual1*: *inADDIr*^{a} φ = *inMULTr*^{b} φ^d **unfolding** *cond* **by** (*smt* (*z3*) *BA-cmpl-equ compl-def dual-invol iDM-a iDM-b im-prop3 op-dual-def setequ-ext subset-in-def subset-in-out*)
lemma *inADDIr-dual2*: *inADDIr*^{b} φ = *inMULTr*^{a} φ^d **unfolding** *cond* **by** (*smt* (*z3*) *BA-cmpl-equ compl-def dual-invol iDM-a iDM-b im-prop3 op-dual-def setequ-ext subset-in-def subset-in-out*)
lemma *inADDIr-dual*: *inADDIr* φ = *inMULTr* φ^d **using** *inADDIr-char inADDIr-dual1 inADDIr-dual2 inMULTr-char* **by** *blast*

Complement interrelations.

lemma *iADDIr-a-cmpl*: *iADDIr*^{a} φ = *inADDIr*^{a} φ^- **unfolding** *cond* **by** (*smt* (*z3*) *compl-def dual-compl-invol iDM-b im-prop2 setequ-ext subset-out-def svfun-compl-def*)
lemma *iADDIr-b-cmpl*: *iADDIr*^{b} φ = *inADDIr*^{b} φ^- **unfolding** *cond* **by** (*smt* (*z3*) *compl-def iDM-b im-prop2 setequ-ext sfun-compl-invol subset-out-def svfun-compl-def*)
lemma *iADDIr-cmpl*: *iADDIr* φ = *inADDIr* φ^- **by** (*simp add*: *iADDIr-a-cmpl iADDIr-b-cmpl iADDIr-char inADDIr-char*)

lemma *iMULTr-a-cmpl*: *iMULTr*^{a} φ = *inMULTr*^{a} φ^- **unfolding** *cond* **by** (*smt* (*z3*) *compl-def iDM-b im-prop2 setequ-ext subset-in-def svfun-compl-def*)
lemma *iMULTr-b-cmpl*: *iMULTr*^{b} φ = *inMULTr*^{b} φ^- **unfolding** *cond* **by** (*smt* (*z3*) *compl-def dual-compl-invol iDM-a im-prop2 setequ-ext subset-in-def svfun-compl-def*)
lemma *iMULTr-cmpl*: *MULTr* φ = *nMULTr* φ^- **by** (*simp add*: *MULTr-a-cmpl MULTr-b-cmpl MULTr-char nMULTr-char*)

Fixed-point interrelations.

lemma *iADDIr-a-fpc*: *iADDIr*^{a} φ = *iADDIr*^{a} φ^{fp-} **unfolding** *cond subset-out-def image-def ofp-fixpoint-compl-def supremum-def sdiff-def* **by** (*smt* (*verit*))
lemma *iADDIr-a-fp*: *iADDIr*^{a} φ = *inADDIr*^{a} φ^{fp} **by** (*metis iADDIr-a-cmpl iADDIr-a-fpc sfun-compl-invol*)
lemma *iADDIr-b-fpc*: *iADDIr*^{b} φ = *iADDIr*^{b} φ^{fp-} **unfolding** *cond subset-out-def image-def ofp-fixpoint-compl-def supremum-def sdiff-def* **by** (*smt* (*verit*))
lemma *iADDIr-b-fp*: *iADDIr*^{b} φ = *inADDIr*^{b} φ^{fp} **by** (*metis iADDIr-b-cmpl iADDIr-b-fpc sfun-compl-invol*)

lemma *iMULTr-a-fp*: *iMULTr*^{a} φ = *iMULTr*^{a} φ^{fp} **unfolding** *cond subset-in-def image-def* **by** (*smt* (*z3*) *dimpl-def infimum-def ofp-invol op-fixpoint-def*)

lemma *iMULTr-a-fpc*: $iMULTr^a\ \varphi = inMULTr^a\ \varphi^{fp-}$ **using** *iMULTr-a-cmpl iMULTr-a-fp* **by** *blast*
lemma *iMULTr-b-fp*: $iMULTr^b\ \varphi = iMULTr^b\ \varphi^{fp}$ **unfolding** *cond subset-in-def image-def dimpl-def infimum-def op-fixpoint-def* **by** (*smt* (*verit*))
lemma *iMULTr-b-fpc*: $iMULTr^b\ \varphi = inMULTr^b\ \varphi^{fp-}$ **using** *iMULTr-b-cmpl iMULTr-b-fp* **by** *blast*

end
theory *logics-consequence*
imports *boolean-algebra*
begin

4 Logics based on Topological Boolean Algebras

4.1 Logical Consequence and Validity

Logical validity can be defined as truth in all points (i.e. as denoting the top element).

abbreviation (*input*) *gtrue*::$'w\ \sigma \Rightarrow bool$ (‹[⊢ -]›)
where $[\vdash A] \equiv \forall w.\ A\ w$
lemma *gtrue-def*: $[\vdash A] \equiv A = \top$ **by** (*simp add*: *setequ-def top-def*)

When defining a logic over an existing algebra we have two choices: a global (truth preserving) and a local (degree preserving) notion of logical consequence. For LFIs/LFUs we prefer the latter.

abbreviation (*input*) *conseq-global1*::$'w\ \sigma \Rightarrow\ 'w\ \sigma \Rightarrow bool$ (‹[- $\vdash_g$ -]›)
where $[A \vdash_g B] \equiv [\vdash A] \longrightarrow [\vdash B]$
abbreviation (*input*) *conseq-global21*::$'w\ \sigma \Rightarrow\ 'w\ \sigma \Rightarrow\ 'w\ \sigma \Rightarrow bool$ (‹[-,- $\vdash_g$ -]›)
where $[A_1,\ A_2 \vdash_g B] \equiv [\vdash A_1] \wedge [\vdash A_2] \longrightarrow [\vdash B]$
abbreviation (*input*) *conseq-global31*::$'w\ \sigma \Rightarrow\ 'w\ \sigma \Rightarrow\ 'w\ \sigma \Rightarrow\ 'w\ \sigma \Rightarrow bool$ (‹[-,-,- $\vdash_g$ -]›)
where $[A_1,\ A_2,\ A_3 \vdash_g B] \equiv [\vdash A_1] \wedge [\vdash A_2] \wedge [\vdash A_3] \longrightarrow [\vdash B]$

abbreviation (*input*) *conseq-local1*::$'w\ \sigma \Rightarrow\ 'w\ \sigma \Rightarrow bool$ (‹[- ⊢ -]›)
where $[A \vdash B] \equiv A \leq B$
abbreviation (*input*) *conseq-local21*::$'w\ \sigma \Rightarrow\ 'w\ \sigma \Rightarrow\ 'w\ \sigma \Rightarrow bool$ (‹[-,- ⊢ -]›)
where $[A_1,\ A_2 \vdash B] \equiv A_1 \wedge A_2 \leq B$
abbreviation (*input*) *conseq-local12*::$'w\ \sigma \Rightarrow\ 'w\ \sigma \Rightarrow\ 'w\ \sigma \Rightarrow bool$ (‹[- ⊢ -,-]›)
where $[A \vdash B_1,\ B_2] \equiv A \leq B_1 \vee B_2$
abbreviation (*input*) *conseq-local31*::$'w\ \sigma \Rightarrow\ 'w\ \sigma \Rightarrow\ 'w\ \sigma \Rightarrow\ 'w\ \sigma \Rightarrow bool$ (‹[-,-,- ⊢ -]›)
where $[A_1,\ A_2,\ A_3 \vdash B] \equiv A_1 \wedge A_2 \wedge A_3 \leq B$

end
theory *logics-operators*
imports *conditions-positive*
begin

4.2 Converting between topological operators

We verify minimal conditions under which operators resulting from conversion functions coincide.

Conversions between interior, closure and exterior are straightforward and hold without restrictions: Interior and closure are each other duals. Exterior is the complement of closure. We focus here on conversions involving the border and frontier operators.

Interior operator as derived from border.

definition *Int-br*::$('w\ \sigma \Rightarrow 'w\ \sigma)\Rightarrow('w\ \sigma \Rightarrow 'w\ \sigma)$ $(\langle \mathcal{I}_B \rangle)$
where $\mathcal{I}_B\ \mathcal{B} \equiv \lambda A.\ A \leftharpoonup (\mathcal{B}\ A)$

Interior operator as derived from frontier.

definition *Int-fr*::$('w\ \sigma \Rightarrow 'w\ \sigma)\Rightarrow('w\ \sigma \Rightarrow 'w\ \sigma)$ $(\langle \mathcal{I}_F \rangle)$
where $\mathcal{I}_F\ \mathcal{F} \equiv \lambda A.\ A \leftharpoonup (\mathcal{F}\ A)$

Closure operator as derived from border.

definition *Cl-br*::$('w\ \sigma \Rightarrow 'w\ \sigma)\Rightarrow('w\ \sigma \Rightarrow 'w\ \sigma)$ $(\langle \mathcal{C}_B \rangle)$
where $\mathcal{C}_B\ \mathcal{B} \equiv \lambda A.\ A \vee \mathcal{B}(-A)$

Closure operator as derived from frontier.

definition *Cl-fr*::$('w\ \sigma \Rightarrow 'w\ \sigma)\Rightarrow('w\ \sigma \Rightarrow 'w\ \sigma)$ $(\langle \mathcal{C}_F \rangle)$
where $\mathcal{C}_F\ \mathcal{F} \equiv \lambda A.\ A \vee (\mathcal{F}\ A)$

Frontier operator as derived from interior.

definition *Fr-int*::$('w\ \sigma \Rightarrow 'w\ \sigma)\Rightarrow('w\ \sigma \Rightarrow 'w\ \sigma)$ $(\langle \mathcal{F}_I \rangle)$
where $\mathcal{F}_I\ \mathcal{I} \equiv \lambda A.\ -(\mathcal{I}\ A \vee \mathcal{I}(-A))$

Frontier operator as derived from closure.

definition *Fr-cl*::$('w\ \sigma \Rightarrow 'w\ \sigma)\Rightarrow('w\ \sigma \Rightarrow 'w\ \sigma)$ $(\langle \mathcal{F}_C \rangle)$
where $\mathcal{F}_C\ \mathcal{C} \equiv \lambda A.\ (\mathcal{C}\ A) \wedge \mathcal{C}(-A)$

Frontier operator as derived from border.

definition *Fr-br*::$('w\ \sigma \Rightarrow 'w\ \sigma)\Rightarrow('w\ \sigma \Rightarrow 'w\ \sigma)$ $(\langle \mathcal{F}_B \rangle)$
where $\mathcal{F}_B\ \mathcal{B} \equiv \lambda A.\ \mathcal{B}\ A \vee \mathcal{B}(-A)$

Border operator as derived from interior.

