Transformer Semantics

Georg Struth

March 17, 2025

Abstract

These mathematical components formalise predicate transformer semantics for programs, yet currently only for partial correctness and in the absence of faults. A first part for isotone (or monotone), Sup-preserving and Inf-preserving transformers follows Back and von Wright's approach, with additional emphasis on the quantalic structure of algebras of transformers. The second part develops Sup-preserving and Inf-preserving predicate transformers from the powerset monad, via its Kleisli category and Eilenberg-Moore algebras, with emphasis on adjunctions and dualities, as well as isomorphisms between relations, state transformers and predicate transformers.

Contents

1 Introductory Remarks **2**

2 Isotone Transformers Between Complete Lattices **3**
- 2.1 Basic Properties . . . 3
- 2.2 Pre-Quantale of Isotone Transformers . . . 4
- 2.3 Propositional Hoare Logic for Transformers without Star . . . 5
- 2.4 Kleene Star of Isotone Transformers . . . 5
- 2.5 Propositional Hoare Logic Completed . . . 8
- 2.6 A Propositional Refinement Calculus . . . 9

3 Sup- and Inf-Preserving Transformers between Complete Lattices **10**
- 3.1 Basic Properties . . . 10
- 3.2 Properties of the Kleene Star . . . 14
- 3.3 Quantales of Inf- and Top-Preserving Transformers . . . 15

4 The Powerset Monad, State Transformers and Predicate Transformers **16**
- 4.1 The Powerset Monad . . . 17
- 4.2 Kleisli Category of the Powerset Monad . . . 17

4.3 Eilenberg-Moore Algebra 18
4.4 Isomorphism between Kleisli Category and Rel 21
4.5 The opposite Kleisli Category 24

5 State Transformers and Predicate Transformers Based on the Powerset Monad 26
5.1 Backward Diamonds from Kleisli Arrows 26
5.2 Backward Diamonds from Relations 29
5.3 Forward Boxes on Kleisli Arrows 31
5.4 Forward Box Operators from Relations 36
5.5 The Remaining Modalities 39

6 The Quantaloid of Kleisli Arrows 44
6.1 Kleene Star 45
6.2 Antidomain 47

7 The Quantale of Kleisli Arrows 48

1 Introductory Remarks

Predicate transformers yield standard denotational semantics for imperative programs; they have been investigated for around fifty years and are widely used in program verification. These components provide yet another take on this topic with Isabelle (previous formalisations in the AFP include [9, 5, 6]).

The first part, like Preoteasa's work [9], follows by and large Back and von Wright's seminal monograph [2]. Isotone (or monotone), sup-preserving and inf-preserving transformers are developed in a categorical setting as morphisms of orderings and complete lattices. The approach is type-driven; concepts are usually formalised with the most general suitable types. Due to this, the algebras of transformers cannot be captured within Isabelle's type classes or locales. They describe algebraic properties of typed function spaces (enriched homsets of categories of complete lattices) in terms of typed quantales or quantaloids [10]. Special focus is on notions of recursion and iteration in this typed setting. In particular, propositional Hoare logics and basic refinement calculi—for partial correctness and without assignment laws—are derived. For transformers that are endofunctions, instance proofs for quantales are given. This brings theorems about quantales and from the Kleene algebra hierarchy into scope.

Based on this, the second part presents an alternative, more detailed development with sets. It starts from the monad of the powerset functor, its Kleisli category and its Eilenberg-Moore algebras; a view that has been promoted, for instance, by Jacobs [7]. General monads cannot be handled by Isabelle's type system, only particular instances can be formalised—at the level of exercises in category theory textbooks. With this approach, binary relations, state transformers modelled as arrows of the Kleisli category of the powerset monad, and predicate transformer algebras, Sup-lattices which arise as Eilenberg-Moore algebras of the powerset monad, are related like in Jacob's state-effect triangles. In particular, the isomorphisms between the quantalic structure of relations, that of state transformers and that of various predicate transformers is spelled out in detail. In addition, the symmetries and dualities between four kinds of predicate transformers (forward and backward modal box and diamond operators in the parlance of dynamic logic) are formalised. Beyond that, the quantalic structure of state transformers is detailed first in a typed setting, and secondly in a single-typed one, where state transformers are shown to form quantales and hence Kleene algebras.

It should be straightforward to integrate these mathematical components into verification components along the lines of [1, 6]. Beyond that, an integration with the predicate transformers obtained from modal Kleene algebras [5] seems interesting for verification applications. Possible extensions and refinements include the development of verification conditions for recursion beyond those for while-loops, approaches to total correctness and fault semantics, more complete (re)encodings of Back and von Wright's approach, formalisations of domain theory, links between isotone transformers and Isabelle components for multirelational semantics [4] and extensions to probabilistic transformers [8].

2 Isotone Transformers Between Complete Lattices

theory *Isotone-Transformers*
imports *Order-Lattice-Props.Fixpoint-Fusion*
Quantales.Quantale-Star

begin

A transformer is a function between lattices; an isotone transformer preserves the order (or is monotone). In this component, statements are developed in a type-driven way. Statements are developed in more general contexts or even the most general one.

2.1 Basic Properties

First I show that some basic transformers are isotone...

lemma *iso-id*: *mono id*
by (*simp add*: *monoI*)

lemma *iso-botf*: *mono* $\bot$
by (*simp add*: *monoI*)

lemma *iso-topf*: *mono* $\top$
by (*simp add*: *monoI*)

... and that compositions, Infs and Sups preserve isotonicity.

lemma *iso-fcomp*: *mono f* $\Longrightarrow$ *mono g* $\Longrightarrow$ *mono* $(f \circ g)$
by (*simp add*: *mono-def*)

lemma *iso-fSup*:
fixes F :: ($'a$::*order* $\Rightarrow$ $'b$::*complete-lattice*) *set*
shows $(\forall f \in F.\ mono\ f) \Longrightarrow mono\ (\bigsqcup F)$
by (*simp add*: *mono-def SUP-subset-mono*)

lemma *iso-fsup*: *mono f* $\Longrightarrow$ *mono g* $\Longrightarrow$ *mono* $(f \sqcup g)$
unfolding *mono-def* **using** *sup-mono* **by** *fastforce*

lemma *iso-fInf*:
fixes F :: ($'a$::*order* $\Rightarrow$ $'b$::*complete-lattice*) *set*
shows $\forall f \in F.\ mono\ f \Longrightarrow mono\ (\bigsqcap F)$
by (*simp add*: *mono-def*, *safe*, *rule Inf-greatest*, *auto simp*: *INF-lower2*)

lemma *iso-finf*: *mono f* $\Longrightarrow$ *mono g* $\Longrightarrow$ *mono* $(f \sqcap g)$
unfolding *mono-def* **using** *inf-mono* **by** *fastforce*

lemma *fun-isol*: *mono f* $\Longrightarrow g \le h \Longrightarrow (f \circ g) \le (f \circ h)$
by (*simp add*: *le-fun-def monoD*)

lemma *fun-isor*: *mono f* $\Longrightarrow g \le h \Longrightarrow (g \circ f) \le (h \circ f)$
by (*simp add*: *le-fun-def monoD*)

2.2 Pre-Quantale of Isotone Transformers

It is well known, and has been formalised within Isabelle, that functions into complete lattices form complete lattices. In the following proof, this needs to be replayed because isotone functions are considered and closure conditions need to be respected.

Functions must now be restricted to a single type.

instantiation *iso* :: (*complete-lattice*) *unital-pre-quantale*
begin

lift-definition *one-iso* :: $'a$::*complete-lattice iso* **is** *id*
by (*simp add*: *iso-id*)

lift-definition *times-iso* :: $'a$::*complete-lattice iso* $\Rightarrow$ $'a$ *iso* $\Rightarrow$ $'a$ *iso* **is** $(\circ)$
by (*simp add*: *iso-fcomp*)

instance
by (*intro-classes*; *transfer*, *simp-all add*: *comp-assoc fInf-distr-var fInf-subdistl-var*)

end

I have previously worked in (pre)quantales with many types or quantaloids. Formally, these are categories enriched over the category of Sup-lattices (complete lattices with Sup-preserving functions). An advantage of the single-typed approach is that the definition of the Kleene star for (pre)quantales is available in this setting.

2.3 Propositional Hoare Logic for Transformers without Star

The rules of an abstract Propositional Hoare logic are derivable.

lemma *H-iso-cond1*: $(x::'a::preorder) \leq y \Longrightarrow y \leq f\ z \Longrightarrow x \leq f\ z$
using *order-trans* **by** *auto*

lemma *H-iso-cond2*: $mono\ f \Longrightarrow y \leq z \Longrightarrow x \leq f\ y \Longrightarrow x \leq f\ z$
by (*meson mono-def order-subst1*)

lemma *H-iso-seq*: $mono\ f \Longrightarrow x \leq f\ y \Longrightarrow y \leq g\ z \Longrightarrow x \leq f\ (g\ z)$
using *H-iso-cond2* **by** *force*

lemma *H-iso-seq-var*: $mono\ f \Longrightarrow x \leq f\ y \Longrightarrow y \leq g\ z \Longrightarrow x \leq (f \circ g)\ z$
by (*simp add*: *H-iso-cond2*)

lemma *H-iso-fInf*:
fixes $F :: ('a \Rightarrow 'b::complete\text{-}lattice)\ set$
shows $(\forall f \in F.\ x \leq f\ y) \Longrightarrow x \leq (\bigsqcap F)\ y$
by (*simp add*: *le-INF-iff*)

lemma *H-iso-fSup*:
fixes $F :: ('a \Rightarrow 'b::complete\text{-}lattice)\ set$
shows $F \neq \{\} \Longrightarrow (\forall f \in F.\ x \leq f\ y) \Longrightarrow x \leq (\bigsqcup F)\ y$
using *SUP-upper2* **by** *fastforce*

These rules are suitable for weakest liberal preconditions. Order-dual ones, in which the order relation is swapped, are consistent with other kinds of transformers. In the context of dynamic logic, the first set corresponds to box modalities whereas the second one would correspond to diamonds.

2.4 Kleene Star of Isotone Transformers

The Hoare rule for loops requires some preparation. On the way I verify some Kleene-algebra-style axioms for iteration.

First I show that functions form monoids.

interpretation *fun-mon*: *monoid-mult id*::$'a \Rightarrow 'a$ $(\circ)$
 by *unfold-locales auto*

definition *fiter-fun* :: $('a \Rightarrow 'c$::*semilattice-inf*$) \Rightarrow ('b \Rightarrow 'c) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'c$ **where**
 fiter-fun f g $= (\sqcap)\ f \circ (\circ)\ g$

definition *fiter* :: $('a \Rightarrow 'b$::*complete-lattice*$) \Rightarrow ('b \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$ **where**
 fiter f g = *gfp* (*fiter-fun f g*)

definition *fiter-id* :: $('a$::*complete-lattice* $\Rightarrow 'a) \Rightarrow 'a \Rightarrow 'a$ **where**
 fiter-id = *fiter id*

abbreviation *fpower* $\equiv$ *fun-mon.power*

definition *fstar* :: $('a$::*complete-lattice* $\Rightarrow 'a) \Rightarrow 'a \Rightarrow 'a$ **where**
 fstar f = ($\bigsqcap$ *i. fpower f i*)

The types in the following statements are often more general than those in the prequantale setting. I develop them generally, instead of inheriting (most of them) with more restrictive types from the quantale components.

lemma *fiter-fun-exp*: *fiter-fun f g h* $= f \sqcap (g \circ h)$
 unfolding *fiter-fun-def* **by** *simp*

The two lemmas that follow set up the relationship between the star for transformers and those in quantales.

lemma *fiter-qiter1*: *Abs-iso* (*fiter-fun* (*Rep-iso f*) (*Rep-iso g*) (*Rep-iso h*)) = *qiter-fun f g h*
 unfolding *fiter-fun-def qiter-fun-def* **by** (*metis Rep-iso-inverse comp-def sup-iso.rep-eq times-iso.rep-eq*)

lemma *fiter-qiter4*: *mono f* $\Longrightarrow$ *mono g* $\Longrightarrow$ *mono h* $\Longrightarrow$ *Rep-iso* (*qiter-fun* (*Abs-iso f*) (*Abs-iso g*) (*Abs-iso h*)) = *fiter-fun f g h*
 by (*simp add*: *Abs-iso-inverse fiter-fun-exp qiter-fun-exp sup-iso.rep-eq times-iso.rep-eq*)

The type coercions are needed to deal with isotone (monotone) functions, which had to be redefined to one single type above, in order to cooperate with the type classes for quantales. Having to deal with these coercions would be another drawback of using the quantale-based setting for the development.

lemma *iso-fiter-fun*: *mono f* $\Longrightarrow$ *mono* (*fiter-fun f*)
 by (*simp add*: *fiter-fun-exp le-fun-def mono-def inf.coboundedI2*)

lemma *iso-fiter-fun2*: *mono f* $\Longrightarrow$ *mono g* $\Longrightarrow$ *mono* (*fiter-fun f g*)
 by (*simp add*: *fiter-fun-exp le-fun-def mono-def inf.coboundedI2*)

lemma *fiter-unfoldl*:
 fixes *f* :: $'a$::*complete-lattice* $\Rightarrow 'a$
 shows *mono f* $\Longrightarrow$ *mono g* $\Longrightarrow f \sqcap (g \circ$ *fiter f g*$) =$ *fiter f g*

by (*metis fiter-def fiter-fun-exp gfp-unfold iso-fiter-fun2*)

lemma *fiter-inductl*:
fixes f :: $'a$::*complete-lattice* $\Rightarrow$ $'a$
shows *mono f* $\Longrightarrow$ *mono g* $\Longrightarrow$ $h \leq f \sqcap (g \circ h) \Longrightarrow h \leq$ *fiter f g*
by (*simp add*: *fiter-def fiter-fun-def gfp-upperbound*)

lemma *fiter-fusion*:
fixes f :: $'a$::*complete-lattice* $\Rightarrow$ $'a$
assumes *mono f*
and *mono g*
shows *fiter f g* = *fiter-id g* $\circ$ *f*
proof−
have *h1*: *mono* (*fiter-fun id g*)
by (*simp add*: *assms*(*2*) *iso-fiter-fun2 iso-id*)
have *h2*: *mono* (*fiter-fun f g*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *iso-fiter-fun2*)
have *h3*: *Inf* $\circ$ *image* ($\lambda x.\ x \circ f$) = ($\lambda x.\ x \circ f$) $\circ$ *Inf*
by (*simp add*: *fun-eq-iff image-comp*)
have ($\lambda x.\ x \circ f$) $\circ$ (*fiter-fun id g*) = (*fiter-fun f g*) $\circ$ ($\lambda x.\ x \circ f$)
by (*simp add*: *fun-eq-iff fiter-fun-def*)
thus *?thesis*
using *gfp-fusion-inf-pres*
by (*metis fiter-def fiter-id-def h1 h2 h3*)
qed

lemma *fpower-supdistl*:
fixes f :: $'a$::*complete-lattice* $\Rightarrow$ $'b$::*complete-lattice*
shows *mono f* $\Longrightarrow$ $f \circ$ *fstar g* $\leq$ ($\bigsqcap i.\ f \circ$ *fpower g i*)
by (*simp add*: *Isotone-Transformers.fun-isol fstar-def mono-INF mono-def*)

lemma *fpower-distr*: *fstar f* $\circ$ *g* = ($\bigsqcap i.$ *fpower f i* $\circ$ *g*)
by (*auto simp*: *fstar-def image-comp*)

lemma *fpower-Sup-subcomm*: *mono f* $\Longrightarrow$ $f \circ$ *fstar f* $\leq$ *fstar f* $\circ$ *f*
unfolding *fpower-distr fun-mon.power-commutes* **by** (*rule fpower-supdistl*)

lemma *fpower-inductl*:
fixes f :: $'a$::*complete-lattice* $\Rightarrow$ $'a$
shows *mono f* $\Longrightarrow$ *mono g* $\Longrightarrow$ $h \leq g \sqcap (f \circ h) \Longrightarrow h \leq$ *fpower f i* $\circ$ *g*
apply (*induct i*, *simp-all*) **by** (*metis* (*no-types*, *opaque-lifting*) *fun.map-comp fun-isol order-trans*)

lemma *fpower-inductr*:
fixes f :: $'a$::*complete-lattice* $\Rightarrow$ $'a$
shows *mono f* $\Longrightarrow$ *mono g* $\Longrightarrow$ $h \leq g \sqcap (h \circ f) \Longrightarrow h \leq g \circ$ *fpower f i*
by (*induct i*, *simp-all add*: *le-fun-def*, *metis comp-eq-elim fun-mon.power-commutes order-trans*)

lemma *fiter-fstar*: *mono f* $\Longrightarrow$ *fiter-id f* $\leq$ *fstar f*
by (*metis* (*no-types*, *lifting*) *fiter-id-def fiter-unfoldl fpower-inductl fstar-def iso-id le-INF-iff o-id order-refl*)

lemma *iso-fiter-ext*:
fixes f :: $'a$::*order* $\Rightarrow$ $'b$::*complete-lattice*
shows *mono f* $\Longrightarrow$ *mono* ($\lambda x.\ y \sqcap f\ x$)
by (*simp add*: *le-infI2 mono-def*)

lemma *fstar-pred-char*:
fixes f :: $'a$::*complete-lattice* $\Rightarrow$ $'a$
shows *mono f* $\Longrightarrow$ *fiter-id f x* = *gfp* ($\lambda y.\ x \sqcap f\ y$)
proof –
assume *hyp*: *mono f*
have $\forall g.$ (*id* $\sqcap$ ($f \circ g$)) $x = x \sqcap f\ (g\ x)$
by *simp*
hence $\forall g.$ *fiter-fun id f g x* = ($\lambda y.\ x \sqcap f\ y$) ($g\ x$)
unfolding *fiter-fun-def* **by** *simp*
thus *?thesis*
by (*simp add*: *fiter-id-def fiter-def gfp-fusion-var hyp iso-fiter-fun2 iso-id iso-fiter-ext*)
qed

2.5 Propositional Hoare Logic Completed

lemma *H-weak-loop*: *mono f* $\Longrightarrow x \leq f\ x \Longrightarrow x \leq$ *fiter-id f x*
by (*force simp*: *fstar-pred-char gfp-def intro*: *Sup-upper*)

lemma *iso-fiter*: *mono f* $\Longrightarrow$ *mono* (*fiter-id f*)
unfolding *mono-def* **by** (*subst fstar-pred-char*, *simp add*: *mono-def*)+ (*auto intro*: *gfp-mono inf-mono*)

As already mentioned, a dual Hoare logic can be built for the dual lattice. In this case, weak iteration is defined with respect to Sup.

The following standard construction lifts elements of (meet semi)lattices to transformers. I allow a more general type.

definition *fqtran* :: $'a$::*inf* $\Rightarrow$ $'a \Rightarrow$ $'a$ **where**
fqtran $x \equiv \lambda y.\ x \sqcap y$

The following standard construction lifts elements of boolean algebras to transformers.

definition *bqtran* :: $'a$::*boolean-algebra* $\Rightarrow$ $'a \Rightarrow$ $'a$ (‹⌊-⌋›) **where**
$\lfloor x \rfloor\ y = -x \sqcup y$

The conditional and while rule of Hoare logic are now derivable.

lemma *bqtran-iso*: *mono* $\lfloor x \rfloor$
by (*metis bqtran-def monoI order-refl sup.mono*)

lemma *cond-iso*: *mono f* $\Longrightarrow$ *mono g* $\Longrightarrow$ *mono* ($\lfloor x \rfloor \circ f \sqcap \lfloor y \rfloor \circ g$)

by (*simp add*: *bqtran-iso iso-fcomp iso-finf*)

lemma *loop-iso*: *mono f* $\Longrightarrow$ *mono* (*fiter-id* ($\lfloor x\rfloor \circ f$) $\circ \lfloor y\rfloor$)
by (*simp add*: *bqtran-iso iso-fcomp iso-fiter*)

lemma *H-iso-cond*: *mono f* $\Longrightarrow$ *mono g* $\Longrightarrow$ $p \sqcap x \leq f\ y \Longrightarrow q \sqcap x \leq g\ y \Longrightarrow x \leq (inf\ (\lfloor p\rfloor \circ f)\ (\lfloor q\rfloor \circ g))\ y$
by (*metis* (*full-types*) *bqtran-def comp-apply inf-apply inf-commute le-inf-iff shunt1*)

lemma *H-iso-loop*: *mono f* $\Longrightarrow p \sqcap x \leq f\ x \Longrightarrow x \leq ((fiter\text{-}id\ (\lfloor p\rfloor \circ f)) \circ \lfloor q\rfloor)\ (x \sqcap q)$
proof−
assume *a*: *mono f*
and $p \sqcap x \leq f\ x$
hence $x \leq (\lfloor p\rfloor \circ f)\ x$
using *H-iso-cond* **by** *fastforce*
hence $x \leq (fiter\text{-}id\ (\lfloor p\rfloor \circ f))\ x$
by (*simp add*: *H-weak-loop a bqtran-iso iso-fcomp*)
also have ... $\leq (fiter\text{-}id\ (\lfloor p\rfloor \circ f))\ (-q \sqcup (x \sqcap q))$
by (*meson a bqtran-iso dual-order.refl iso-fcomp iso-fiter monoD shunt1*)
finally show $x \leq ((fiter\text{-}id\ (\lfloor p\rfloor \circ f)) \circ \lfloor q\rfloor)\ (x \sqcap q)$
by (*simp add*: *bqtran-def*)
qed

lemma *btran-spec*: $x \leq \lfloor y\rfloor\ (x \sqcap y)$
by (*simp add*: *bqtran-def sup-inf-distrib1*)

lemma *btran-neg-spec*: $x \leq \lfloor -y\rfloor\ (x - y)$
by (*simp add*: *btran-spec diff-eq*)

2.6 A Propositional Refinement Calculus

Next I derive the laws of an abstract Propositional Refinement Calculus, Morgan-style. These are given without the co-called frames, which capture information about local and global variables in variants of this calculus.

definition $Ri\ x\ y\ z = \bigsqcap\{f\ z\ |f.\ x \leq f\ y \wedge mono\ (f{::}'a{::}order \Rightarrow 'b{::}complete\text{-}lattice)\}$

lemma *Ri-least*: *mono f* $\Longrightarrow x \leq f\ y \Longrightarrow Ri\ x\ y\ z \leq f\ z$
unfolding *Ri-def* **by** (*metis* (*mono-tags*, *lifting*) *Inf-lower mem-Collect-eq*)

lemma *Ri-spec*: $x \leq Ri\ x\ y\ y$
unfolding *Ri-def* **by** (*rule Inf-greatest*, *safe*)

lemma *Ri-spec-var*: $(\forall z.\ Ri\ x\ y\ z \leq f\ z) \Longrightarrow x \leq f\ y$
using *Ri-spec dual-order.trans* **by** *blast*

lemma *Ri-prop*: *mono f* $\Longrightarrow x \leq f\ y \longleftrightarrow (\forall z.\ Ri\ x\ y\ z \leq f\ z)$
using *Ri-least Ri-spec-var* **by** *blast*

lemma *iso-Ri*: *mono* (*Ri x y*)
  unfolding *mono-def Ri-def* **by** (*auto intro*!: *Inf-mono*)

lemma *Ri-weaken*: $x \leq x' \Longrightarrow\ y' \leq y \Longrightarrow Ri\ x\ y\ z \leq Ri\ x'\ y'\ z$
  by (*meson H-iso-cond2 Ri-least Ri-spec iso-Ri order.trans*)

lemma *Ri-seq*: $Ri\ x\ y\ z \leq Ri\ x\ w\ (Ri\ w\ y\ z)$
  by (*metis* (*no-types*, *opaque-lifting*) *H-iso-cond2 Ri-prop Ri-spec iso-Ri iso-fcomp o-apply*)