definition *Br-int*::$('w\ \sigma \Rightarrow 'w\ \sigma)\Rightarrow('w\ \sigma \Rightarrow 'w\ \sigma)$ $(\langle \mathcal{B}_I \rangle)$
where $\mathcal{B}_I\ \mathcal{I} \equiv \lambda A.\ A \leftharpoonup (\mathcal{I}\ A)$

Border operator as derived from closure.

definition *Br-cl*::$('w\ \sigma \Rightarrow 'w\ \sigma)\Rightarrow('w\ \sigma \Rightarrow 'w\ \sigma)$ $(\langle \mathcal{B}_C \rangle)$
where $\mathcal{B}_C\ \mathcal{C} \equiv \lambda A.\ A \wedge \mathcal{C}(-A)$

Border operator as derived from frontier.

definition *Br-fr*::$('w\ \sigma \Rightarrow 'w\ \sigma)\Rightarrow('w\ \sigma \Rightarrow 'w\ \sigma)$ $(\langle \mathcal{B}_F \rangle)$
where $\mathcal{B}_F\ \mathcal{F} \equiv \lambda A.\ A \wedge (\mathcal{F}\ A)$

Inter-definitions involving border or frontier do not hold without restrictions.

lemma $\mathcal{B} = \mathcal{B}_C\ (\mathcal{C}_B\ \mathcal{B})$ **nitpick oops** — countermodel
lemma $\mathcal{B} = \mathcal{B}_I\ (\mathcal{I}_B\ \mathcal{B})$ **nitpick oops** — countermodel
lemma $\mathcal{B} = \mathcal{B}_F\ (\mathcal{F}_B\ \mathcal{B})$ **nitpick oops** — countermodel
lemma $\mathcal{F} = \mathcal{F}_C\ (\mathcal{C}_F\ \mathcal{F})$ **nitpick oops** — countermodel
lemma $\mathcal{F} = \mathcal{F}_I\ (\mathcal{I}_F\ \mathcal{F})$ **nitpick oops** — countermodel
lemma $\mathcal{F} = \mathcal{F}_B\ (\mathcal{B}_F\ \mathcal{F})$ **nitpick oops** — countermodel

lemma $\mathcal{C} = \mathcal{C}_B\ (\mathcal{B}_C\ \mathcal{C})$ **nitpick oops** — countermodel
lemma $\mathcal{C} = \mathcal{C}_F\ (\mathcal{F}_C\ \mathcal{C})$ **nitpick oops** — countermodel
lemma $\mathcal{I} = \mathcal{I}_B\ (\mathcal{B}_C\ \mathcal{I})$ **nitpick oops** — countermodel
lemma $\mathcal{I} = \mathcal{I}_F\ (\mathcal{F}_C\ \mathcal{I})$ **nitpick oops** — countermodel

Inter-definitions involving border or frontier always assume the second Kuratowski condition (or its respective counterpart: C2, I2, B2 or F2).

abbreviation *C2* $\varphi \equiv$ *EXPN* φ
abbreviation *I2* $\varphi \equiv$ *CNTR* φ
abbreviation *B2* $\varphi \equiv$ *CNTR* φ
abbreviation *F2* $\varphi \equiv \forall A.\ \varphi(-A) = \varphi\ A$

lemma *B2* $\mathcal{B} \Longrightarrow \mathcal{B} = \mathcal{B}_C\ (\mathcal{C}_B\ \mathcal{B})$ **unfolding** *CNTR-def Br-cl-def Cl-br-def conn order* **by** *metis*
lemma *B2* $\mathcal{B} \Longrightarrow \mathcal{B} = \mathcal{B}_I\ (\mathcal{I}_B\ \mathcal{B})$ **unfolding** *CNTR-def Br-int-def Int-br-def conn order* **by** *metis*
lemma *B2* $\mathcal{B} \Longrightarrow \mathcal{B} = \mathcal{B}_F\ (\mathcal{F}_B\ \mathcal{B})$ **unfolding** *CNTR-def Br-fr-def Fr-br-def conn order* **by** *metis*
lemma *F2* $\mathcal{F} \Longrightarrow \mathcal{F} = \mathcal{F}_C\ (\mathcal{C}_F\ \mathcal{F})$ **unfolding** *Cl-fr-def Fr-cl-def conn order* **by** *metis*
lemma *F2* $\mathcal{F} \Longrightarrow \mathcal{F} = \mathcal{F}_I\ (\mathcal{I}_F\ \mathcal{F})$ **unfolding** *Int-fr-def Fr-int-def conn order* **by** *metis*
lemma *F2* $\mathcal{F} \Longrightarrow \mathcal{F} = \mathcal{F}_B\ (\mathcal{B}_F\ \mathcal{F})$ **unfolding** *Br-fr-def Fr-br-def conn order* **by** *metis*

lemma *C2* $\mathcal{C} \Longrightarrow \mathcal{C} = \mathcal{C}_B\ (\mathcal{B}_C\ \mathcal{C})$ **unfolding** *EXPN-def Br-cl-def Cl-br-def conn order* **by** *metis*
lemma *C2* $\mathcal{C} \Longrightarrow \mathcal{C} = \mathcal{C}_F\ (\mathcal{F}_C\ \mathcal{C})$ **unfolding** *EXPN-def Fr-cl-def Cl-fr-def conn order* **by** *metis*
lemma *I2* $\mathcal{I} \Longrightarrow \mathcal{I} = \mathcal{I}_B\ (\mathcal{B}_I\ \mathcal{I})$ **unfolding** *CNTR-def Int-br-def Br-int-def conn order* **by** *metis*
lemma *I2* $\mathcal{I} \Longrightarrow \mathcal{I} = \mathcal{I}_F\ (\mathcal{F}_I\ \mathcal{I})$ **unfolding** *CNTR-def Int-fr-def Fr-int-def conn order* **by** *metis*

end
theory *logics-negation*
  imports *logics-consequence conditions-relativized*
begin

4.3 Properties of negation(-like) operators

To avoid visual clutter we introduce convenient notation for type for properties of operators.

type-synonym $'w\ \Omega = ('w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow$ *bool*

named-theorems *neg*

4.3.1 Principles of excluded middle, contradiction and explosion

TND: tertium non datur, aka. law of excluded middle (resp. strong, weak, minimal).

abbreviation *pTND* (‹*TND*¯ -›) **where** $TND^a\ \eta \equiv \quad [\vdash a \vee \eta\ a]$
abbreviation *pTNDw* (‹*TNDw*¯ -›) **where** $TNDw^a\ \eta \equiv \forall b.\ [\eta\ b \vdash a,\ \eta\ a]$
abbreviation *pTNDm* (‹*TNDm*¯ -›) **where** $TNDm^a\ \eta \equiv \quad [\eta\ \bot \vdash a,\ \eta\ a]$
definition *TND* $::'w\ \Omega$ **where** $TND\ \ \eta \equiv \forall \varphi.\ TND^{\varphi}\ \ \eta$
definition *TNDw*$::'w\ \Omega$ **where** $TNDw\ \eta \equiv \forall \varphi.\ TNDw^{\varphi}\ \eta$
definition *TNDm*$::'w\ \Omega$ **where** $TNDm\ \eta \equiv \forall \varphi.\ TNDm^{\varphi}\ \eta$
declare *TND-def*[*neg*] *TNDw-def*[*neg*] *TNDm-def*[*neg*]

Explore some (non)entailment relations.

lemma *TND* $\eta \Longrightarrow$ *TNDw* η **unfolding** *neg conn order* **by** *simp*
lemma *TNDw* $\eta \Longrightarrow$ *TND* η **nitpick oops** — counterexample
lemma *TNDw* $\eta \Longrightarrow$ *TNDm* η **unfolding** *neg* **by** *simp*
lemma *TNDm* $\eta \Longrightarrow$ *TNDw* η **nitpick oops** — counterexample

ECQ: ex contradictione (sequitur) quodlibet (variants: strong, weak, minimal).

abbreviation *pECQ* (‹*ECQ*¯ -›) **where** $ECQ^a\ \ \eta \equiv \quad [a,\ \eta\ a \vdash \bot]$
abbreviation *pECQw* (‹*ECQw*¯ -›) **where** $ECQw^a\ \eta \equiv \forall b.\ [a,\ \eta\ a \vdash \eta\ b]$
abbreviation *pECQm* (‹*ECQm*¯ -›) **where** $ECQm^a\ \eta \equiv \quad [a,\ \eta\ a \vdash \eta\ \top]$
definition *ECQ* $::'w\ \Omega$ **where** $ECQ\ \eta \equiv \forall a.\ ECQ^a\ \ \eta$
definition *ECQw*$::'w\ \Omega$ **where** $ECQw\ \eta \equiv \forall a.\ ECQw^a\ \eta$
definition *ECQm*$::'w\ \Omega$ **where** $ECQm\ \eta \equiv \forall a.\ ECQm^a\ \eta$
declare *ECQ-def*[*neg*] *ECQw-def*[*neg*] *ECQm-def*[*neg*]

Explore some (non)entailment relations.

lemma $ECQ\ \ \eta \Longrightarrow ECQw\ \eta$ **unfolding** *neg conn order* **by** *blast*
lemma $ECQw\ \eta \Longrightarrow ECQ\ \ \eta$ **nitpick oops** — counterexample
lemma $ECQw\ \eta \Longrightarrow ECQm\ \eta$ **unfolding** *neg* **by** *simp*
lemma $ECQm\ \eta \Longrightarrow ECQw\ \eta$ **nitpick oops** — counterexample

LNC: law of non-contradiction.

abbreviation *pLNC* (‹LNC^{-} -›) **where** $LNC^{a}\ \eta \equiv [\vdash \eta(a \wedge \eta\ a)]$
definition $LNC::'w\ \Omega$ **where** $LNC\ \eta \equiv \forall\, a.\ LNC^{a}\ \eta$
declare *LNC-def*[*neg*]

ECQ and LNC are in general independent.

lemma $ECQ\ \eta \Longrightarrow LNC\ \eta$ **nitpick oops** — counterexample
lemma $LNC\ \eta \Longrightarrow ECQm\ \eta$ **nitpick oops** — counterexample

4.3.2 Contraposition rules

CoP: contraposition (weak 'rule-like' variants). Variant 0 is antitonicity (ANTI). Variants 1-3 are stronger.