lemma *Ri-seq-var*: $Ri\ x\ y\ z \leq ((Ri\ x\ w) \circ (Ri\ w\ y))\ z$
  by (*simp add*: *Ri-seq*)

lemma *Ri-Inf*: $Ri\ (\prod\ X)\ y\ z \leq \prod\{Ri\ x\ y\ z\ |x.\ x \in X\}$
  by (*safe intro*!: *Inf-greatest*, *simp add*: *Ri-weaken Inf-lower*)

lemma *Ri-weak-iter*: $Ri\ x\ x\ y\ \ \leq fiter\text{-}id\ (Ri\ x\ x)\ y$
  by (*simp add*: *H-weak-loop Ri-least Ri-spec iso-Ri iso-fiter*)

lemma *Ri-cond*: $Ri\ x\ y\ z \leq (inf\ (\lfloor p\rfloor \circ (Ri\ (p \sqcap x)\ y))\ ((\lfloor q\rfloor \circ (Ri\ (q \sqcap x)\ y))))\ z$
  by (*meson H-iso-cond Ri-least Ri-spec bqtran-iso iso-Ri iso-fcomp iso-finf*)

lemma *Ri-loop*: $Ri\ x\ (q \sqcap x)\ y \leq ((fiter\text{-}id\ (\lfloor p\rfloor \circ (Ri\ (x \sqcap p)\ x))) \circ \lfloor q\rfloor)\ (q \sqcap y)$
proof−
  have $(p \sqcap x) \leq Ri\ (p \sqcap x)\ x\ x$
    by (*simp add*: *Ri-spec*)
  hence $x \leq ((fiter\text{-}id\ (\lfloor p\rfloor \circ (Ri\ (x \sqcap p)\ x))) \circ \lfloor q\rfloor)\ (q \sqcap x)$
    by (*metis H-iso-loop inf-commute iso-Ri*)
  thus *?thesis*
    apply (*subst Ri-least*, *safe*, *simp-all add*: *mono-def*)
    by (*metis bqtran-iso inf-mono iso-Ri iso-fcomp iso-fiter mono-def order-refl*)
qed

end

3 Sup- and Inf-Preserving Transformers between Complete Lattices

theory *Sup-Inf-Preserving-Transformers*
  imports *Isotone-Transformers*

begin

3.1 Basic Properties

Definitions and basic properties of Sup-preserving and Inf-preserving functions can be found in the Lattice components. The main purose of the lemmas that follow is to bring properties of isotone transformers into scope.

lemma *Sup-pres-iso*:
fixes *f* :: *'a::complete-lattice* ⇒ *'b::complete-lattice*
shows *Sup-pres f* ⟹ *mono f*
by (*simp add*: *Sup-supdistl-iso*)

lemma *Inf-pres-iso*:
fixes *f* :: *'a::complete-lattice* ⇒ *'b::complete-lattice*
shows *Inf-pres f* ⟹ *mono f*
by (*simp add*: *Inf-subdistl-iso*)

lemma *sup-pres-iso*:
fixes *f* :: *'a::lattice* ⇒ *'b::lattice*
shows *sup-pres f* ⟹ *mono f*
by (*metis le-iff-sup mono-def*)

lemma *inf-pres-iso*:
fixes *f* :: *'a::lattice* ⇒ *'b::lattice*
shows *inf-pres f* ⟹ *mono f*
by (*metis inf.absorb-iff2 monoI*)

lemma *Sup-sup-dual*:
fixes *f* :: *'a::complete-lattice* ⇒ *'b::complete-lattice*
shows *Sup-dual f* ⟹ *sup-dual f*
by (*smt comp-eq-elim image-empty image-insert inf-Inf sup-Sup*)

lemma *Inf-inf-dual*:
fixes *f* :: *'a::complete-lattice* ⇒ *'b::complete-lattice*
shows *Inf-dual f* ⟹ *inf-dual f*
by (*smt comp-eq-elim image-empty image-insert inf-Inf sup-Sup*)

lemma *Sup-bot-dual*:
fixes *f* :: *'a::complete-lattice* ⇒ *'b::complete-lattice*
shows*Sup-dual f* ⟹ *bot-dual f*
by (*metis INF-empty Sup-empty comp-eq-elim*)

lemma *Inf-top-dual*:
fixes *f* :: *'a::complete-lattice* ⇒ *'b::complete-lattice*
shows *Inf-dual f* ⟹ *top-dual f*
by (*metis Inf-empty SUP-empty comp-eq-elim*)

Next I show some basic preservation properties.

lemma *Sup-dual2*: *Sup-dual f* ⟹ *Inf-dual g* ⟹ *Sup-pres* (*g* ∘ *f*)
by (*simp add*: *fun-eq-iff image-comp*)

lemma *Inf-dual2*: *Sup-dual f* ⟹ *Inf-dual g* ⟹ *Inf-pres* (*f* ∘ *g*)
by (*simp add*: *fun-eq-iff image-comp*)

lemma *Sup-pres-id*: *Sup-pres id*
by *simp*

lemma *Inf-pres-id*: *Inf-pres id*
  by *simp*

lemma *Sup-pres-comp*: *Sup-pres f* $\Longrightarrow$ *Sup-pres g* $\Longrightarrow$ *Sup-pres* $(f \circ g)$
  by (*simp add*: *fun-eq-iff image-comp*)

lemma *Inf-pres-comp*: *Inf-pres f* $\Longrightarrow$ *Inf-pres g* $\Longrightarrow$ *Inf-pres*$(f \circ g)$
  by (*simp add*: *fun-eq-iff image-comp*)

lemma *Sup-pres-Sup*:
  fixes F :: $('a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice)$ *set*
  shows $\forall f \in F.$ *Sup-pres f* $\Longrightarrow$ *Sup-pres* $(\bigsqcup F)$
proof−
  assume *h*: $\forall f \in F.\ f \circ Sup = Sup \circ image\ f$
  hence $\forall f \in F.\ f \circ Sup \leq Sup \circ image\ (\bigsqcup F)$
    by (*simp add*: *SUP-subset-mono Sup-upper le-fun-def*)
  hence $(\bigsqcup F) \circ Sup \leq Sup \circ image\ (\bigsqcup F)$
    by (*simp add*: *SUP-le-iff le-fun-def*)
  thus *?thesis*
    by (*simp add*: *Sup-pres-iso h antisym iso-Sup-supdistl iso-fSup*)
qed

lemma *Inf-pres-Inf*:
  fixes F :: $('a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice)$ *set*
  shows $\forall f \in F.$ *Inf-pres f* $\Longrightarrow$ *Inf-pres* $(\bigsqcap F)$
proof−
  assume *h*: $\forall f \in F.\ f \circ Inf = Inf \circ image\ f$
  hence $\forall f \in F.\ Inf \circ image\ (\bigsqcap F) \leq f \circ Inf$
    by (*simp add*: *le-fun-def*, *safe*, *meson INF-lower INF-mono*)
  hence $Inf \circ image\ (\bigsqcap F) \leq (\bigsqcap F) \circ Inf$
    by (*simp add*: *le-INF-iff le-fun-def*)
  thus *?thesis*
    by (*simp add*: *Inf-pres-iso h antisym iso-Inf-subdistl iso-fInf*)
qed

lemma *Sup-pres-sup*:
  fixes f :: $'a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice$
  shows *Sup-pres f* $\Longrightarrow$ *Sup-pres g* $\Longrightarrow$ *Sup-pres* $(f \sqcup g)$
  by (*metis Sup-pres-Sup insert-iff singletonD sup-Sup*)

lemma *Inf-pres-inf*:
  fixes f :: $'a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice$
  shows *Inf-pres f* $\Longrightarrow$ *Inf-pres g* $\Longrightarrow$ *Inf-pres* $(f \sqcap g)$
  by (*metis Inf-pres-Inf inf-Inf insert-iff singletonD*)

lemma *Sup-pres-botf*: *Sup-pres* $(\lambda x.\ \bot::'a::complete\text{-}lattice)$
  by (*simp add*: *fun-eq-iff*)

It is important to note that $\lambda x.\bot$ is not Inf-preserving and that $\lambda x.\top$ is not

Sup-preserving.

lemma *Inf-pres* $(\lambda x.\ \bot{::}'a{::}complete\text{-}lattice)$
oops

lemma *Sup-pres* $(\lambda x.\ \top{::}'a{::}complete\text{-}lattice)$
oops

lemma *Inf-pres-topf*: *Inf-pres* $(\lambda x.\ \top{::}'a{::}complete\text{-}lattice)$
by (*simp add*: *fun-eq-iff*)

In complete boolean algebras, complementation yields an explicit variant of duality, which can be expressed within the language.

lemma *uminus-galois*:
fixes f :: $'a{::}complete\text{-}boolean\text{-}algebra \Rightarrow {}'b{::}complete\text{-}boolean\text{-}algebra\text{-}alt$
shows $(uminus\ f = g) = (uminus\ g = f)$
using *double-compl* **by** *force*

lemma *uminus-galois-var*:
fixes f :: $'a{::}complete\text{-}boolean\text{-}algebra\text{-}alt\text{-}with\text{-}dual \Rightarrow {}'b{::}complete\text{-}boolean\text{-}algebra\text{-}alt\text{-}with\text{-}dual$
shows $(\partial \circ f = g) = (\partial \circ g = f)$
by *force*

lemma *uminus-galois-var2*:
fixes f :: $'a{::}complete\text{-}boolean\text{-}algebra\text{-}alt\text{-}with\text{-}dual \Rightarrow {}'b{::}complete\text{-}boolean\text{-}algebra\text{-}alt\text{-}with\text{-}dual$
shows $(f \circ \partial = g) = (g \circ \partial = f)$
by *force*

lemma *uminus-mono-iff*:
fixes f :: $'a{::}complete\text{-}boolean\text{-}algebra\text{-}alt\text{-}with\text{-}dual \Rightarrow {}'b{::}complete\text{-}boolean\text{-}algebra\text{-}alt\text{-}with\text{-}dual$
shows $(\partial \circ f = \partial \circ g) = (f = g)$
using *uminus-galois-var* **by** *force*

lemma *uminus-epi-iff*:
fixes f :: $'a{::}complete\text{-}boolean\text{-}algebra\text{-}alt\text{-}with\text{-}dual \Rightarrow {}'b{::}complete\text{-}boolean\text{-}algebra\text{-}alt\text{-}with\text{-}dual$
shows $(f \circ \partial = g \circ \partial) = (f = g)$
using *uminus-galois-var2* **by** *force*

lemma *Inf-pres-Sup-pres*:
fixes f :: $'a{::}complete\text{-}boolean\text{-}algebra\text{-}alt\text{-}with\text{-}dual \Rightarrow {}'b{::}complete\text{-}boolean\text{-}algebra\text{-}alt\text{-}with\text{-}dual$
shows $(Inf\text{-}pres\ f) = (Sup\text{-}pres\ (\partial_F\ f))$
by (*simp add*: *Inf-pres-map-dual-var*)

lemma *Sup-pres-Inf-pres*:
fixes f :: $'a{::}complete\text{-}boolean\text{-}algebra\text{-}alt\text{-}with\text{-}dual \Rightarrow {}'b{::}complete\text{-}boolean\text{-}algebra\text{-}alt\text{-}with\text{-}dual$
shows $(Sup\text{-}pres\ f) = (Inf\text{-}pres\ (\partial_F\ f))$
by (*simp add*: *Sup-pres-map-dual-var*)

3.2 Properties of the Kleene Star

I develop the star for Inf-preserving functions only. This is suitable for weakest liberal preconditions. The case of sup-preserving functions is dual, and straightforward. The main difference to isotone transformers is that Kleene's fixpoint theorem now applies, that is, the star can be represented by iteration.

lemma *H-Inf-pres-fpower*:
  fixes f :: *'a::complete-lattice* ⇒ *'a*
  shows *Inf-pres f* ⟹ $x \leq f\ x$ ⟹ $x \leq$ *fpower f i x*
  apply (*induct i*, *simp-all*) **using** *H-iso-cond2 Inf-pres-iso* **by** *blast*

lemma *H-Inf-pres-fstar*:
  fixes f :: *'a::complete-lattice* ⇒ *'a*
  shows *Inf-pres f* ⟹ $x \leq f\ x$ ⟹ $x \leq$ *fstar f x*
  by (*simp add*: *H-Inf-pres-fpower fstar-def le-INF-iff*)

lemma *fpower-Inf-pres*: *Inf-pres f* ⟹ *Inf-pres* (*fpower f i*)
  by (*induct i*, *simp-all add*: *Inf-pres-comp*)

lemma *fstar-Inf-pres*:
  fixes f :: *'a::complete-lattice* ⇒ *'a*
  shows *Inf-pres f* ⟹ *Inf-pres* (*fstar f*)
  by (*simp add*: *fstar-def Inf-pres-Inf fpower-Inf-pres*)

lemma *fstar-unfoldl-var* [*simp*]:
  fixes f :: *'a::complete-lattice* ⇒ *'a*
  shows *Inf-pres f* ⟹ $x \sqcap f$ (*fstar f x*) = *fstar f x*
proof−
  assume *hyp*: *Inf-pres f*
  have $x \sqcap f$ (*fstar f x*) = *fpower f 0 x* ⊓ (⨅ *n*. *fpower f* (*Suc n*) *x*)
    by (*simp add*: *fstar-def image-comp*) (*metis* (*no-types*) *comp-apply hyp image-image*)
  also have ... = (⨅ *n*. *fpower f n x*)
    by (*subst fInf-unfold*, *auto*)
  finally show *?thesis*
    by (*simp add*: *fstar-def image-comp*)
qed

lemma *fstar-fiter-id*: *Inf-pres f* ⟹ *fstar f* = *fiter-id f*
proof−
  assume *hyp*: *Inf-pres f*
  {**fix** *x*::*'a*::*complete-lattice*
  have *fstar f x* = $x \sqcap f$ (*fstar f x*)
    by (*simp add*: *hyp*)
  hence *a*: *fstar f x* ≤ *gfp* (λ*y*. $x \sqcap f\ y$)
    by (*metis gfp-upperbound order-refl*)
  have ∀ *y*. $y \leq x \sqcap f\ y \longrightarrow y \leq$ *fstar f x*

by (*meson H-Inf-pres-fstar H-iso-cond2 Inf-pres-iso fstar-Inf-pres hyp le-infE*)
hence *fstar f x* = *gfp* ($\lambda y.$ *x* ⊓ *f y*)
by (*metis a antisym gfp-least*)**}**
thus *?thesis*
by (*simp add*: *fun-eq-iff Inf-pres-iso fstar-pred-char hyp*)
qed

lemma *fstar-unfoldl* [*simp*]:
fixes *f* :: *'a*::*complete-lattice* ⇒ *'a*
shows *Inf-pres f* ⟹ *id* ⊓ (*f* ∘ *fstar f*) = *fstar f*
by (*simp add*: *fun-eq-iff*)

lemma *fpower-Inf-comm*:
fixes *f* :: *'a*::*complete-lattice* ⇒ *'a*
shows *Inf-pres f* ⟹ *f* (⨅*i*. *fpower f i x*) = (⨅*i*. *fpower f i* (*f x*))
proof−
assume *Inf-pres f*
hence *f* (⨅*i*. *fpower f i x*) = (⨅*i*. *fpower f* (*Suc i*) *x*)
by (*simp add*: *fun-eq-iff image-comp*)
also have ... = (⨅*i*. *fpower f i* (*f x*))
by (*metis comp-eq-dest-lhs fun-mon.power-Suc2*)
finally show *?thesis* **.**
qed

lemma *fstar-comm*:
fixes *f* :: *'a*::*complete-lattice* ⇒ *'a*
shows *Inf-pres f* ⟹ *f* ∘ *fstar f* = *fstar f* ∘ *f*
apply (*simp add*: *fun-eq-iff fstar-def image-comp*)
by (*metis* (*mono-tags*, *lifting*) *INF-cong comp-eq-dest fun-mon.power-commutes*)

lemma *fstar-unfoldr* [*simp*]:
fixes *f* :: *'a*::*complete-lattice* ⇒ *'a*
shows *Inf-pres f* ⟹ *id* ⊓ (*fstar f* ∘ *f*) = *fstar f*
using *fstar-comm fstar-unfoldl* **by** *fastforce*

3.3 Quantales of Inf- and Top-Preserving Transformers

As for itotone transformers, types must now be restricted to a single one. It is well known that Inf-preserving transformers need not be top-preserving, and that Sup-preserving transformers need not be bot-preserving. This has been shown elsewhere. This does not affect the following proof, but it has an impact on how elements are represented. I show only the result for Inf-preserving transformers; that for Sup-preserving ones is dual.

typedef (**overloaded**) *'a Inf-pres* = {*f*::*'a*::*complete-lattice* ⇒ *'a*. *Inf-pres f*}
using *Inf-pres-topf* **by** *blast*

setup-lifting *type-definition-Inf-pres*

instantiation *Inf-pres* :: (*complete-lattice*) *unital-Sup-quantale*
begin

lift-definition *one-Inf-pres* :: *'a::complete-lattice Inf-pres* **is** *id*
  by (*simp add*: *iso-id*)

lift-definition *times-Inf-pres* :: *'a::complete-lattice Inf-pres* ⇒ *'a Inf-pres* ⇒ *'a Inf-pres* **is** (∘)
  by (*simp add*: *Inf-pres-comp*)

lift-definition *Sup-Inf-pres* :: *'a::complete-lattice Inf-pres set* ⇒ *'a Inf-pres* **is** *Inf*
  by (*simp add*: *Inf-pres-Inf*)

lift-definition *less-eq-Inf-pres* :: *'a Inf-pres* ⇒ *'a Inf-pres* ⇒ *bool* **is** (≥)**.**

lift-definition *less-Inf-pres* :: *'a Inf-pres* ⇒ *'a Inf-pres* ⇒ *bool* **is** (>)**.**

instance
  by (*intro-classes*; *transfer*, *simp-all add*: *o-assoc Inf-lower Inf-greatest fInf-distr-var fInf-distl-var*)

end

Three comments seem worth making. Firstly, the result bakes in duality by considering Infs in the function space as Sups in the quantale, hence as Infs in the dual quantale. Secondly, the use of Sup-quantales not only reduces the number of proof obligations. It also copes with the fact that Sups and top are not represented faithfully by this construction. They are generally different from those in the super-quantale of isotone transformers. But of course they can be defined from Infs as usual. Alternatively, I could have proved the results for Inf-quantales, which may have been more straightforward. But Sup-lattices are more conventional. Thirdly, as in the case of isotone transformers, the proof depends on a restriction to one single type, whereas previous results have been obtained for poly-typed quantales or quantaloids.

end

4 The Powerset Monad, State Transformers and Predicate Transformers

theory *Powerset-Monad*

imports *Order-Lattice-Props.Order-Lattice-Props*

begin

notation *relcomp* (**infixl** ‹;› *75*)
  and *image* (‹𝒫›)

4.1 The Powerset Monad

First I recall functoriality of the powerset functor.

lemma *P-func1*: $\mathcal{P}\ (f \circ g) = \mathcal{P}\ f \circ \mathcal{P}\ g$
  unfolding *fun-eq-iff* **by** *force*

lemma *P-func2*: $\mathcal{P}\ id = id$
  by *simp*

Isabelle' type systems doesn't allow formalising arbitrary monads, but instances such as the powerset monad can still be developed.

abbreviation *eta* :: $'a \Rightarrow 'a\ set$ (‹η›) **where**
  $\eta \equiv (\lambda x.\ \{x\})$

abbreviation *mu* :: $'a\ set\ set \Rightarrow 'a\ set$ (‹μ›) **where**
  $\mu \equiv Union$

η and μ are natural transformations.

lemma *eta-nt*: $\mathcal{P}\ f \circ \eta = \eta \circ id\ f$
  by *fastforce*

lemma *mu-nt*: $\mu \circ (\mathcal{P} \circ \mathcal{P})\ f = (\mathcal{P}\ f) \circ \mu$
  by *fastforce*

They satisfy the following coherence conditions. Explicit typing clarifies that η and μ have different type in these expressions.

lemma *pow-assoc*: $(\mu::'a\ set\ set \Rightarrow 'a\ set) \circ \mathcal{P}\ (\mu::'a\ set\ set \Rightarrow 'a\ set) = (\mu\ ::'a\ set\ set \Rightarrow 'a\ set) \circ (\mu::'a\ set\ set\ set \Rightarrow 'a\ set\ set)$
  using *fun-eq-iff* **by** *fastforce*

lemma *pow-un1*: $(\mu::'a\ set\ set \Rightarrow 'a\ set) \circ (\mathcal{P}\ (\eta::\ 'a\ \Rightarrow 'a\ set)) = (id::'a\ set\ \Rightarrow 'a\ set)$
  using *fun-eq-iff* **by** *fastforce*

lemma *pow-un2*: $(\mu::'a\ set\ set \Rightarrow 'a\ set) \circ (\eta::'a\ set \Rightarrow 'a\ set\ set) = (id::'a\ set \Rightarrow 'a\ set)$
  using *fun-eq-iff* **by** *fastforce*

Thus the powerset monad is indeed a monad.

4.2 Kleisli Category of the Powerset Monad

Next I define the Kleisli composition and Kleisli lifting (Kleisli extension) of Kleisli arrows. The Kleisli lifting turns Kleisli arrows into forward predicate transformers.

definition *kcomp* :: $('a \Rightarrow 'b\ set) \Rightarrow ('b \Rightarrow 'c\ set) \Rightarrow ('a\ \Rightarrow 'c\ set)$ (**infixl** ‹$\circ_K$› *75*) **where**

$f \circ_K g = \mu \circ \mathcal{P}\ g \circ f$

lemma *kcomp-prop*: $(f \circ_K g)\ x = (\bigsqcup y \in f\ x.\ g\ y)$
by (*simp add*: *kcomp-def*)

definition *klift* :: $('a \Rightarrow 'b\ set) \Rightarrow 'a\ set \Rightarrow 'b\ set$ (‹-†› [*101*] *100*) **where**
$f^\dagger = \mu \circ \mathcal{P}\ f$

lemma *klift-prop*: $(f^\dagger)\ X = (\bigsqcup x \in X.\ f\ x)$
by (*simp add*: *klift-def*)

lemma *kcomp-klift*: $f \circ_K g = g^\dagger \circ f$
unfolding *kcomp-def klift-def* **by** *simp*

lemma *klift-prop1*: $(f^\dagger \circ g)^\dagger = f^\dagger \circ g^\dagger$
unfolding *fun-eq-iff klift-def* **by** *simp*

lemma *klift-eta-inv1* [*simp*]: $f^\dagger \circ \eta = f$
unfolding *fun-eq-iff klift-def* **by** *simp*

lemma *klift-eta-pres* [*simp*]: $\eta^\dagger = (id::'a\ set \Rightarrow 'a\ set)$
unfolding *fun-eq-iff klift-def* **by** *simp*

lemma *klift-id-pres* [*simp*]: $id^\dagger = \mu$
unfolding *klift-def* **by** *simp*

lemma *kcomp-assoc*: $(f \circ_K g) \circ_K h = f \circ_K (g \circ_K h)$
unfolding *kcomp-klift klift-prop1* **by** *force*

lemma *kcomp-idl* [*simp*]: $\eta \circ_K f = f$
unfolding *kcomp-klift* **by** *simp*

lemma *kcomp-idr* [*simp*]: $f \circ_K \eta = f$
unfolding *kcomp-klift* **by** *simp*

In the following interpretation statement, types are restricted. This is needed for defining iteration.

interpretation *kmon*: *monoid-mult* η $(\circ_K)$
by *unfold-locales* (*simp-all add*: *kcomp-assoc*)

Next I show that η is a (contravariant) functor from Set into the Kleisli category of the powerset monad. It simply turns functions into Kleisli arrows.

lemma *eta-func1*: $\eta \circ (f \circ g) = (\eta \circ g) \circ_K (\eta \circ f)$
unfolding *fun-eq-iff kcomp-def* **by** *simp*

4.3 Eilenberg-Moore Algebra

It is well known that the Eilenberg-Moore algebras of the powerset monad form complete join semilattices (hence Sup-lattices).