abbreviation *pCoP1* (‹$CoP1^{--}$ -›) **where** $CoP1^{ab}\ \eta \equiv [a \vdash \eta\ b] \longrightarrow [b \vdash \eta\ a]$
abbreviation *pCoP2* (‹$CoP2^{--}$ -›) **where** $CoP2^{ab}\ \eta \equiv [\eta\ a \vdash b] \longrightarrow [\eta\ b \vdash a]$
abbreviation *pCoP3* (‹$CoP3^{--}$ -›) **where** $CoP3^{ab}\ \eta \equiv [\eta\ a \vdash \eta\ b] \longrightarrow [b \vdash a]$

abbreviation $CoP0\ ::'w\ \Omega$ **where** $CoP0\ \ \eta \equiv ANTI\ \eta$
definition $CoP1\ ::'w\ \Omega$ **where** $CoP1\ \ \eta \equiv \forall\, a\ b.\ CoP1^{ab}\ \eta$
definition $CoP2\ ::'w\ \Omega$ **where** $CoP2\ \ \eta \equiv \forall\, a\ b.\ CoP2^{ab}\ \eta$
definition $CoP3\ ::'w\ \Omega$ **where** $CoP3\ \ \eta \equiv \forall\, a\ b.\ CoP3^{ab}\ \eta$

declare *CoP1-def*[*neg*] *CoP2-def*[*neg*] *CoP3-def*[*neg*]

Explore some (non)entailment relations.

lemma $CoP1\ \eta \Longrightarrow CoP0\ \eta$ **unfolding** *ANTI-def CoP1-def* **using** *subset-char1* **by** *blast*
lemma $CoP0\ \eta \Longrightarrow CoP1\ \eta$ **nitpick oops** — counterexample
lemma $CoP2\ \eta \Longrightarrow CoP0\ \eta$ **unfolding** *ANTI-def CoP2-def* **using** *subset-char1* **by** *blast*
lemma $CoP0\ \eta \Longrightarrow CoP2\ \eta$ **nitpick oops** — counterexample
lemma $CoP3\ \eta \Longrightarrow CoP0\ \eta$ **oops**
lemma $CoP0\ \eta \Longrightarrow CoP3\ \eta$ **nitpick oops** — counterexample

All three strong variants are pairwise independent. However, CoP3 follows from CoP1 plus CoP2.

lemma *CoP123*: $CoP1\ \eta \Longrightarrow CoP2\ \eta \Longrightarrow CoP3\ \eta$ **unfolding** *neg order* **by** *smt*

Taking all CoP together still leaves room for a boldly paraconsistent resp. paracomplete logic.

lemma $CoP1\ \eta \Longrightarrow CoP2\ \eta \Longrightarrow ECQm\ \eta$ **nitpick oops** — counterexample
lemma $CoP1\ \eta \Longrightarrow CoP2\ \eta \Longrightarrow TNDm\ \eta$ **nitpick oops** — counterexample

4.3.3 Modus tollens rules

MT: modus (tollendo) tollens (weak 'rule-like' variants).

abbreviation *pMT0* (‹$MT0^{--}$ -›) **where** $MT0^{ab}\ \eta \equiv [a \vdash b] \wedge [\vdash \eta\ b] \longrightarrow [\vdash \eta\ a]$
abbreviation *pMT1* (‹$MT1^{--}$ -›) **where** $MT1^{ab}\ \eta \equiv [a \vdash \eta\ b] \wedge [\vdash b] \longrightarrow [\vdash \eta\ a]$
abbreviation *pMT2* (‹$MT2^{--}$ -›) **where** $MT2^{ab}\ \eta \equiv [\eta\ a \vdash b] \wedge [\vdash \eta\ b] \longrightarrow [\vdash a]$

abbreviation *pMT3* (‹*MT3*$^{--}$ -›) **where** $MT3^{ab}\ \eta \equiv [\eta\ a \vdash \eta\ b] \wedge [\vdash b] \longrightarrow [\vdash a]$
definition $MT0{::}'w\ \Omega$ **where** $MT0\ \eta \equiv \forall\, a\ b.\ MT0^{ab}\ \eta$
definition $MT1{::}'w\ \Omega$ **where** $MT1\ \eta \equiv \forall\, a\ b.\ MT1^{ab}\ \eta$
definition $MT2{::}'w\ \Omega$ **where** $MT2\ \eta \equiv \forall\, a\ b.\ MT2^{ab}\ \eta$
definition $MT3{::}'w\ \Omega$ **where** $MT3\ \eta \equiv \forall\, a\ b.\ MT3^{ab}\ \eta$

declare *MT0-def*[*neg*] *MT1-def*[*neg*] *MT2-def*[*neg*] *MT3-def*[*neg*]

Again, all MT variants are pairwise independent. We explore some (non)entailment relations.

lemma $CoP0\ \eta \Longrightarrow MT0\ \eta$ **unfolding** *neg order cond conn* **by** *blast*
lemma $CoP1\ \eta \Longrightarrow MT1\ \eta$ **unfolding** *neg order conn* **by** *blast*
lemma $CoP2\ \eta \Longrightarrow MT2\ \eta$ **unfolding** *neg order conn* **by** *blast*
lemma $CoP3\ \eta \Longrightarrow MT3\ \eta$ **unfolding** *neg order conn* **by** *blast*
lemma $MT0\ \eta \Longrightarrow MT1\ \eta \Longrightarrow MT2\ \eta \Longrightarrow MT3\ \eta \Longrightarrow CoP0\ \eta$ **nitpick oops** — counterexample
lemma $MT0\ \eta \Longrightarrow MT1\ \eta \Longrightarrow MT2\ \eta \Longrightarrow MT3\ \eta \Longrightarrow ECQm\ \eta$ **nitpick oops** — counterexample
lemma $MT0\ \eta \Longrightarrow MT1\ \eta \Longrightarrow MT2\ \eta \Longrightarrow MT3\ \eta \Longrightarrow TNDm\ \eta$ **nitpick oops** — counterexample
lemma *MT123*: $MT1\ \eta \Longrightarrow MT2\ \eta \Longrightarrow MT3\ \eta$ **unfolding** *neg order conn* **by** *metis*

4.3.4 Double negation introduction and elimination

DNI/DNE: double negation introduction/elimination (strong 'axiom-like' variants).

abbreviation *pDNI* (‹*DNI*$^{-}$ -›) **where** $DNI^{a}\ \eta \equiv [a \vdash \eta(\eta\ a)]$
abbreviation *pDNE* (‹*DNE*$^{-}$ -›) **where** $DNE^{a}\ \eta \equiv [\eta(\eta\ a) \vdash a]$
definition $DNI{::}'w\ \Omega$ **where** $DNI\ \eta \equiv \forall\, a.\ DNI^{a}\ \eta$
definition $DNE{::}'w\ \Omega$ **where** $DNE\ \eta \equiv \forall\, a.\ DNE^{a}\ \eta$
declare *DNI-def*[*neg*] *DNE-def*[*neg*]

CoP1 (resp. CoP2) can alternatively be defined as CoPw plus DNI (resp. DNE).

lemma $DNI\ \eta \Longrightarrow CoP1\ \eta$ **nitpick oops** — counterexample
lemma *CoP1-def2*: $CoP1\ \eta = (CoP0\ \eta \wedge DNI\ \eta)$ **unfolding** *neg cond* **using** *subset-char2* **by** *blast*
lemma $DNE\ \eta \Longrightarrow CoP2\ \eta$ **nitpick oops** — counterexample
lemma *CoP2-def2*: $CoP2\ \eta = (CoP0\ \eta \wedge DNE\ \eta)$ **unfolding** *neg cond* **using** *subset-char1* **by** *blast*

Explore some non-entailment relations:

lemma $DNI\ \eta \Longrightarrow DNE\ \eta \Longrightarrow CoP0\ \eta$ **nitpick oops** — counterexample
lemma $DNI\ \eta \Longrightarrow DNE\ \eta \Longrightarrow TNDm\ \eta$ **nitpick oops** — counterexample
lemma $DNI\ \eta \Longrightarrow DNE\ \eta \Longrightarrow ECQm\ \eta$ **nitpick oops** — counterexample
lemma $DNI\ \eta \Longrightarrow DNE\ \eta \Longrightarrow MT0\ \eta$ **nitpick oops** — counterexample
lemma $DNI\ \eta \Longrightarrow DNE\ \eta \Longrightarrow MT1\ \eta$ **nitpick oops** — counterexample
lemma $DNI\ \eta \Longrightarrow DNE\ \eta \Longrightarrow MT2\ \eta$ **nitpick oops** — counterexample
lemma $DNI\ \eta \Longrightarrow DNE\ \eta \Longrightarrow MT3\ \eta$ **nitpick oops** — counterexample

DNI/DNE: double negation introduction/elimination (weak 'rule-like' variants).

abbreviation *prDNI* (‹*rDNI*$^{-}$ -›) **where** $rDNI^{a}\ \eta \equiv [\vdash a] \longrightarrow [\vdash \eta(\eta\ a)]$
abbreviation *prDNE* (‹*rDNE*$^{-}$ -›) **where** $rDNE^{a}\ \eta \equiv [\vdash \eta(\eta\ a)] \longrightarrow [\vdash a]$
definition $rDNI{::}'w\ \Omega$ **where** $rDNI\ \eta \equiv \forall\, a.\ rDNI^{a}\ \eta$
definition $rDNE{::}'w\ \Omega$ **where** $rDNE\ \eta \equiv \forall\, a.\ rDNE^{a}\ \eta$
declare *rDNI-def*[*neg*] *rDNE-def*[*neg*]

The 'rule-like' variants for DNI/DNE are strictly weaker than the 'axiom-like' ones.

lemma $DNI\ \eta \Longrightarrow rDNI\ \eta$ **unfolding** *neg order conn* **by** *simp*
lemma $rDNI\ \eta \Longrightarrow DNI\ \eta$ **nitpick oops** — counterexample
lemma $DNE\ \eta \Longrightarrow rDNE\ \eta$ **unfolding** *neg order conn* **by** *blast*
lemma $rDNE\ \eta \Longrightarrow DNE\ \eta$ **nitpick oops** — counterexample

The 'rule-like' variants for DNI/DNE follow already from modus tollens.

lemma *MT1-rDNI*: *MT1* $\eta \Longrightarrow$ *rDNI* η **unfolding** *neg order* **by** *blast*
lemma *MT2-rDNE*: *MT2* $\eta \Longrightarrow$ *rDNE* η **unfolding** *neg order* **by** *blast*

4.3.5 (Anti)Normality and its dual

nNORM (resp. nDNRM) is entailed by CoP1 (resp. CoP2).

lemma *CoP1-NORM*: *CoP1* $\eta \Longrightarrow$ *nNORM* η **unfolding** *neg cond conn order* **by** *simp*
lemma *CoP2-DNRM*: *CoP2* $\eta \Longrightarrow$ *nDNRM* η **unfolding** *neg cond conn* **by** (*smt* (*verit*) *setequ-char subset-def*)
lemma *DNI* $\eta \Longrightarrow$ *nNORM* η **nitpick oops** — counterexample
lemma *DNE* $\eta \Longrightarrow$ *nDNRM* η **nitpick oops** — counterexample

nNORM and nDNRM together entail the rule variant of DNI (rDNI).

lemma *nDNRM-rDNI*: *nNORM* $\eta \Longrightarrow$ *nDNRM* $\eta \Longrightarrow$ *rDNI* η **unfolding** *neg cond* **by** (*simp add*: *gtrue-def setequ-ext*)
lemma *nNORM* $\eta \Longrightarrow$ *nDNRM* $\eta \Longrightarrow$ *rDNE* η **nitpick oops** — counterexample

4.3.6 De Morgan laws

De Morgan laws correspond to anti-additivity and anti-multiplicativity).

DM3 (resp. DM4) are entailed by CoP0/ANTI together with DNE (resp. DNI).

lemma *CoP0-DNE-nMULTb*: *CoP0* $\eta \Longrightarrow$ *DNE* $\eta \Longrightarrow$ *nMULT*^{b} η **unfolding** *neg cond* **by** (*metis ANTI-def ANTI-nADDIb L12 nADDI-b-def subset-char1*)
lemma *CoP0-DNI-nADDIa*: *CoP0* $\eta \Longrightarrow$ *DNI* $\eta \Longrightarrow$ *nADDI*^{a} η **unfolding** *neg cond* **by** (*metis ANTI-def ANTI-nMULTa L11 nMULT-a-def subset-char2*)

From this follows that DM3 (resp. DM4) is entailed by CoP2 (resp. CoP1).

lemma *CoP2-nMULTb*: *CoP2* $\eta \Longrightarrow$ *nMULT*^{b} η **by** (*simp add*: *CoP0-DNE-nMULTb CoP2-def2*)
lemma *CoP1-nADDIa*: *CoP1* $\eta \Longrightarrow$ *nADDI*^{a} η **by** (*simp add*: *CoP0-DNI-nADDIa CoP1-def2*)

Explore some non-entailment relations:

lemma *CoP0* $\eta \Longrightarrow$ *nADDI*^{a} $\eta \Longrightarrow$ *nMULT*^{b} $\eta \Longrightarrow$ *nNORM* $\eta \Longrightarrow$ *nDNRM* $\eta \Longrightarrow$ *DNI* η **nitpick oops** — counterexample
lemma *CoP0* $\eta \Longrightarrow$ *nADDI*^{a} $\eta \Longrightarrow$ *nMULT*^{b} $\eta \Longrightarrow$ *nNORM* $\eta \Longrightarrow$ *nDNRM* $\eta \Longrightarrow$ *DNE* η **nitpick oops** — counterexample
lemma *CoP0* $\eta \Longrightarrow$ *nADDI*^{a} $\eta \Longrightarrow$ *nMULT*^{b} $\eta \Longrightarrow$ *DNI* $\eta \Longrightarrow$ *DNE* $\eta \Longrightarrow$ *ECQm* η **nitpick oops** — counterexample
lemma *CoP0* $\eta \Longrightarrow$ *nADDI*^{a} $\eta \Longrightarrow$ *nMULT*^{b} $\eta \Longrightarrow$ *DNI* $\eta \Longrightarrow$ *DNE* $\eta \Longrightarrow$ *TNDm* η **nitpick oops** — counterexample

4.3.7 Strong contraposition (axiom-like)

Observe that the definitions below take implication as an additional parameter: ι.

lCoP: (local) contraposition (strong 'axiom-like' variants, using local consequence).

abbreviation *plCoP0* (‹*lCoP0*$^{--}$ - -›) **where** *lCoP0*^{ab} $\iota\ \eta \equiv [\iota\ a\ b \vdash \iota\ (\eta\ b)\ (\eta\ a)]$
abbreviation *plCoP1* (‹*lCoP1*$^{--}$ - -›) **where** *lCoP1*^{ab} $\iota\ \eta \equiv [\iota\ a\ (\eta\ b) \vdash \iota\ b\ (\eta\ a)]$
abbreviation *plCoP2* (‹*lCoP2*$^{--}$ - -›) **where** *lCoP2*^{ab} $\iota\ \eta \equiv [\iota\ (\eta\ a)\ b \vdash \iota\ (\eta\ b)\ a]$
abbreviation *plCoP3* (‹*lCoP3*$^{--}$ - -›) **where** *lCoP3*^{ab} $\iota\ \eta \equiv [\iota\ (\eta\ a)\ (\eta\ b) \vdash \iota\ b\ a]$
definition *lCoP0*::$('w\ \sigma \Rightarrow 'w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow\ 'w\ \Omega$ **where** *lCoP0* $\iota\ \eta \equiv \forall a\ b.$ *lCoP0*^{ab} $\iota\ \eta$
definition *lCoP1*::$('w\ \sigma \Rightarrow 'w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow\ 'w\ \Omega$ **where** *lCoP1* $\iota\ \eta \equiv \forall a\ b.$ *lCoP1*^{ab} $\iota\ \eta$

definition *lCoP2*::$('w\ \sigma \Rightarrow 'w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow 'w\ \Omega$ **where** *lCoP2* $\iota\ \eta \equiv \forall\, a\ b.\ lCoP2^{ab}\ \iota\ \eta$
definition *lCoP3*::$('w\ \sigma \Rightarrow 'w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow 'w\ \Omega$ **where** *lCoP3* $\iota\ \eta \equiv \forall\, a\ b.\ lCoP3^{ab}\ \iota\ \eta$

declare *lCoP0-def*[*neg*] *lCoP1-def*[*neg*] *lCoP2-def*[*neg*] *lCoP3-def*[*neg*]

All these contraposition variants are in general independent from each other. However if we employ classical implication we can verify some relationships.

lemma *lCoP1-def2*: $lCoP1(\rightarrow)\ \eta = (lCoP0(\rightarrow)\ \eta \wedge DNI\ \eta)$ **unfolding** *neg conn order* **by** *metis*
lemma *lCoP2-def2*: $lCoP2(\rightarrow)\ \eta = (lCoP0(\rightarrow)\ \eta \wedge DNE\ \eta)$ **unfolding** *neg conn order* **by** *blast*
lemma $lCoP1(\rightarrow)\ \eta \Longrightarrow lCoP0(\rightarrow)\ \eta$ **unfolding** *neg conn order* **by** *blast*
lemma $lCoP0(\rightarrow)\ \eta \Longrightarrow lCoP1(\rightarrow)\ \eta$ **nitpick oops** — counterexample
lemma $lCoP2(\rightarrow)\ \eta \Longrightarrow lCoP0(\rightarrow)\ \eta$ **unfolding** *neg conn order* **by** *blast*
lemma $lCoP0(\rightarrow)\ \eta \Longrightarrow lCoP2(\rightarrow)\ \eta$ **nitpick oops** — counterexample
lemma $lCoP3(\rightarrow)\ \eta \Longrightarrow lCoP0(\rightarrow)\ \eta$ **unfolding** *neg conn order* **by** *blast*
lemma $lCoP0(\rightarrow)\ \eta \Longrightarrow lCoP3(\rightarrow)\ \eta$ **nitpick oops** — counterexample
lemma *lCoP123*: $lCoP1(\rightarrow)\ \eta \wedge lCoP2(\rightarrow)\ \eta \Longrightarrow lCoP3(\rightarrow)\ \eta$ **unfolding** *neg conn order* **by** *metis*

Strong/axiom-like variants imply weak/rule-like ones as expected.

lemma $lCoP0(\rightarrow)\ \eta \Longrightarrow CoP0\ \eta$ **unfolding** *neg cond conn order* **by** *blast*
lemma $lCoP1(\rightarrow)\ \eta \Longrightarrow CoP1\ \eta$ **unfolding** *neg conn order* **by** *blast*
lemma $lCoP2(\rightarrow)\ \eta \Longrightarrow CoP2\ \eta$ **unfolding** *neg conn order* **by** *blast*
lemma $lCoP3(\rightarrow)\ \eta \Longrightarrow CoP3\ \eta$ **unfolding** *neg conn order* **by** *blast*

Explore some (non)entailment relations.

lemma *lCoP1-TND*: $lCoP1(\rightarrow)\ \eta \Longrightarrow TND\ \eta$ **unfolding** *neg conn* **by** (*smt* (*verit*, *best*) *setequ-char subset-def*)
lemma $TND\ \eta \Longrightarrow lCoP1(\rightarrow)\ \eta$ **nitpick oops** — counterexample
lemma *lCoP2-ECQ*: $lCoP2(\rightarrow)\ \eta \Longrightarrow ECQ\ \eta$ **unfolding** *neg conn* **by** (*smt* (*verit*) *setequ-def subset-def*)
lemma $ECQ\ \eta \Longrightarrow lCoP2(\rightarrow)\ \eta$ **nitpick oops** — counterexample

4.3.8 Local modus tollens axioms

lMT: (local) Modus tollens (strong, 'axiom-like' variants, using local consequence).

abbreviation *plMT0* (‹$lMT0^{--}$ - -›) **where** $lMT0^{ab}\ \iota\ \eta \equiv [\iota\ a\ b,\ \eta\ b \vdash \eta\ a]$
abbreviation *plMT1* (‹$lMT1^{--}$ - -›) **where** $lMT1^{ab}\ \iota\ \eta \equiv [\iota\ a\ (\eta\ b),\ b \vdash \eta\ a]$
abbreviation *plMT2* (‹$lMT2^{--}$ - -›) **where** $lMT2^{ab}\ \iota\ \eta \equiv [\iota\ (\eta\ a)\ b,\ \eta\ b \vdash a]$
abbreviation *plMT3* (‹$lMT3^{--}$ - -›) **where** $lMT3^{ab}\ \iota\ \eta \equiv [\iota\ (\eta\ a)\ (\eta\ b),\ b \vdash a]$
definition *lMT0*::$('w\ \sigma \Rightarrow 'w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow 'w\ \Omega$ **where** *lMT0* $\iota\ \eta \equiv \forall\, a\ b.\ lMT0^{ab}\ \iota\ \eta$
definition *lMT1*::$('w\ \sigma \Rightarrow 'w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow 'w\ \Omega$ **where** *lMT1* $\iota\ \eta \equiv \forall\, a\ b.\ lMT1^{ab}\ \iota\ \eta$
definition *lMT2*::$('w\ \sigma \Rightarrow 'w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow 'w\ \Omega$ **where** *lMT2* $\iota\ \eta \equiv \forall\, a\ b.\ lMT2^{ab}\ \iota\ \eta$
definition *lMT3*::$('w\ \sigma \Rightarrow 'w\ \sigma \Rightarrow 'w\ \sigma) \Rightarrow 'w\ \Omega$ **where** *lMT3* $\iota\ \eta \equiv \forall\, a\ b.\ lMT3^{ab}\ \iota\ \eta$

declare *lMT0-def*[*neg*] *lMT1-def*[*neg*] *lMT2-def*[*neg*] *lMT3-def*[*neg*]