First I verify that every complete lattice with structure map Sup satisfies the laws of Eilenberg-Moore algebras.

notation *Sup* (‹σ›)

lemma *em-assoc* [*simp*]: $\sigma \circ \mathcal{P}$ (σ::'*a*::*complete-lattice set* ⇒ '*a*) = $\sigma \circ \mu$
  apply (*standard*, *rule antisym*)
   apply (*simp add*: *SUP-least Sup-subset-mono Sup-upper*)
  by (*metis* (*no-types*, *lifting*) *SUP-upper2 Sup-least Sup-upper UnionE comp-def*)

lemma *em-id* [*simp*]: $\sigma \circ \eta$ = (*id*::'*a*::*complete-lattice* ⇒ '*a*)
  by (*simp add*: *fun-eq-iff*)

Hence every Sup-lattice is an Eilenberg-Moore algebra for the powerset monad. The morphisms between Eilenberg-Moore algebras of the powerset monad are Sup-preserving maps. In particular, powersets with structure map μ form an Eilenberg-Moore algebra (in fact the free one):

lemma *em-mu-assoc* [*simp*]: $\mu \circ \mathcal{P}\ \mu = \mu \circ \mu$
  by *simp*

lemma *em-mu-id* [*simp*]: $\mu \circ \eta$ = *id*
  by *simp*

Next I show that every Eilenberg-Moore algebras for the powerset functor is a Sup-lattice.

class *eilenberg-moore-pow* =
  fixes *smap* :: '*a set* ⇒ '*a*
  assumes *smap-assoc*: *smap* ∘ $\mathcal{P}$ *smap* = *smap* ∘ μ
  and *smap-id*: *smap* ∘ η = *id*

begin

definition *sleq* = (λ*x y*. *smap* {*x*,*y*} = *y*)

definition *sle* = (λ*x y*. *sleq x y* ∧ *y* ≠ *x*)

lemma *smap-un1*: *smap* {*x*, *smap Y*} = *smap* ({*x*} ∪ *Y*)
proof−
  have *smap* {*x*, *smap Y*} = *smap* {*smap* {*x*}, *smap Y*}
    by (*metis comp-apply id-apply smap-id*)
  also have ... = (*smap* ∘ $\mathcal{P}$ *smap*) {{*x*}, *Y*}
    by *simp*
  finally show *?thesis*
    using *local.smap-assoc* **by** *auto*
qed

lemma *smap-comm*: *smap* {*x*, *smap Y*} = *smap* {*smap Y*, *x*}
  by (*simp add*: *insert-commute*)

lemma *smap-un2*: *smap* $\{$*smap X*, *y*$\}$ = *smap* ($X \cup \{y\}$)
using *smap-comm smap-un1* **by** *auto*

lemma *sleq-refl*: *sleq x x*
by (*metis id-apply insert-absorb2 local.smap-id o-apply sleq-def*)

lemma *sleq-trans*: *sleq x y* $\Longrightarrow$ *sleq y z* $\Longrightarrow$ *sleq x z*
by (*metis* (*no-types*, *lifting*) *sleq-def smap-un1 smap-un2 sup-assoc*)

lemma *sleq-antisym*: *sleq x y* $\Longrightarrow$ *sleq y x* $\Longrightarrow$ $x = y$
by (*simp add*: *insert-commute sleq-def*)

lemma *smap-ub*: $x \in A \Longrightarrow$ *sleq x* (*smap A*)
using *insert-absorb sleq-def smap-un1* **by** *fastforce*

lemma *smap-lub*: ($\bigwedge x.\ x \in A \Longrightarrow$ *sleq x z*) $\Longrightarrow$ *sleq* (*smap A*) *z*
proof−
assume *h*: $\bigwedge x.\ x \in A \Longrightarrow$ *sleq x z*
have *smap* $\{$*smap A*, *z*$\}$ = *smap* ($A \cup \{z\}$)
by (*simp add*: *smap-un2*)
also have ... = *smap* (($\bigcup x \in A.\ \{x,z\}$) $\cup \{z\}$)
by (*rule-tac f=smap* **in** *arg-cong*, *auto*)
also have ... = *smap* $\{$(*smap* $\circ\ \mu$) $\{\{x,z\}\ |x.\ x \in A\}$, *z*$\}$
by (*simp add*: *Setcompr-eq-image smap-un2*)
also have ... = *smap* $\{$(*smap* $\circ\ \mathcal{P}$ *smap*) $\{\{x,z\}\ |x.\ x \in A\}$, *z*$\}$
by (*simp add*: *local.smap-assoc*)
also have ... = *smap* $\{$*smap* $\{$*smap* $\{x,z\}\ |x.\ x \in A\}$, *z*$\}$
by (*simp add*: *Setcompr-eq-image image-image*)
also have ... = *smap* $\{$*smap* $\{z\ |x.\ x \in A\}$, *z*$\}$
by (*metis h sleq-def*)
also have ... = *smap* ($\{z\ |x.\ x \in A\} \cup \{z\}$)
by (*simp add*: *smap-un2*)
also have ... = *smap* $\{z\}$
by (*rule-tac f=smap* **in** *arg-cong*, *auto*)
finally show *?thesis*
using *sleq-def sleq-refl* **by** *auto*
qed

sublocale *smap-Sup-lat*: *Sup-lattice smap sleq sle*
by *unfold-locales* (*simp-all add*: *sleq-refl sleq-antisym sleq-trans smap-ub smap-lub*)

Hence every complete lattice is an Eilenberg-Moore algebra of $\mathcal{P}$.

no-notation *Sup* (‹σ›)

end

4.4 Isomorphism between Kleisli Category and Rel

This is again well known—the isomorphism is essentially curry vs uncurry. Kleisli arrows are nondeterministic functions; they are also known as state transformers. Binary relations are very well developed in Isabelle; Kleisli composition of Kleisli arrows isn't. Ideally one should therefore use the isomorphism to transport theorems from relations to Kleisli arrows automatically. I spell out the isomorphisms and prove that the full quantalic structure, that is, complete lattices plus compositions, is preserved by the isomorphisms.

abbreviation *kzero* :: *'a* ⇒ *'b set* (‹ζ›) **where**
  ζ ≡ (λ*x*::*'a*. {})

First I define the morphisms. The second one is nothing but the graph of a function.

definition *r2f* :: (*'a* × *'b*) *set* ⇒ *'a* ⇒ *'b set* (‹$\mathcal{F}$›) **where**
  $\mathcal{F}$ *R* = *Image R* ∘ η

definition *f2r* :: (*'a* ⇒ *'b set*) ⇒ (*'a* × *'b*) *set* (‹$\mathcal{R}$›) **where**
  $\mathcal{R}$ *f* = {(*x*,*y*). *y* ∈ *f x*}

The functors form a bijective pair.

lemma *r2f2r-inv1* [*simp*]: $\mathcal{R}$ ∘ $\mathcal{F}$ = *id*
  unfolding *f2r-def r2f-def* **by** *force*

lemma *f2r2f-inv2* [*simp*]: $\mathcal{F}$ ∘ $\mathcal{R}$ = *id*
  unfolding *f2r-def r2f-def* **by** *force*

lemma *r2f-f2r-galois*: ($\mathcal{R}$ *f* = *R*) = ($\mathcal{F}$ *R* = *f*)
  by (*force simp*: *f2r-def r2f-def*)

lemma *r2f-f2r-galois-var*: ($\mathcal{R}$ ∘ *f* = *R*) = ($\mathcal{F}$ ∘ *R* = *f*)
  by (*force simp*: *f2r-def r2f-def*)

lemma *r2f-f2r-galois-var2*: (*f* ∘ $\mathcal{R}$ = *R*) = (*R* ∘ $\mathcal{F}$ = *f*)
  by (*metis* (*no-types*, *opaque-lifting*) *comp-id f2r2f-inv2 map-fun-def o-assoc r2f2r-inv1*)

lemma *r2f-inj*: *inj* $\mathcal{F}$
  by (*meson inj-on-inverseI r2f-f2r-galois*)

lemma *f2r-inj*: *inj* $\mathcal{R}$
  unfolding *inj-def* **using** *r2f-f2r-galois* **by** *metis*

lemma *r2f-mono*: ∀ *f g*. $\mathcal{F}$ ∘ *f* = $\mathcal{F}$ ∘ *g* ⟶ *f* = *g*
  by (*force simp*: *fun-eq-iff r2f-def*)

lemma *f2r-mono*: ∀ *f g*. $\mathcal{R}$ ∘ *f* = $\mathcal{R}$ ∘ *g* ⟶ *f* = *g*
  by (*force simp*: *fun-eq-iff f2r-def*)

lemma *r2f-mono-iff*: $(\mathcal{F} \circ f = \mathcal{F} \circ g) = (f = g)$
  using *r2f-mono* **by** *blast*

lemma *f2r-mono-iff* : $(\mathcal{R} \circ f = \mathcal{R} \circ g) = (f = g)$
  using *f2r-mono* **by** *blast*

lemma *r2f-inj-iff*: $(\mathcal{R}\ f = \mathcal{R}\ g) = (f = g)$
  by (*simp add*: *f2r-inj inj-eq*)

lemma *f2r-inj-iff*: $(\mathcal{F}\ R = \mathcal{F}\ S) = (R = S)$
  by (*simp add*: *r2f-inj inj-eq*)

lemma *r2f-surj*: *surj* $\mathcal{F}$
  by (*metis r2f-f2r-galois surj-def*)

lemma *f2r-surj*: *surj* $\mathcal{R}$
  using *r2f-f2r-galois* **by** *auto*

lemma *r2f-epi*: $\forall f\ g.\ f \circ \mathcal{F} = g \circ \mathcal{F} \longrightarrow f = g$
  by (*metis r2f-f2r-galois-var2*)

lemma *f2r-epi*: $\forall f\ g.\ f \circ \mathcal{R} = g \circ \mathcal{R} \longrightarrow f = g$
  by (*metis r2f-f2r-galois-var2*)

lemma *r2f-epi-iff*: $(f \circ \mathcal{F} = g \circ \mathcal{F}) = (f = g)$
  using *r2f-epi* **by** *blast*

lemma *f2r-epi-iff*: $(f \circ \mathcal{R} = g \circ \mathcal{R}) = (f = g)$
  using *f2r-epi* **by** *blast*

lemma *r2f-bij*: *bij* $\mathcal{F}$
  by (*simp add*: *bijI r2f-inj r2f-surj*)

lemma *f2r-bij*: *bij* $\mathcal{R}$
  by (*simp add*: *bij-def f2r-inj f2r-surj*)

r2f is essentially curry and f2r is uncurry, yet in Isabelle the type of sets and predicates (boolean-valued functions) are different. Collect transforms predicates into sets and the following function sets into predicates:

abbreviation *s2p* $X \equiv (\lambda x.\ x \in X)$

lemma *r2f-curry*: *r2f* $R =$ *Collect* $\circ$ (*curry* $\circ$ *s2p*) R
  by (*force simp*: *r2f-def fun-eq-iff curry-def*)

lemma *f2r-uncurry*: *f2r* $f =$ (*Collect* $\circ$ *case-prod*) (*s2p* $\circ f$)
  by (*force simp*: *fun-eq-iff f2r-def*)

Uncurry is case-prod in Isabelle.

f2r and r2f preserve the quantalic structures of relations and Kleisli arrows. In particular they are functors.