All these MT variants are in general independent from each other and also from (strong) CoP instances. However if we take classical implication we can verify that local MT and CoP are indeed equivalent.

lemma $lMT0(\rightarrow)\ \eta = lCoP0(\rightarrow)\ \eta$ **unfolding** *neg conn order* **by** *blast*
lemma $lMT1(\rightarrow)\ \eta = lCoP1(\rightarrow)\ \eta$ **unfolding** *neg conn order* **by** *blast*
lemma $lMT2(\rightarrow)\ \eta = lCoP2(\rightarrow)\ \eta$ **unfolding** *neg conn order* **by** *blast*
lemma $lMT3(\rightarrow)\ \eta = lCoP3(\rightarrow)\ \eta$ **unfolding** *neg conn order* **by** *blast*

4.3.9 Disjunctive syllogism

DS: disjunctive syllogism.

abbreviation *pDS1* (‹*DS1*$^{--}$ - -›) **where** $DS1^{a\,b}\ \iota\ \eta \equiv [a \vee b \vdash \iota\ (\eta\ a)\ b]$
abbreviation *pDS2* (‹*DS2*$^{--}$ - -›) **where** $DS2^{a\,b}\ \iota\ \eta \equiv [\iota\ (\eta\ a)\ b \vdash a \vee b]$
abbreviation *pDS3* (‹*DS3*$^{--}$ - -›) **where** $DS3^{a\,b}\ \iota\ \eta \equiv [\eta\ a \vee b \vdash \iota\ a\ b]$
abbreviation *pDS4* (‹*DS4*$^{--}$ - -›) **where** $DS4^{a\,b}\ \iota\ \eta \equiv [\iota\ a\ b \vdash \eta\ a \vee b]$
definition $DS1{::}('w\ \sigma{\Rightarrow}'w\ \sigma{\Rightarrow}'w\ \sigma) \Rightarrow\ 'w\ \Omega$ **where** $DS1\ \iota\ \eta \equiv \forall\, a\ b.\ DS1^{a\,b}\ \iota\ \eta$
definition $DS2{::}('w\ \sigma{\Rightarrow}'w\ \sigma{\Rightarrow}'w\ \sigma) \Rightarrow\ 'w\ \Omega$ **where** $DS2\ \iota\ \eta \equiv \forall\, a\ b.\ DS2^{a\,b}\ \iota\ \eta$
definition $DS3{::}('w\ \sigma{\Rightarrow}'w\ \sigma{\Rightarrow}'w\ \sigma) \Rightarrow\ 'w\ \Omega$ **where** $DS3\ \iota\ \eta \equiv \forall\, a\ b.\ DS3^{a\,b}\ \iota\ \eta$
definition $DS4{::}('w\ \sigma{\Rightarrow}'w\ \sigma{\Rightarrow}'w\ \sigma) \Rightarrow\ 'w\ \Omega$ **where** $DS4\ \iota\ \eta \equiv \forall\, a\ b.\ DS4^{a\,b}\ \iota\ \eta$

declare *DS1-def*[*neg*] *DS2-def*[*neg*] *DS3-def*[*neg*] *DS4-def*[*neg*]

All DS variants are in general independent from each other. However if we take classical implication we can verify that the pairs DS1-DS3 and DS2-DS4 are indeed equivalent.

lemma $DS1(\rightarrow)\ \eta = DS3(\rightarrow)\ \eta$ **unfolding** *neg conn order* **by** *blast*
lemma $DS2(\rightarrow)\ \eta = DS4(\rightarrow)\ \eta$ **unfolding** *neg conn order* **by** *blast*

Explore some (non)entailment relations.

lemma *DS1-nDNor*: $DS1(\rightarrow)\ \eta \Longrightarrow nDNRM\ \eta$ **unfolding** *neg cond conn order* **by** *metis*
lemma *DS2-nNor*: $DS2(\rightarrow)\ \eta \Longrightarrow nNORM\ \eta$ **unfolding** *neg cond conn order* **by** *metis*
lemma *lCoP2-DS1*: $lCoP2(\rightarrow)\ \eta \Longrightarrow DS1(\rightarrow)\ \eta$ **unfolding** *neg conn order* **by** *blast*
lemma *lCoP1-DS2*: $lCoP1(\rightarrow)\ \eta \Longrightarrow DS2(\rightarrow)\ \eta$ **unfolding** *neg conn order* **by** *blast*
lemma $CoP2\ \eta \Longrightarrow DS1(\rightarrow)\ \eta$ **nitpick oops** — counterexample
lemma $CoP1\ \eta \Longrightarrow DS2(\rightarrow)\ \eta$ **nitpick oops** — counterexample

end
theory *logics-quantifiers*
  imports *boolean-algebra-infinitary*
begin

4.4 Quantifiers (restricted and unrestricted)

Introduce pedagogically convenient notation.

notation *HOL.All* (‹Π›) **notation** *HOL.Ex* (‹Σ›)

Let us recall that in HOL we have:

lemma $(\forall\, x.\ P) = \Pi(\lambda x.\ P)$ **by** *simp*
lemma $(\exists\, x.\ P) = \Sigma(\lambda x.\ P)$ **by** *simp*
lemma $\Sigma = (\lambda P.\ \neg\Pi(\lambda x.\ \neg P\ x))$ **by** *simp*

We can introduce their respective 'w-type-lifted variants as follows:

definition $mforall{::}('i{\Rightarrow}'w\ \sigma){\Rightarrow}'w\ \sigma$ (‹**Π**-›)
  where $\mathbf{\Pi}\varphi \equiv \lambda w.\ \forall\, X.\ \varphi\ X\ w$
definition $mexists{::}('i{\Rightarrow}'w\ \sigma){\Rightarrow}'w\ \sigma$ (‹**Σ**-›)
  where $\mathbf{\Sigma}\varphi \equiv \lambda w.\ \exists\, X.\ \varphi\ X\ w$

To improve readability, we introduce for them standard binder notation.

notation *mforall* (**binder** ‹∀› [*48*]*49*) **notation** *mexists* (**binder** ‹∃› [*48*]*49*)

And thus we obtain the 'w-type-lifted variant of the standard (variable-binding) quantifiers.

lemma $(\forall\, X.\ \varphi) = \mathbf{\Pi}(\lambda X.\ \varphi)$ **by** (*simp add*: *mforall-def*)

lemma $(\exists X.\ \varphi) = \mathbf{\Sigma}(\lambda X.\ \varphi)$ **by** (*simp add*: *mexists-def*)

Quantifiers are dual to each other in the expected way.

lemma $\mathbf{\Pi}\varphi = -(\mathbf{\Sigma}\varphi^{-})$ **by** (*simp add*: *compl-def mexists-def mforall-def svfun-compl-def*)
lemma $(\forall X.\ \varphi\ X) = -(\exists X.\ -(\varphi\ X))$ **by** (*simp add*: *compl-def mexists-def mforall-def*)

Relationship between quantifiers and the infinitary supremum and infimum operations.

lemma *mforall-char*: $\mathbf{\Pi}\varphi = \bigwedge[\![\varphi\ \text{-}]\!]$ **unfolding** *infimum-def mforall-def range-def* **by** *metis*
lemma *mexists-char*: $\mathbf{\Sigma}\varphi = \bigvee[\![\varphi\ \text{-}]\!]$ **unfolding** *supremum-def mexists-def range-def* **by** *metis*

lemma *mforallb-char*: $(\forall X.\ \varphi) = \bigwedge[\![(\lambda X.\ \varphi)\ \text{-}]\!]$ **unfolding** *infimum-def mforall-def range-def* **by** *simp*
lemma *mexistsb-char*: $(\exists X.\ \varphi) = \bigvee[\![(\lambda X.\ \varphi)\ \text{-}]\!]$ **unfolding** *supremum-def mexists-def range-def* **by** *simp*

We now consider quantification restricted over constant and varying domains.

Constant domains: first generalization of quantifiers above (e.g. free logic).

definition *mforall-const*::$'i\ \sigma \Rightarrow ('i \Rightarrow 'w\ \sigma) \Rightarrow 'w\ \sigma$ (‹$\mathbf{\Pi}$[-]-›)
  where $\mathbf{\Pi}[D]\varphi \equiv \lambda w.\ \forall X.\ (D\ X) \longrightarrow (\varphi\ X)\ w$
definition *mexists-const*::$'i\ \sigma \Rightarrow ('i \Rightarrow 'w\ \sigma) \Rightarrow 'w\ \sigma$ (‹$\mathbf{\Sigma}$[-]-›)
  where $\mathbf{\Sigma}[D]\varphi \equiv \lambda w.\ \exists X.\ (D\ X)\ \wedge\ (\varphi\ X)\ w$

Constant-domain quantification generalises its unrestricted counterpart.

lemma $\mathbf{\Pi}\varphi = \mathbf{\Pi}[\top]\varphi$ **by** (*simp add*: *mforall-const-def mforall-def top-def*)
lemma $\mathbf{\Sigma}\varphi = \mathbf{\Sigma}[\top]\varphi$ **by** (*simp add*: *mexists-const-def mexists-def top-def*)

Constant-domain quantification can also be characterised using infimum and supremum.

lemma *mforall-const-char*: $\mathbf{\Pi}[D]\varphi = \bigwedge[\![\varphi\ D]\!]$ **unfolding** *image-def infimum-def mforall-const-def* **by** *metis*
lemma *mexists-const-char*: $\mathbf{\Sigma}[D]\varphi = \bigvee[\![\varphi\ D]\!]$ **unfolding** *image-def supremum-def mexists-const-def* **by** *metis*

Constant-domain quantifiers also allow us to nicely characterize the interaction between function composition and (restricted) quantification:

lemma *mforall-comp*: $\mathbf{\Pi}(\varphi\circ\psi) = \mathbf{\Pi}[[\![\psi\ \text{-}]\!]]\ \varphi$ **unfolding** *fun-comp-def mforall-const-def mforall-def range-def* **by** *metis*
lemma *mexists-comp*: $\mathbf{\Sigma}(\varphi\circ\psi) = \mathbf{\Sigma}[[\![\psi\ \text{-}]\!]]\ \varphi$ **unfolding** *fun-comp-def mexists-const-def mexists-def range-def* **by** *metis*

Varying domains: we can also restrict quantifiers by taking a 'functional domain' as additional parameter. The latter is a set-valued mapping each element 'i to a set of points (e.g. where it 'exists').

definition *mforall-var*::$('i \Rightarrow 'w\ \sigma) \Rightarrow ('i \Rightarrow 'w\ \sigma) \Rightarrow 'w\ \sigma$ (‹$\mathbf{\Pi}$\{-\}-›)
  where $\mathbf{\Pi}\{\psi\}\varphi \equiv \lambda w.\ \forall X.\ (\psi\ X)\ w \longrightarrow (\varphi\ X)\ w$
definition *mexists-var*::$('i \Rightarrow 'w\ \sigma) \Rightarrow ('i \Rightarrow 'w\ \sigma) \Rightarrow 'w\ \sigma$ (‹$\mathbf{\Sigma}$\{-\}-›)
  where $\mathbf{\Sigma}\{\psi\}\varphi \equiv \lambda w.\ \exists X.\ (\psi\ X)\ w\ \wedge\ (\varphi\ X)\ w$

Varying-domain quantification generalizes its constant-domain counterpart.

lemma $\mathbf{\Pi}[D]\varphi = \mathbf{\Pi}\{D\upharpoonleft\}\varphi$ **by** (*simp add*: *mforall-const-def mforall-var-def*)
lemma $\mathbf{\Sigma}[D]\varphi = \mathbf{\Sigma}\{D\upharpoonleft\}\varphi$ **by** (*simp add*: *mexists-const-def mexists-var-def*)

Restricted quantifiers are dual to each other in the expected way.

lemma $\mathbf{\Pi}[D]\varphi = -(\mathbf{\Sigma}[D]\varphi^{-})$ **by** (*metis iDM-b im-prop2 mexists-const-char mforall-const-char setequ-ext*)

lemma $\mathbf{\Pi}\{\psi\}\varphi = -(\mathbf{\Sigma}\{\psi\}\varphi^-)$ **by** (*simp add*: *compl-def mexists-var-def mforall-var-def svfun-compl-def*)

We can use 2nd-order connectives on set-valued functions to encode restricted quantifiers as unrestricted.

lemma $\mathbf{\Pi}\{\psi\}\varphi = \mathbf{\Pi}(\psi \rightarrow^{:} \varphi)$ **by** (*simp add*: *impl-def mforall-def mforall-var-def svfun-impl-def*)
lemma $\mathbf{\Sigma}\{\psi\}\varphi = \mathbf{\Sigma}(\psi \wedge^{:} \varphi)$ **by** (*simp add*: *meet-def mexists-def mexists-var-def svfun-meet-def*)

Observe that using these operators has the advantage of allowing for binder notation.

lemma $\mathbf{\Pi}\{\psi\}\varphi = (\forall X.\ (\psi \rightarrow^{:} \varphi)\ X)$ **by** (*simp add*: *impl-def mforall-def mforall-var-def svfun-impl-def*)
lemma $\mathbf{\Sigma}\{\psi\}\varphi = (\exists X.\ (\psi \wedge^{:} \varphi)\ X)$ **by** (*simp add*: *meet-def mexists-def mexists-var-def svfun-meet-def*)

To sumarize: different sorts of restricted quantification can be emulated by employing 2nd-order operations to adequately relativize predicates.

lemma $\mathbf{\Pi}[D]\varphi = (\forall X.\ (D\upharpoonleft \rightarrow^{:} \varphi)\ X)$ **by** (*simp add*: *impl-def mforall-const-def mforall-def svfun-impl-def*)
lemma $\mathbf{\Pi}\{\top^{:}\}\varphi = (\forall X.\ (\top^{:} \rightarrow^{:} \varphi)\ X)$ **by** (*simp add*: *impl-def mforall-def mforall-var-def svfun-impl-def*)
lemma $\mathbf{\Pi}\varphi = \mathbf{\Pi}\{\top^{:}\}\varphi$ **by** (*simp add*: *mforall-def mforall-var-def svfun-top-def top-def*)
lemma $(\forall X.\ \varphi\ X) = \mathbf{\Pi}\{\top^{:}\}\varphi$ **by** (*simp add*: *mforall-def mforall-var-def svfun-top-def top-def*)

named-theorems *quant*
declare *mforall-def*[*quant*] *mexists-def*[*quant*]
mforall-const-def[*quant*] *mexists-const-def*[*quant*]
mforall-var-def[*quant*] *mexists-var-def*[*quant*]

end
theory *logics-quantifiers-example*
imports *logics-quantifiers conditions-positive-infinitary*
begin

4.5 Examples on Quantifiers

First-order quantification example.

lemma $(\forall x.\ A\ x \longrightarrow (\exists y.\ B\ x\ y)) \longleftrightarrow (\forall x.\ \exists y.\ A\ x \longrightarrow B\ x\ y)$ **by** *simp*
lemma $(\forall x.\ A\ x\ \rightarrow (\exists y.\ B\ x\ y))\ = (\forall x.\ \exists y.\ A\ x\ \rightarrow\ B\ x\ y)$ **by** (*simp add*: *impl-def mexists-def setequ-def*)

Propositional quantification example.

lemma $\forall A.\ (\exists B.\ (A \longleftrightarrow \neg B))$ **by** *blast*
lemma $(\forall A.\ (\exists B.\ A\ \leftrightarrow -B)) = \top$ **unfolding** *mforall-def mexists-def* **by** (*smt* (*verit*) *compl-def dimpl-def setequ-def top-def*)

Drinker's principle.

lemma $\exists x.\ Drunk\ x \rightarrow (\forall y.\ Drunk\ y) = \top$
by (*simp add*: *impl-def mexists-def mforall-def setequ-def top-def*)

Example in non-classical logics.

typedecl w
type-synonym $\sigma = (w\ \sigma)$

consts $\mathcal{C}{::}\sigma \Rightarrow \sigma$
abbreviation $\mathcal{I} \equiv \mathcal{C}^d$
abbreviation $CLOSURE\ \varphi \equiv ADDI\ \varphi \wedge EXPN\ \varphi \wedge NORM\ \varphi \wedge IDEM\ \varphi$
abbreviation $INTERIOR\ \varphi \equiv MULT\ \varphi \wedge CNTR\ \varphi \wedge DNRM\ \varphi \wedge IDEM\ \varphi$

definition *mforallInt*::$(\sigma \Rightarrow \sigma) \Rightarrow \sigma$ (‹$\mathbf{\Pi}^I$-›)
where $\mathbf{\Pi}^I\varphi \equiv \mathbf{\Pi}[fp\ \mathcal{I}]\varphi$
definition *mexistsInt*::$(\sigma \Rightarrow \sigma) \Rightarrow \sigma$ (‹$\mathbf{\Sigma}^I$-›)
where $\mathbf{\Sigma}^I\varphi \equiv \mathbf{\Sigma}[fp\ \mathcal{I}]\varphi$

notation *mforallInt* (**binder** ‹$\forall^I$› *[48]49*)
notation *mexistsInt* (**binder** ‹$\exists^I$› *[48]49*)

abbreviation *intneg* (‹$\neg^I$-›) **where** $\neg^I A \equiv \mathcal{I}^{d-}\ A$
abbreviation *parneg* (‹$\neg^C$-›) **where** $\neg^C A \equiv \mathcal{C}^{d-}\ A$

lemma $(\forall X.\ (\exists B.\ (X \leftrightarrow -B))) = \top$ **by** (*smt* (*verit*, *del-insts*) *compl-def dimpl-def mexists-def mforall-def setequ-def top-def*)
lemma $(\forall^I X.\ (\exists^I B.\ (X \leftrightarrow \neg^I B))) = \top$ **nitpick oops** — counterexample

4.6 Barcan formula and its converse

The converse Barcan formula follows readily from monotonicity.

lemma *CBarcan1*: *MONO* $\varphi \Longrightarrow \forall\pi.\ \varphi(\forall x.\ \pi\ x) \le (\forall x.\ \varphi(\pi\ x))$ **by** (*smt* (*verit*, *ccfv-SIG*) *MONO-def mforall-def subset-def*)
lemma *CBarcan2*: *MONO* $\varphi \Longrightarrow \forall\pi.\ (\exists x.\ \varphi(\pi\ x)) \le \varphi(\exists x.\ \pi\ x)$ **by** (*smt* (*verit*) *MONO-def mexists-def subset-def*)

However, the Barcan formula requires a stronger assumption (of an infinitary character).

lemma *Barcan1*: *iMULT*^{b} $\varphi \Longrightarrow \forall\pi.\ (\forall x.\ \varphi(\pi\ x)) \le \varphi(\forall x.\ \pi\ x)$ **unfolding** *iMULT-b-def* **by** (*smt* (*verit*) *infimum-def mforall-char image-def range-char1 subset-def*)
lemma *Barcan2*: *iADDI*^{a} $\varphi \Longrightarrow \forall\pi.\ \varphi(\exists x.\ \pi\ x) \le (\exists x.\ \varphi(\pi\ x))$ **unfolding** *iADDI-a-def* **by** (*smt* (*verit*, *ccfv-threshold*) *mexists-char image-def range-char1 subset-def supremum-def*)

Converse Barcan Formula and composition.

lemma *MONO* $\varphi \Longrightarrow \forall\pi.\ \varphi(\mathbf{\Pi}\ \pi) \le \mathbf{\Pi}(\varphi \circ \pi)$ **by** (*metis MONO-iMULTa iMULT-a-def mforall-char mforall-comp mforall-const-char*)
lemma *MONO* $\varphi \Longrightarrow \forall\pi.\ \varphi(\mathbf{\Pi}[D]\ \pi) \le \mathbf{\Pi}[D](\varphi \circ \pi)$ **by** (*smt* (*verit*) *MONO-iMULTa fun-comp-def iMULT-a-def mforall-const-char mforall-const-def image-def subset-def*)
lemma *CNTR* $\varphi \Longrightarrow$ *iMULT* $\varphi \Longrightarrow$ *IDEM* $\varphi \Longrightarrow \forall\pi.\ \varphi(\mathbf{\Pi}\{\psi\}\ \pi) \le \mathbf{\Pi}\{\psi\}(\varphi \circ \pi)$ **nitpick oops** — counterexample

Barcan Formula and composition.

lemma *iMULT*^{b} $\varphi \Longrightarrow \forall\pi.\ \mathbf{\Pi}(\varphi \circ \pi) \le \varphi(\mathbf{\Pi}\ \pi)$ **by** (*metis iMULT-b-def mforall-char mforall-comp mforall-const-char*)
lemma *iMULT*^{b} $\varphi \Longrightarrow \forall\pi.\ \mathbf{\Pi}[D](\varphi \circ \pi) \le \varphi(\mathbf{\Pi}[D]\ \pi)$ **by** (*smt* (*verit*) *fun-comp-def iMULT-b-def infimum-def mforall-const-char image-def subset-def*)
lemma *iADDI* $\varphi \Longrightarrow$ *iMULT* $\varphi \Longrightarrow \forall\pi.\ \mathbf{\Pi}\{\psi\}(\varphi \circ \pi) \le \varphi(\mathbf{\Pi}\{\psi\}\ \pi)$ **nitpick oops** — counterexample

end
theory *logics-LFI*
imports *logics-consequence conditions-relativized-infinitary*
begin

4.7 Logics of Formal Inconsistency (LFIs)

The LFIs are a family of paraconsistent logics featuring a 'consistency' operator $\circ$ that can be used to recover some classical properties of negation (in particular ECQ). We show a shallow semantical embedding of a family of self-extensional LFIs using the border operator as primitive.