lemma *r2f-comp-pres*: $\mathcal{F}\ (R\ ;\ S) = \mathcal{F}\ R \circ_K \mathcal{F}\ S$
  unfolding *fun-eq-iff r2f-def kcomp-def* **by** *force*

lemma *r2f-Id-pres* [*simp*]: $\mathcal{F}\ Id = \eta$
  unfolding *fun-eq-iff r2f-def* **by** *simp*

lemma *r2f-Sup-pres*: *Sup-pres* $\mathcal{F}$
  unfolding *fun-eq-iff r2f-def* **by** *force*

lemma *r2f-Sup-pres-var*: $\mathcal{F}\ (\bigcup R) = (\bigsqcup r \in R.\ \mathcal{F}\ r)$
  unfolding *r2f-def* **by** *force*

lemma *r2f-sup-pres*: *sup-pres* $\mathcal{F}$
  unfolding *r2f-def* **by** *force*

lemma *r2f-Inf-pres*: *Inf-pres* $\mathcal{F}$
  unfolding *fun-eq-iff r2f-def* **by** *force*

lemma *r2f-Inf-pres-var*: $\mathcal{F}\ (\bigcap R) = (\bigcap r \in R.\ \mathcal{F}\ r)$
  unfolding *r2f-def* **by** *force*

lemma *r2f-inf-pres*: *inf-pres* $\mathcal{F}$
  unfolding *r2f-def* **by** *force*

lemma *r2f-bot-pres*: *bot-pres* $\mathcal{F}$
  by (*metis SUP-empty Sup-empty r2f-Sup-pres-var*)

lemma *r2f-top-pres*: *top-pres* $\mathcal{F}$
  by (*metis Sup-UNIV r2f-Sup-pres-var r2f-surj*)

lemma *r2f-leq*: $(R \subseteq S) = (\mathcal{F}\ R \leq \mathcal{F}\ S)$
  by (*metis le-iff-sup r2f-f2r-galois r2f-sup-pres*)

Dual statements for f2r hold. Can one automate this?

lemma *f2r-kcomp-pres*: $\mathcal{R}\ (f \circ_K g) = \mathcal{R}\ f\ ;\ \mathcal{R}\ g$
  by (*simp add*: *r2f-f2r-galois r2f-comp-pres pointfree-idE*)

lemma *f2r-eta-pres* [*simp*]: $\mathcal{R}\ \eta = Id$
  by (*simp add*: *r2f-f2r-galois*)

lemma *f2r-Sup-pres*:*Sup-pres* $\mathcal{R}$
 by (*auto simp*: *r2f-f2r-galois-var comp-assoc*[*symmetric*] *r2f-Sup-pres image-comp*)

lemma *f2r-Sup-pres-var*: $\mathcal{R}\ (\bigsqcup F) = (\bigsqcup f \in F.\ \mathcal{R}\ f)$
  by (*simp add*: *r2f-f2r-galois r2f-Sup-pres-var image-comp*)

lemma *f2r-sup-pres*: *sup-pres* $\mathcal{R}$

by (*simp add*: *r2f-f2r-galois r2f-sup-pres pointfree-idE*)

lemma *f2r-Inf-pres*: *Inf-pres* $\mathcal{R}$
by (*auto simp*: *r2f-f2r-galois-var comp-assoc*[*symmetric*] *r2f-Inf-pres image-comp*)

lemma *f2r-Inf-pres-var*: $\mathcal{R}$ $(\bigsqcap F) = (\bigcap f \in F.\ \mathcal{R}\ f)$
by (*simp add*: *r2f-f2r-galois r2f-Inf-pres-var image-comp*)

lemma *f2r-inf-pres*: *inf-pres* $\mathcal{R}$
by (*simp add*: *r2f-f2r-galois r2f-inf-pres pointfree-idE*)

lemma *f2r-bot-pres*: *bot-pres* $\mathcal{R}$
by (*simp add*: *r2f-bot-pres r2f-f2r-galois*)

lemma *f2r-top-pres*: *top-pres* $\mathcal{R}$
by (*simp add*: *r2f-f2r-galois r2f-top-pres*)

lemma *f2r-leq*: $(f \le g) = (\mathcal{R}\ f \subseteq \mathcal{R}\ g)$
by (*metis r2f-f2r-galois r2f-leq*)

Relational subidentities are isomorphic to particular Kleisli arrows.

lemma *r2f-Id-on1*: $\mathcal{F}$ (*Id-on X*) $= (\lambda x.$ *if* $x \in X$ *then* $\{x\}$ *else* $\{\})$
by (*force simp add*: *fun-eq-iff r2f-def Id-on-def*)

lemma *r2f-Id-on2*: $\mathcal{F}$ (*Id-on X*) $\circ_K f = (\lambda x.$ *if* $x \in X$ *then* $f\ x$ *else* $\{\})$
unfolding *fun-eq-iff Id-on-def r2f-def kcomp-def* **by** *auto*

lemma *r2f-Id-on3*: $f \circ_K \mathcal{F}$ (*Id-on X*) $= (\lambda x.\ X \cap f\ x)$
unfolding *kcomp-def r2f-def Id-on-def fun-eq-iff* **by** *auto*

4.5 The opposite Kleisli Category

Opposition is funtamental for categories; yet hard to realise in Isabelle in general. Due to the access to relations, the Kleisli category of the powerset functor is an exception.

notation *converse* (‹$\smile$›)

definition *kop* :: ($'a \Rightarrow 'b$ *set*) $\Rightarrow 'b \Rightarrow 'a$ *set* (‹op_K›) **where**
$op_K = \mathcal{F} \circ (\smile) \circ \mathcal{R}$

Kop is a contravariant functor.

lemma *kop-contrav*: $op_K\ (f \circ_K g) = op_K\ g \circ_K op_K\ f$
unfolding *kop-def r2f-def f2r-def converse-def kcomp-def fun-eq-iff comp-def* **by** *fastforce*

lemma *kop-func2* [*simp*]: $op_K\ \eta = \eta$
unfolding *kop-def r2f-def f2r-def converse-def comp-def fun-eq-iff* **by** *fastforce*

lemma *converse-idem* [*simp*]: $(\smile) \circ (\smile) = id$
using *comp-def* **by** *auto*

lemma *converse-galois*: $((\smile) \circ f = g) = ((\smile) \circ g = f)$
by *auto*

lemma *converse-galois2*: $(f \circ (\smile) = g) = (g \circ (\smile) = f)$
apply (*simp add*: *fun-eq-iff*)
by (*metis converse-converse*)

lemma *converse-mono-iff*: $((\smile) \circ f = (\smile) \circ g) = (f = g)$
using *converse-galois* **by** *force*

lemma *converse-epi-iff*: $(f \circ (\smile) = g \circ (\smile)) = (f = g)$
using *converse-galois2* **by** *force*

lemma *kop-idem* [*simp*]: $op_K \circ op_K = id$
unfolding *kop-def comp-def fun-eq-iff* **by** (*metis converse-converse id-apply r2f-f2r-galois*)

lemma *kop-galois*: $(op_K\ f = g) = (op_K\ g = f)$
by (*metis kop-idem pointfree-idE*)

lemma *kop-galois-var*: $(op_K \circ f = g) = (op_K \circ g = f)$
by (*auto simp*: *kop-def f2r-def r2f-def converse-def fun-eq-iff*)

lemma *kop-galois-var2*: $(f \circ op_K = g) = (g \circ op_K = f)$
by (*metis* (*no-types*, *opaque-lifting*) *comp-assoc comp-id kop-idem*)

lemma *kop-inj*: *inj* op_K
unfolding *inj-def* **by** (*simp add*: *f2r-inj-iff kop-def r2f-inj-iff*)

lemma *kop-inj-iff*: $(op_K\ f = op_K\ g) = (f = g)$
by (*simp add*: *inj-eq kop-inj*)

lemma *kop-surj*: *surj* op_K
unfolding *surj-def* **by** (*metis kop-galois*)

lemma *kop-bij*: *bij* op_K
by (*simp add*: *bij-def kop-inj kop-surj*)

lemma *kop-mono*: $(op_K \circ f = op_K \circ g) \Longrightarrow (f = g)$
by (*simp add*: *fun.inj-map inj-eq kop-inj*)

lemma *kop-mono-iff*: $(op_K \circ f = op_K \circ g) = (f = g)$
using *kop-mono* **by** *blast*

lemma *kop-epi*: $(f \circ op_K = g \circ op_K) \Longrightarrow (f = g)$
by (*metis kop-galois-var2*)

lemma *kop-epi-iff*: $(f \circ op_K = g \circ op_K) = (f = g)$
using *kop-epi* **by** *blast*

lemma *Sup-pres-kop*: *Sup-pres* op_K
unfolding *kop-def fun-eq-iff comp-def r2f-def f2r-def converse-def* **by** *auto*

lemma *Inf-pres-kop*: *Inf-pres* op_K
unfolding *kop-def fun-eq-iff comp-def r2f-def f2r-def converse-def* **by** *auto*

end

5 State Transformers and Predicate Transformers Based on the Powerset Monad

theory *Kleisli-Transformers*

imports *Powerset-Monad*
Sup-Inf-Preserving-Transformers
begin

5.1 Backward Diamonds from Kleisli Arrows

First I verify the embedding of the Kleisli category of the powerset functor into its Eilenberg-Moore category. This functor maps sets to their mus and functions to their Kleisli liftings. But this is just functoriality of dagger!. I model it as a backward diamond operator in the sense of dynamic logic. It corresponds to a strongest postcondition operator. In the parlance of program semantics, this is an embedding of state into prediate transformers.

notation *klift* $(\langle bd_{\mathcal{F}} \rangle)$

bd stands for backward diamond, the index indicates the setting of Kleisli arrows or nondeterministic functions. ifbd is its inverse.

abbreviation *ifbd* :: $('a\ set \Rightarrow 'b\ set) \Rightarrow 'a \Rightarrow 'b\ set$ $(\langle bd^-_{\mathcal{F}} \rangle)$ **where**
$bd^-_{\mathcal{F}} \equiv (\lambda\varphi.\ \varphi \circ \eta)$

lemma *fbd-set*: $bd_{\mathcal{F}}\ f\ X = \{y.\ \exists x.\ y \in f\ x \wedge x \in X\}$
by (*force simp*: *klift-prop*)

lemma *ifbd-set*: $bd^-_{\mathcal{F}}\ \varphi\ x = \{y.\ y \in \varphi\ \{x\}\}$
by *simp*

The two functors form a bijective pair.

lemma *fbd-ifbd-inv2*: *Sup-pres* $\varphi \Longrightarrow (bd_{\mathcal{F}} \circ bd^-_{\mathcal{F}})\ \varphi = \varphi$
proof −
assume *h*: *Sup-pres* φ
have $(bd_{\mathcal{F}} \circ bd^-_{\mathcal{F}})\ \varphi = Sup \circ \mathcal{P}\ (\varphi \circ \eta)$

unfolding *klift-def* **by** *simp*
also have ... = $Sup \circ \mathcal{P}\ \varphi \circ \mathcal{P}\ \eta$
by (*simp add*: *comp-assoc P-func1*)
also have ... = $\varphi \circ Sup \circ \mathcal{P}\ \eta$
by (*simp add*: *h*)
also have ... = $\varphi \circ id$
by *force*
finally show *?thesis*
by *simp*
qed

lemma *fbd-ifbd-inv2-inv*: $(bd_{\mathcal{F}} \circ bd^{-}{}_{\mathcal{F}})\ \varphi = \varphi \Longrightarrow Sup\text{-}pres\ \varphi$
unfolding *fun-eq-iff comp-def* **by** (*metis* (*no-types*, *lifting*) *Inf.INF-cong UN-extend-simps*(*8*) *klift-prop*)

lemma *fbd-ifbd-inv2-iff*: $((bd_{\mathcal{F}} \circ bd^{-}{}_{\mathcal{F}})\ \varphi = \varphi) = (Sup\text{-}pres\ \varphi)$
using *fbd-ifbd-inv2 fbd-ifbd-inv2-inv* **by** *force*

lemma *fbd-inj*: *inj* $bd_{\mathcal{F}}$
by (*meson inj-on-inverseI klift-eta-inv1*)

lemma *fbd-inj-iff*: $(bd_{\mathcal{F}}\ f = bd_{\mathcal{F}}\ g) = (f = g)$
by (*meson injD fbd-inj*)

lemma *ifbd-inj*: $Sup\text{-}pres\ \varphi \Longrightarrow Sup\text{-}pres\ \psi \Longrightarrow bd^{-}{}_{\mathcal{F}}\ \varphi = bd^{-}{}_{\mathcal{F}}\ \psi \Longrightarrow \varphi = \psi$
proof−
assume *h1*: $Sup\text{-}pres\ \varphi$
and *h2*: $Sup\text{-}pres\ \psi$
and $bd^{-}{}_{\mathcal{F}}\ \varphi = bd^{-}{}_{\mathcal{F}}\ \psi$
hence $(bd_{\mathcal{F}} \circ bd^{-}{}_{\mathcal{F}})\ \varphi = (bd_{\mathcal{F}} \circ bd^{-}{}_{\mathcal{F}})\ \psi$
by *simp*
thus *?thesis*
by (*metis h1 h2 fbd-ifbd-inv2*)
qed

lemma *ifbd-inj-iff*: $Sup\text{-}pres\ \varphi \Longrightarrow Sup\text{-}pres\ \psi \Longrightarrow (bd^{-}{}_{\mathcal{F}}\ \varphi = bd^{-}{}_{\mathcal{F}}\ \psi) = (\varphi = \psi)$
using *ifbd-inj* **by** *force*

lemma *fbd-ifbd-galois*: $Sup\text{-}pres\ \varphi \Longrightarrow (bd^{-}{}_{\mathcal{F}}\ \varphi = f) = (bd_{\mathcal{F}}\ f = \varphi)$
using *fbd-ifbd-inv2* **by** *force*

lemma *fbd-surj*: $Sup\text{-}pres\ \varphi \Longrightarrow (\exists f.\ bd_{\mathcal{F}}\ f = \varphi)$
using *fbd-ifbd-inv2* **by** *auto*

lemma *ifbd-surj*: *surj* $bd^{-}{}_{\mathcal{F}}$
unfolding *surj-def* **by** (*metis klift-eta-inv1*)

In addition they preserve the Sup-quantale structure of the powerset algebra.

This means that morphisms preserve compositions, units and Sups, but not Infs, hence also bottom but not top.

lemma *fbd-comp-pres*: $bd_{\mathcal{F}}\ (f \circ_K g) = bd_{\mathcal{F}}\ g \circ bd_{\mathcal{F}}\ f$
unfolding *kcomp-klift klift-prop1* **by** *simp*

lemma *fbd-Sup-pres*: *Sup-pres* $bd_{\mathcal{F}}$
by (*force simp*: *fun-eq-iff klift-def*)

lemma *fbd-sup-pres*: *sup-pres* $bd_{\mathcal{F}}$
using *Sup-sup-pres fbd-Sup-pres* **by** *blast*

lemma *fbd-Disj*: *Sup-pres* ($bd_{\mathcal{F}}\ f$)
by (*simp add*: *fbd-ifbd-inv2-inv*)

lemma *fbd-disj*: *sup-pres* ($bd_{\mathcal{F}}\ f$)
by (*simp add*: *klift-prop*)

lemma *fbd-bot-pres*: *bot-pres* $bd_{\mathcal{F}}$
unfolding *klift-def* **by** *fastforce*

lemma *fbd-zero-pres2* [*simp*]: $bd_{\mathcal{F}}\ f\ \{\} = \{\}$
by (*simp add*: *klift-prop*)

lemma *fbd-iso*: $X \subseteq Y \longrightarrow bd_{\mathcal{F}}\ f\ X \subseteq bd_{\mathcal{F}}\ f\ Y$
by (*metis fbd-disj le-iff-sup*)

The following counterexamples show that Infs are not preserved.

lemma *top-pres* $bd_{\mathcal{F}}$
oops

lemma *inf-pres* $bd_{\mathcal{F}}$
oops

Dual preservation statements hold for ifbd ... and even Inf-preservation.

lemma *ifbd-comp-pres*: *Sup-pres* $\varphi \Longrightarrow bd^-{}_{\mathcal{F}}\ (\varphi \circ \psi) = bd^-{}_{\mathcal{F}}\ \psi \circ_K bd^-{}_{\mathcal{F}}\ \varphi$
by (*smt fbd-ifbd-galois fun.map-comp kcomp-def klift-def*)

lemma *ifbd-Sup-pres*: *Sup-pres* $bd^-{}_{\mathcal{F}}$
by (*simp add*: *fun-eq-iff*)

lemma *ifbd-sup-pres*: *sup-pres* $bd^-{}_{\mathcal{F}}$
by *force*

lemma *ifbd-Inf-pres*: *Inf-pres* $bd^-{}_{\mathcal{F}}$
by (*simp add*: *fun-eq-iff*)

lemma *ifbd-inf-pres*: *inf-pres* $bd^-{}_{\mathcal{F}}$
by *force*

lemma *ifbd-bot-pres*: *bot-pres* $bd^{-}{}_{\mathcal{F}}$
by *auto*

lemma *ifbd-top-pres*: *top-pres* $bd^{-}{}_{\mathcal{F}}$
by *auto*

Preservation of units by the Kleisli lifting has been proved in klift-prop3.

These results estabilish the isomorphism between state and predicate transformers given by backward diamonds. The isomorphism preserves the Sup-quantale structure, but not Infs.

5.2 Backward Diamonds from Relations

Using the isomorphism between binary relations and Kleisli arrows (or state transformers), it is straightforward to define backward diamonds from relations, by composing isomorphisms. It follows that Sup-quantales of binary relations (under relational composition, the identity relation and Sups) are isomorphic to the Sup-quantales of predicate transformers. Once again, Infs are not preserved.

definition *rbd* :: $('a \times 'b)$ *set* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'b$ *set* $(\langle bd_{\mathcal{R}} \rangle)$ **where**
$bd_{\mathcal{R}} = bd_{\mathcal{F}} \circ \mathcal{F}$

definition *irbd* :: $('a\ set \Rightarrow 'b\ set) \Rightarrow ('a \times 'b)\ set$ $(\langle bd^{-}{}_{\mathcal{R}} \rangle)$ **where**
$bd^{-}{}_{\mathcal{R}} = \mathcal{R} \circ bd^{-}{}_{\mathcal{F}}$

lemma *rbd-Im*: $bd_{\mathcal{R}} = (``)$
unfolding *rbd-def klift-def r2f-def fun-eq-iff* **by** *force*

lemma *rbd-set*: $bd_{\mathcal{R}}\ R\ X = \{y.\ \exists x \in X.\ (x,y) \in R\}$
by (*force simp*: *rbd-Im Image-def*)

lemma *irbd-set*: $bd^{-}{}_{\mathcal{R}}\ \varphi = \{(x,y).\ y \in (\varphi \circ \eta)\ x\}$
by (*simp add*: *irbd-def f2r-def o-def*)

lemma *irbd-set-var*: $bd^{-}{}_{\mathcal{R}}\ \varphi = \{(x,y).\ y \in \varphi\ \{x\}\}$
by (*simp add*: *irbd-def f2r-def o-def*)

lemma *rbd-irbd-inv1* [*simp*]: $bd^{-}{}_{\mathcal{R}} \circ bd_{\mathcal{R}} = id$
by (*metis* (*no-types*, *lifting*) *comp-eq-dest-lhs eq-id-iff fbd-Disj fbd-ifbd-galois irbd-def r2f-f2r-galois rbd-def*)

lemma *irbd-rbd-inv2*: *Sup-pres* $\varphi \Longrightarrow (bd_{\mathcal{R}} \circ bd^{-}{}_{\mathcal{R}})\ \varphi = \varphi$
by (*metis comp-apply fbd-ifbd-galois irbd-def r2f-f2r-galois rbd-def*)

lemma *irbd-rbd-inv2-inv*: $(bd_{\mathcal{R}} \circ bd^{-}{}_{\mathcal{R}})\ \varphi = \varphi \Longrightarrow$ *Sup-pres* φ
by (*simp add*: *rbd-def irbd-def*, *metis fbd-Disj*)

lemma *irbd-rbd-inv2-iff*: $((bd_{\mathcal{R}} \circ bd^{-}{}_{\mathcal{R}})\ \varphi = \varphi) = (\textit{Sup-pres}\ \varphi)$
using *irbd-rbd-inv2 irbd-rbd-inv2-inv* **by** *blast*

lemma *rbd-inj*: *inj* $bd_{\mathcal{R}}$
by (*simp add*: *fbd-inj inj-compose r2f-inj rbd-def*)

lemma *rbd-translate*: $(bd_{\mathcal{R}}\ R = bd_{\mathcal{R}}\ S) = (R = S)$
by (*simp add*: *rbd-inj inj-eq*)

lemma *irbd-inj*: $\textit{Sup-pres}\ \varphi \Longrightarrow \textit{Sup-pres}\ \psi \Longrightarrow bd^{-}{}_{\mathcal{R}}\ \varphi = bd^{-}{}_{\mathcal{R}}\ \psi \Longrightarrow \varphi = \psi$
by (*metis rbd-Im comp-eq-dest-lhs irbd-rbd-inv2*)

lemma *irbd-inj-iff*: $\textit{Sup-pres}\ \varphi \Longrightarrow \textit{Sup-pres}\ \psi \Longrightarrow (bd^{-}{}_{\mathcal{R}}\ \varphi = bd^{-}{}_{\mathcal{R}}\ \psi) = (\varphi = \psi)$
using *irbd-inj* **by** *force*

lemma *rbd-surj*: $\textit{Sup-pres}\ \varphi \Longrightarrow (\exists R.\ bd_{\mathcal{R}}\ R = \varphi)$
using *irbd-rbd-inv2* **by** *force*

lemma *irbd-surj*: *surj* $bd^{-}{}_{\mathcal{R}}$
by (*metis UNIV-I fun.set-map imageE rbd-irbd-inv1 surj-def surj-id*)

lemma *rbd-irbd-galois*: $\textit{Sup-pres}\ \varphi \Longrightarrow (\varphi = bd_{\mathcal{R}}\ R) = (R = bd^{-}{}_{\mathcal{R}}\ \varphi)$
by (*smt comp-apply fbd-ifbd-galois irbd-def r2f-f2r-galois rbd-def*)

lemma *rbd-comp-pres*: $bd_{\mathcal{R}}\ (R\ ;\ S) = bd_{\mathcal{R}}\ S \circ bd_{\mathcal{R}}\ R$
by (*simp add*: *rbd-def r2f-comp-pres fbd-comp-pres*)

lemma *rbd-Id-pres*: $bd_{\mathcal{R}}\ Id = id$
unfolding *rbd-def* **by** *simp*

lemma *rbd-Un-pres*: *Sup-pres* $bd_{\mathcal{R}}$
by (*simp add*: *rbd-def Sup-pres-comp fbd-Sup-pres r2f-Sup-pres*)

lemma *rbd-un-pres*: *sup-pres* $bd_{\mathcal{R}}$
by (*simp add*: *rbd-def fbd-sup-pres r2f-sup-pres*)

lemma *inf-pres* $bd_{\mathcal{R}}$
oops

lemma *rbd-disj*: *Sup-pres* $(bd_{\mathcal{R}}\ R)$
by (*simp add*: *rbd-def fbd-Disj*)

lemma *rbd-disj2*: *sup-pres* $(bd_{\mathcal{R}}\ R)$
by (*simp add*: *Image-Un rbd-Im*)

lemma *rbd-bot-pres*: *bot-pres* $bd_{\mathcal{R}}$
by (*simp add*: *fbd-bot-pres r2f-bot-pres rbd-def*)

lemma *rbd-zero-pres2* [*simp*]: $bd_{\mathcal{R}}\ R\ \{\} = \{\}$
  by (*simp add*: *rbd-Im*)

lemma *rbd-univ*: $bd_{\mathcal{R}}\ R\ UNIV = Range\ R$
  unfolding *rbd-def fun-eq-iff klift-def r2f-def* **by** *force*

lemma *rbd-iso*: $X \subseteq Y \Longrightarrow bd_{\mathcal{R}}\ R\ X \subseteq bd_{\mathcal{R}}\ R\ Y$
  by (*metis le-iff-sup rbd-disj2*)

lemma *irbd-comp-pres*: $Sup\text{-}pres\ \varphi \Longrightarrow bd^{-}{}_{\mathcal{R}}\ (\varphi \circ \psi) = bd^{-}{}_{\mathcal{R}}\ \psi\ ;\ bd^{-}{}_{\mathcal{R}}\ \varphi$
  by (*simp add*: *ifbd-comp-pres f2r-kcomp-pres irbd-def*)

lemma *irbd-id-pres* [*simp*]: $bd^{-}{}_{\mathcal{R}}\ id = Id$
  unfolding *irbd-def* **by** *simp*

lemma *irbd-Sup-pres*: $Sup\text{-}pres\ bd^{-}{}_{\mathcal{R}}$
  by (*simp add*: *irbd-def Sup-pres-comp ifbd-Sup-pres f2r-Sup-pres*)

lemma *irbd-sup-pres*: $sup\text{-}pres\ bd^{-}{}_{\mathcal{R}}$
  by (*simp add*: *irbd-def ifbd-sup-pres f2r-sup-pres*)

lemma *irbd-Inf-pres*: $Inf\text{-}pres\ bd^{-}{}_{\mathcal{R}}$
  by (*auto simp*: *fun-eq-iff irbd-def f2r-def*)

lemma *irbd-inf-pres*: $inf\text{-}pres\ bd^{-}{}_{\mathcal{R}}$
  by (*auto simp*: *fun-eq-iff irbd-def f2r-def*)

lemma *irbd-bot-pres*: $bot\text{-}pres\ bd^{-}{}_{\mathcal{R}}$
  by (*metis comp-def ifbd-bot-pres f2r-bot-pres irbd-def*)

This shows that relations are isomorphic to disjunctive forward predicate transformers. In many cases Isabelle picks up the composition of morphisms in proofs.