Let us assume a concrete type w (for 'worlds' or 'points').

typedecl w

Let us assume the following primitive unary operation intended as a border operator.

consts $\mathcal{B}::w\ \sigma \Rightarrow w\ \sigma$

From the topological cube of opposition we have that:

abbreviation $\mathcal{C} \equiv (\mathcal{B}^{fp})^{d-}$
abbreviation $\mathcal{I} \equiv \mathcal{B}^{fp-}$
lemma $\mathcal{C}^{d-} = \mathcal{B}^{fp}$ **by** (*simp add*: *dualcompl-invol*)

Let us recall that:

lemma *expn-cntr*: *EXPN* $\mathcal{C}$ = *CNTR* $\mathcal{B}$ **by** (*metis EXPN-CNTR-dual2 EXPN-fp ofp-comm-dc1*)

For LFIs we use the negation previously defined as $\mathcal{C}^{d-} = \mathcal{B}^{fp}$.

abbreviation *neg* (‹¬-›[*70*]*71*) **where** $neg \equiv \mathcal{B}^{fp}$

In terms of the border operator the negation looks as follows (under appropriate assumptions):

lemma *neg-char*: *CNTR* $\mathcal{B} \Longrightarrow \neg A = (-A \vee \mathcal{B}\ A)$ **unfolding** *conn* **by** (*metis CNTR-def dimpl-def op-fixpoint-def subset-def*)

This negation is of course boldly paraconsistent (for both local and global consequence).

lemma $[a, \neg a \vdash b]$ **nitpick oops** — countermodel
lemma $[a, \neg a \vdash \neg b]$ **nitpick oops** — countermodel
lemma $[a, \neg a \vdash_g b]$ **nitpick oops** — countermodel
lemma $[a, \neg a \vdash_g \neg b]$ **nitpick oops** — countermodel

We define two pairs of in/consistency operators and show how they relate to each other. Using LFIs terminology, the minimal logic so encoded corresponds to RmbC + 'ciw' axiom.

abbreviation *op-inc-a*$::w\ \sigma \Rightarrow w\ \sigma$ (‹$\bullet^A$-› [*57*]*58*)
  where $\bullet^A A \equiv A \wedge \neg A$
abbreviation *op-con-a*$::w\ \sigma \Rightarrow w\ \sigma$ (‹$\circ^A$-› [*57*]*58*)
  where $\circ^A A \equiv -\bullet^A A$
abbreviation *op-inc-b*$::w\ \sigma \Rightarrow w\ \sigma$ (‹$\bullet^B$-› [*57*]*58*)
  where $\bullet^B A \equiv \mathcal{B}\ A$
abbreviation *op-con-b*$::w\ \sigma \Rightarrow w\ \sigma$ (‹$\circ^B$-› [*57*]*58*)
  where $\circ^B A \equiv \mathcal{B}^-\ A$

Observe that assumming CNTR for border we are allowed to exchange A and B variants.

lemma *pincAB*: *CNTR* $\mathcal{B} \Longrightarrow \bullet^A A = \bullet^B A$ **using** *neg-char* **by** (*metis CNTR-def CNTR-fpc L5 L6 L9 dimpl-char impl-char ofp-invol op-fixpoint-def setequ-ext svfun-compl-def*)
lemma *pconAB*: *CNTR* $\mathcal{B} \Longrightarrow \circ^A A = \circ^B A$ **by** (*metis pincAB setequ-ext svfun-compl-def*)

Variants A and B give us slightly different properties (there are countermodels for those not shown).

lemma *Prop1*: $\circ^B A = \mathcal{I}^{fp}\ A$ **by** (*simp add*: *ofp-comm-compl ofp-invol*)
lemma *Prop2*: $\circ^A A = (A \rightarrow \mathcal{I}\ A)$ **by** (*simp add*: *compl-def impl-def meet-def svfun-compl-def*)
lemma *Prop3*: *fp* $\mathcal{C}\ A \longleftrightarrow [\vdash \circ^B -A]$ **by** (*simp add*: *fp-rel gtrue-def ofp-comm-dc2 ofp-invol op-dual-def svfun-compl-def*)
lemma *Prop4a*: *fp* $\mathcal{I}\ A \longleftrightarrow [\vdash \circ^B A]$ **by** (*simp add*: *fp-rel gtrue-def ofp-comm-compl ofp-invol*)
lemma *Prop4b*: *fp* $\mathcal{I}\ A \longrightarrow [\vdash \circ^A A]$ **by** (*simp add*: *Prop2 fixpoints-def impl-def setequ-ext*)

The 'principle of gentle explosion' works for both variants (both locally and globally).

lemma $[\circ^A a,\ a,\ \neg a \vdash b]$ **by** (*metis* (*mono-tags*, *lifting*) *compl-def meet-def subset-def*)
lemma $[\circ^A a,\ a,\ \neg a \vdash_g b]$ **by** (*metis compl-def meet-def*)
lemma $[\circ^B a,\ a,\ \neg a \vdash b]$ **by** (*smt* (*z3*) *meet-def ofp-fixpoint-compl-def ofp-invol sdiff-def subset-def*)
lemma $[\circ^B a,\ a,\ \neg a \vdash_g b]$ **by** (*metis compl-def fixpoints-def fp-rel gtrue-def setequ-ext svfun-compl-def*)

abbreviation $\mathit{BORDER}\ \varphi \equiv \mathit{nMULTr}\ \varphi \wedge \mathit{CNTR}\ \varphi \wedge \mathit{nDNRM}\ \varphi \wedge \mathit{nIDEMr}^b\ \varphi$

We show how (local) contraposition variants (among others) can be recovered using the consistency operators.

lemma $[\circ^A b,\ a \rightarrow b \vdash \neg b \rightarrow \neg a]$ **nitpick oops** — countermodel
lemma *cons-lcop0-A*: $\mathit{CNTR}\ \mathcal{B} \longrightarrow [\circ^A b,\ a \rightarrow b \vdash \neg b \rightarrow \neg a]$ **by** (*smt* (*verit*, *del-insts*) *neg-char compl-def impl-char join-def meet-def subset-def*)
lemma $[\circ^B b,\ a \rightarrow b \vdash \neg b \rightarrow \neg a]$ **nitpick oops** — countermodel
lemma *cons-lcop0-B*: $\mathit{CNTR}\ \mathcal{B} \longrightarrow [\circ^B b,\ a \rightarrow b \vdash \neg b \rightarrow \neg a]$ **by** (*metis cons-lcop0-A pconAB*)
lemma $[\circ^A b,\ a \rightarrow \neg b \vdash b \rightarrow \neg a]$ **nitpick oops** — countermodel
lemma *cons-lcop1-A*: $\mathit{CNTR}\ \mathcal{B} \longrightarrow [\circ^A b,\ a \rightarrow \neg b \vdash b \rightarrow \neg a]$ **by** (*smt* (*verit*, *del-insts*) *neg-char compl-def impl-char join-def meet-def subset-def*)
lemma $[\circ^B b,\ a \rightarrow \neg b \vdash b \rightarrow \neg a]$ **nitpick oops** — countermodel
lemma *cons-lcop1-B*: $\mathit{CNTR}\ \mathcal{B} \longrightarrow [\circ^B b,\ a \rightarrow \neg b \vdash b \rightarrow \neg a]$ **by** (*metis cons-lcop1-A pconAB*)
lemma $[\circ^A b,\ \neg a \rightarrow b \vdash \neg b \rightarrow a]$ **nitpick oops** — countermodel
lemma *cons-lcop2-A*: $\mathit{CNTR}\ \mathcal{B} \longrightarrow [\circ^A b,\ \neg a \rightarrow b \vdash \neg b \rightarrow a]$ **by** (*smt* (*verit*, *del-insts*) *neg-char compl-def impl-char join-def meet-def subset-def*)
lemma $[\circ^B b,\ \neg a \rightarrow b \vdash \neg b \rightarrow a]$ **nitpick oops** — countermodel
lemma *cons-lcop2-B*: $\mathit{CNTR}\ \mathcal{B} \longrightarrow [\circ^B b,\ \neg a \rightarrow b \vdash \neg b \rightarrow a]$ **by** (*metis cons-lcop2-A pconAB*)

The following axioms are commonly employed in the literature on LFIs to obtain stronger logics. We explore under which conditions they can be assumed while keeping the logic boldly paraconsistent.

abbreviation *cf* **where** $\mathit{cf} \equiv \forall P.\ [\neg\neg P \vdash P]$
abbreviation *ce* **where** $\mathit{ce} \equiv \forall P.\ [P \vdash \neg\neg P]$
abbreviation *ciw-a* **where** $\mathit{ciw\text{-}a} \equiv \forall P.\ [\vdash \circ^A P \vee \bullet^A P]$
abbreviation *ciw-b* **where** $\mathit{ciw\text{-}b} \equiv \forall P.\ [\vdash \circ^B P \vee \bullet^B P]$
abbreviation *ci-a* **where** $\mathit{ci\text{-}a} \equiv \forall P.\ [\neg(\circ^A P) \vdash \bullet^A P]$
abbreviation *ci-b* **where** $\mathit{ci\text{-}b} \equiv \forall P.\ [\neg(\circ^B P) \vdash \bullet^B P]$
abbreviation *cl-a* **where** $\mathit{cl\text{-}a} \equiv \forall P.\ [\neg(\bullet^A P) \vdash \circ^A P]$
abbreviation *cl-b* **where** $\mathit{cl\text{-}b} \equiv \forall P.\ [\neg(\bullet^B P) \vdash \circ^B P]$
abbreviation *ca-conj-a* **where** $\mathit{ca\text{-}conj\text{-}a} \equiv \forall P\ Q.\ [\circ^A P, \circ^A Q \vdash \circ^A(P \wedge Q)]$
abbreviation *ca-conj-b* **where** $\mathit{ca\text{-}conj\text{-}b} \equiv \forall P\ Q.\ [\circ^B P, \circ^B Q \vdash \circ^B(P \wedge Q)]$
abbreviation *ca-disj-a* **where** $\mathit{ca\text{-}disj\text{-}a} \equiv \forall P\ Q.\ [\circ^A P, \circ^A Q \vdash \circ^A(P \vee Q)]$
abbreviation *ca-disj-b* **where** $\mathit{ca\text{-}disj\text{-}b} \equiv \forall P\ Q.\ [\circ^B P, \circ^B Q \vdash \circ^B(P \vee Q)]$
abbreviation *ca-impl-a* **where** $\mathit{ca\text{-}impl\text{-}a} \equiv \forall P\ Q.\ [\circ^A P, \circ^A Q \vdash \circ^A(P \rightarrow Q)]$
abbreviation *ca-impl-b* **where** $\mathit{ca\text{-}impl\text{-}b} \equiv \forall P\ Q.\ [\circ^B P, \circ^B Q \vdash \circ^B(P \rightarrow Q)]$
abbreviation *ca-a* **where** $\mathit{ca\text{-}a} \equiv \mathit{ca\text{-}conj\text{-}a} \wedge \mathit{ca\text{-}disj\text{-}a} \wedge \mathit{ca\text{-}impl\text{-}a}$
abbreviation *ca-b* **where** $\mathit{ca\text{-}b} \equiv \mathit{ca\text{-}conj\text{-}b} \wedge \mathit{ca\text{-}disj\text{-}b} \wedge \mathit{ca\text{-}impl\text{-}b}$

cf

lemma $\mathit{BORDER}\ \mathcal{B} \Longrightarrow \mathit{cf}$ **nitpick oops** — countermodel

ce

lemma $\mathit{BORDER}\ \mathcal{B} \Longrightarrow \mathit{ce}$ **nitpick oops** — countermodel

ciw

lemma *prop-ciw-a*: *ciw-a* **by** (*simp add*: *conn*)
lemma *prop-ciw-b*: *ciw-b* **by** (*simp add*: *conn svfun-compl-def*)

ci

lemma *BORDER* $\mathcal{B} \Longrightarrow$ *ci-a* **nitpick oops** — countermodel
lemma *BORDER* $\mathcal{B} \Longrightarrow$ *ci-b* **nitpick oops** — countermodel

cl

lemma *BORDER* $\mathcal{B} \Longrightarrow$ *cl-a* **nitpick oops** — countermodel
lemma *BORDER* $\mathcal{B} \Longrightarrow$ *cl-b* **nitpick oops** — countermodel

ca-conj

lemma *prop-ca-conj-b*: $nMULT^b\ \mathcal{B} = ca\text{-}conj\text{-}b$ **by** (*metis MULT-b-def nMULTb-compl sfun-compl-invol*)
lemma *prop-ca-conj-a*: $nMULTr^b\ \mathcal{B} = ca\text{-}conj\text{-}a$ **unfolding** *cond op-fixpoint-def* **by** (*smt* (*z3*) *compl-def dimpl-def join-def meet-def op-fixpoint-def subset-def subset-in-def*)

ca-disj

lemma *prop-ca-disj-b*: $ADDI^a\ \mathcal{B} = ca\text{-}disj\text{-}b$ **by** (*simp add*: *nADDI-a-def nADDIa-compl*)
lemma *prop-ca-disj-a*: $nMULTr^a\ \mathcal{B} = ca\text{-}disj\text{-}a$ **oops**

ca-impl

lemma *BORDER* $\mathcal{B} \Longrightarrow$ *ca-impl-a* **nitpick oops** — countermodel
lemma *BORDER* $\mathcal{B} \Longrightarrow$ *ca-impl-b* **nitpick oops** — countermodel

end
theory *logics-LFU*
  imports *logics-consequence conditions-relativized-infinitary*
begin

4.8 Logics of Formal Undeterminedness (LFUs)

The LFUs are a family of paracomplete logics featuring a 'determinedness' operator ¤ that can be used to recover some classical properties of negation (in particular TND). LFUs behave in a sense dually to LFIs. Both can be semantically embedded as extensions of Boolean algebras. We show a shallow semantical embedding of a family of self-extensional LFUs using the closure operator as primitive.

typedecl *w*

consts $\mathcal{C}$::$w\ \sigma \Rightarrow w\ \sigma$

abbreviation $\mathcal{I} \equiv \mathcal{C}^d$
abbreviation $\mathcal{B} \equiv (\mathcal{C}^{fp})^d$

lemma $EXPN\ \mathcal{C} = CNTR\ \mathcal{B}$ **using** *EXPN-CNTR-dual1 EXPN-fp* **by** *blast*
lemma $EXPN\ \mathcal{C} = CNTR\ \mathcal{I}$ **by** (*simp add*: *EXPN-CNTR-dual1*)

For LFUs we use the negation previously defined as $\mathcal{I}^{d-} = \mathcal{C}^-$.

abbreviation *neg* (‹¬-›[*70*]*71*) **where** $neg \equiv \mathcal{C}^-$

In terms of the border operator the negation looks as follows:

lemma *neg-char*: $EXPN\ \mathcal{C} \Longrightarrow \neg A = (-A \wedge \mathcal{B}^d\ A)$ **unfolding** *conn* **by** (*metis EXPN-def compl-def dimpl-def dual-invol op-fixpoint-def subset-def svfun-compl-def*)

abbreviation $CLOSURE\ \varphi \equiv ADDI\ \varphi \wedge EXPN\ \varphi \wedge NORM\ \varphi \wedge IDEM\ \varphi$

This negation is of course paracomplete.

lemma $CLOSURE\ \mathcal{C} \Longrightarrow [\vdash a \vee \neg a]$ **nitpick oops** — countermodel

We define two pairs of un/determinedness operators and show how they relate to each other. This logic corresponds to the paracomplete dual of the LFI 'RmbC-ciw'.

abbreviation *op-det-a*::$w\ \sigma \Rightarrow w\ \sigma$ (‹¤^{A}-› [57]58)
  where $¤^A A \equiv A \vee \neg A$
abbreviation *op-und-a*::$w\ \sigma \Rightarrow w\ \sigma$ (‹$\star^A$-› [57]58)
  where $\star^A A \equiv -¤^A A$
abbreviation *op-det-b*::$w\ \sigma \Rightarrow w\ \sigma$ (‹¤^{B}-› [57]58)
  where $¤^B A \equiv \mathcal{B}^d\ A$
abbreviation *op-und-b*::$w\ \sigma \Rightarrow w\ \sigma$ (‹$\star^B$-› [57]58)
  where $\star^B A \equiv \mathcal{B}^{d-}\ A$

Observe that assumming EXPN for closure we are allowed to exchange A and B variants.

lemma *pundAB*: $EXPN\ \mathcal{C} \Longrightarrow \star^A A = \star^B A$ **using** *neg-char* **by** (*metis BA-deMorgan1 BA-dn L4 L9 dimpl-char impl-char ofp-comm-dc2 op-fixpoint-def sdfun-dcompl-def setequ-ext svfun-compl-def*)
lemma *pdetAB*: $EXPN\ \mathcal{C} \Longrightarrow ¤^A A = ¤^B A$ **by** (*metis dual-compl-char1 pundAB sfun-compl-invol svfun-compl-def*)

Variants A and B give us slightly different properties (there are countermodels for those not shown).

lemma *Prop1*: $¤^B A = \mathcal{C}^{fp}\ A$ **by** (*simp add*: *dual-invol setequ-ext*)
lemma *Prop2*: $¤^A A = (\mathcal{C}\ A \rightarrow A)$ **unfolding** *conn* **by** (*metis compl-def svfun-compl-def*)
lemma *Prop3*: $fp\ \mathcal{I}\ A \longleftrightarrow [\vdash ¤^B -A]$ **by** (*simp add*: *dual-invol fp-d-rel gtrue-def*)
lemma *Prop4a*: $fp\ \mathcal{C}\ A \longleftrightarrow [\vdash ¤^B A]$ **by** (*simp add*: *dual-invol fp-rel gtrue-def*)
lemma *Prop4b*: $fp\ \mathcal{C}\ A \longrightarrow [\vdash ¤^A A]$ **by** (*simp add*: *compl-def fixpoints-def join-def setequ-ext svfun-compl-def*)

Recovering TND works for both variants.

lemma $[¤^A a \vdash a, \neg a]$ **by** (*simp add*: *subset-def*)
lemma $[\vdash \star^A a \vee a \vee \neg a]$ **by** (*metis compl-def join-def*)
lemma $[¤^B a \vdash a, \neg a]$ **by** (*simp add*: *compl-def dimpl-def dual-invol join-def op-fixpoint-def subset-def svfun-compl-def*)
lemma $[\vdash \star^B a \vee a \vee \neg a]$ **by** (*metis dimpl-def dual-compl-char1 dual-invol join-def ofp-comm-compl op-fixpoint-def*)

We show how (local) contraposition variants (among others) can be recovered using the determinedness operators.

lemma $[¤^A a,\ a \rightarrow b \vdash \neg b \rightarrow \neg a]$ **nitpick oops**
lemma *det-lcop0-A*: $EXPN\ \mathcal{C} \Longrightarrow [¤^A a,\ a \rightarrow b \vdash \neg b \rightarrow \neg a]$ **using** *neg-char impl-char* **unfolding** *conn order* **by** *fastforce*
lemma $[¤^B a,\ a \rightarrow b \vdash \neg b \rightarrow \neg a]$ **nitpick oops**
lemma *det-lcop0-B*: $EXPN\ \mathcal{C} \Longrightarrow [¤^B a,\ a \rightarrow b \vdash \neg b \rightarrow \neg a]$ **by** (*metis det-lcop0-A pdetAB*)
lemma $[¤^A a,\ a \rightarrow \neg b \vdash b \rightarrow \neg a]$ **nitpick oops**
lemma *det-lcop1-A*: $EXPN\ \mathcal{C} \Longrightarrow [¤^A a,\ a \rightarrow \neg b \vdash b \rightarrow \neg a]$ **by** (*smt* (*verit, ccfv-SIG*) *impl-char impl-def join-def meet-def neg-char subset-def*)
lemma $[¤^B a,\ a \rightarrow \neg b \vdash b \rightarrow \neg a]$ **nitpick oops**
lemma *det-lcop1-B*: $EXPN\ \mathcal{C} \Longrightarrow [¤^B a,\ a \rightarrow \neg b \vdash b \rightarrow \neg a]$ **by** (*metis det-lcop1-A pdetAB*)
lemma $[¤^A a,\ \neg a \rightarrow b \vdash \neg b \rightarrow a]$ **nitpick oops**
lemma *det-lcop2-A*: $EXPN\ \mathcal{C} \Longrightarrow [¤^A a,\ \neg a \rightarrow b \vdash \neg b \rightarrow a]$ **by** (*smt* (*verit, del-insts*) *neg-char compl-def impl-char join-def meet-def subset-def*)

lemma $[\text{¤}^B a, \neg a \to b \vdash \neg b \to a]$ **nitpick oops**
lemma *det-lcop2-B*: *EXPN* $\mathcal{C} \Longrightarrow [\text{¤}^B a, \neg a \to b \vdash \neg b \to a]$ **by** (*metis det-lcop2-A pdetAB*)

end

References

[1] D. Fuenmayor. Semantical investigations on non-classical logics with recovery operators: Negation. *The Logic Journal of the IGPL. Special Issue on Non-classical Modal and Predicate Logics*, 202X. To appear (preprint: https://arxiv.org/abs/2104.04284).