5.3 Forward Boxes on Kleisli Arrows

Forward box operators correspond to weakest liberal preconditions in program semantics. Here, Kleisli arrows are mapped to the opposite of the Eilenberg-Moore category, that is, Inf-lattices. It follows that the Inf-quantale structure is preserved. Modelling opposition is based on the fact that Kleisli arrows can be swapped by going through relations.

definition *ffb* :: $('a \Rightarrow 'b\ set) \Rightarrow 'b\ set \Rightarrow 'a\ set$ ($\langle fb_{\mathcal{F}} \rangle$) **where**
  $fb_{\mathcal{F}} = \partial_F \circ bd_{\mathcal{F}} \circ op_K$

Here, ∂_F is map-dual, which amounts to De Morgan duality. Hence the forward box operator is obtained from the backward diamond by taking the opposite Kleisli arrow, applying the backward diamond, and then De Morgan duality.

lemma *ffb-prop*: $fb_{\mathcal{F}}\ f = \partial \circ bd_{\mathcal{F}}\ (op_K\ f) \circ \partial$
  by (*simp add*: *ffb-def map-dual-def*)

lemma *ffb-prop-var*: $fb_{\mathcal{F}}\ f = uminus \circ bd_{\mathcal{F}}\ (op_K\ f) \circ uminus$
  by (*simp add*: *dual-set-def ffb-prop*)

lemma *ffb-fbd-dual*: $\partial \circ fb_{\mathcal{F}}\ f = bd_{\mathcal{F}}\ (op_K\ f) \circ \partial$
  by (*simp add*: *ffb-prop o-assoc*)

I give a set-theoretic definition of iffb, because the algebraic one below depends on Inf-preservation.

definition *iffb* :: $('b\ set \Rightarrow 'a\ set) \Rightarrow 'a \Rightarrow 'b\ set$ (‹$fb^-{}_{\mathcal{F}}$›) **where**
  $fb^-{}_{\mathcal{F}}\ \varphi = (\lambda x.\ \bigcap\{X.\ x \in \varphi\ X\})$

lemma *ffb-set*: $fb_{\mathcal{F}}\ f = (\lambda Y.\ \{x.\ f\, x \subseteq Y\})$
  by (*force simp*: *fun-eq-iff ffb-prop-var kop-def klift-def f2r-def r2f-def*)

Forward boxes and backward diamonds are adjoints.

lemma *ffb-fbd-galois*: $(bd_{\mathcal{F}}\ f) \dashv (fb_{\mathcal{F}}\ f)$
  unfolding *adj-def ffb-set klift-prop* **by** *blast*

lemma *iffb-inv1*: $fb^-{}_{\mathcal{F}} \circ fb_{\mathcal{F}} = id$
  unfolding *fun-eq-iff ffb-set iffb-def* **by** *force*

lemma *iffb-inv2-aux*: *Inf-pres* $\varphi \Longrightarrow \bigsqcap\{X.\ x \in \varphi\ X\} \subseteq Y \Longrightarrow x \in \varphi\ Y$
proof−
  assume *Inf-pres* φ
    and *h1*: $\bigsqcap\{X.\ x \in \varphi\ X\} \subseteq Y$
  hence *h2*: $\forall X.\ \varphi\ (\bigsqcap X) = (\bigcap x \in X.\ \varphi\ x)$
    by (*metis comp-eq-dest*)
  hence $\varphi\ (\bigsqcap\{X.\ x \in \varphi\ X\}) \subseteq \varphi\ Y$
    by (*metis h1 INF-lower2 cInf-eq-minimum mem-Collect-eq order-refl*)
  hence $(\bigsqcap\{\varphi\ X\ |X.\ x \in \varphi\ X\}) \subseteq \varphi\ Y$
    by (*metis h2 setcompr-eq-image*)
  thus *?thesis*
    by (*force simp add*: *subset-iff*)
qed

lemma *iffb-inv2*: *Inf-pres* $\varphi \Longrightarrow (fb_{\mathcal{F}} \circ fb^-{}_{\mathcal{F}})\ \varphi = \varphi$
proof−
  assume *h*: *Inf-pres* φ
  {fix *Y*
  have $(fb_{\mathcal{F}} \circ fb^-{}_{\mathcal{F}})\ \varphi\ Y = \{x.\ \bigsqcap\{X.\ x \in \varphi\ X\} \subseteq Y\}$
    by (*simp add*: *ffb-set iffb-def*)
  hence $\bigwedge x.\ x \in (fb_{\mathcal{F}} \circ fb^-{}_{\mathcal{F}})\ \varphi\ Y \longleftrightarrow \bigsqcap\{X.\ x \in \varphi\ X\} \subseteq Y$
    by *auto*

hence $\bigwedge x.\ x \in (fb_{\mathcal{F}} \circ fb^{-}{}_{\mathcal{F}})\ \varphi\ Y \longleftrightarrow x \in \varphi\ Y$
  by (*auto simp*: *h iffb-inv2-aux*)
  hence $(fb_{\mathcal{F}} \circ fb^{-}{}_{\mathcal{F}})\ \varphi\ Y = \varphi\ Y$
  by (*simp add*: *fun-eq-iff set-eq-iff*)**}**
  thus *?thesis*
  unfolding *fun-eq-iff* **by** *simp*
qed

lemma *iffb-inv2-inv*: $(fb_{\mathcal{F}} \circ fb^{-}{}_{\mathcal{F}})\ \varphi = \varphi \Longrightarrow$ *Inf-pres* φ
  by (*auto simp*: *fun-eq-iff ffb-set iffb-def*)

lemma *iffb-inv2-iff*: $((fb_{\mathcal{F}} \circ fb^{-}{}_{\mathcal{F}})\ \varphi = \varphi) = ($*Inf-pres* $\varphi)$
  using *iffb-inv2 iffb-inv2-inv* **by** *blast*

lemma *ffb-inj*: *inj* $fb_{\mathcal{F}}$
  unfolding *inj-def* **by** (*metis iffb-inv1 pointfree-idE*)

lemma *ffb-inj-iff*: $(fb_{\mathcal{F}}\ f = fb_{\mathcal{F}}\ g) = (f = g)$
  by (*simp add*: *ffb-inj inj-eq*)

lemma *ffb-iffb-galois*: *Inf-pres* $\varphi \Longrightarrow (fb^{-}{}_{\mathcal{F}}\ \varphi = f) = (fb_{\mathcal{F}}\ f = \varphi)$
  using *ffb-inj-iff iffb-inv2* **by** *force*

lemma *iffb-inj*: *Inf-pres* $\varphi \Longrightarrow$ *Inf-pres* $\psi \Longrightarrow fb^{-}{}_{\mathcal{F}}\ \varphi = fb^{-}{}_{\mathcal{F}}\ \psi \Longrightarrow \varphi = \psi$
  by (*metis ffb-iffb-galois*)

lemma *iffb-inj-iff*: *Inf-pres* $\varphi \Longrightarrow$ *Inf-pres* $\psi \Longrightarrow (fb^{-}{}_{\mathcal{F}}\ \varphi = fb^{-}{}_{\mathcal{F}}\ \psi) = (\varphi = \psi)$
  using *iffb-inj* **by** *blast*

lemma *ffb-surj*: *Inf-pres* $\varphi \Longrightarrow (\exists f.\ fb_{\mathcal{F}}\ f = \varphi)$
  using *iffb-inv2* **by** *auto*

lemma *iffb-surj*: *surj* $fb^{-}{}_{\mathcal{F}}$
  using *surj-def* **by** (*metis comp-apply iffb-inv1 surj-id*)

This is now the explicit "definition" of iffb, for Inf-preserving transformers.

lemma *iffb-ifbd-dual*: *Inf-pres* $\varphi \Longrightarrow fb^{-}{}_{\mathcal{F}}\ \varphi = (op_K \circ bd^{-}{}_{\mathcal{F}} \circ \partial_F)\ \varphi$
proof−
  assume *h*: *Inf-pres* φ
  {fix *f*
  have $(fb^{-}{}_{\mathcal{F}}\ \varphi = f) = ((\partial_F \circ bd_{\mathcal{F}} \circ op_K)\ f = \varphi)$
  by (*simp add*: *ffb-def ffb-iffb-galois h*)
  also have ... $= (op_K\ f = (bd^{-}{}_{\mathcal{F}} \circ \partial_F)\ \varphi)$
  by (*metis* (*mono-tags*, *lifting*) *comp-apply map-dual-dual ffb-def ffb-surj h klift-eta-inv1 map-dual-dual*)
  finally have $(fb^{-}{}_{\mathcal{F}}\ \varphi = f) = (f = (op_K \circ bd^{-}{}_{\mathcal{F}} \circ \partial_F)\ \varphi)$
  using *kop-galois* **by** *auto***}**
  thus *?thesis*
  by *blast*

qed

lemma *fbd-ffb-dual*: $\partial_F \circ fb_{\mathcal{F}} \circ op_K = bd_{\mathcal{F}}$
proof−
  have $\partial_F \circ fb_{\mathcal{F}} \circ op_K = \partial_F \circ \partial_F \circ bd_{\mathcal{F}} \circ (op_K \circ op_K)$
    by (*simp add*: *comp-def ffb-def*)
  thus *?thesis*
    by *simp*
qed

lemma *ffbd-ffb-dual-var*: $\partial \circ bd_{\mathcal{F}}\ f = fb_{\mathcal{F}}\ (op_K\ f) \circ \partial$
  by (*metis ffb-prop fun-dual1 kop-galois*)

lemma *ifbd-iffb-dual*: *Sup-pres* $\varphi \Longrightarrow bd^-{}_{\mathcal{F}}\ \varphi = (op_K \circ fb^-{}_{\mathcal{F}} \circ \partial_F)\ \varphi$
proof−
  assume *h*: *Sup-pres* φ
  hence *Inf-pres* $(\partial_F\ \varphi)$
    using *Sup-pres-Inf-pres* **by** *blast*
  hence $(op_K \circ fb^-{}_{\mathcal{F}} \circ \partial_F)\ \varphi = (op_K \circ (op_K \circ bd^-{}_{\mathcal{F}} \circ \partial_F) \circ \partial_F)\ \varphi$
    by (*simp add*: *iffb-ifbd-dual*)
  thus *?thesis*
    by (*metis comp-def kop-galois map-dual-dual*)
qed

lemma *ffb-kcomp-pres*: $fb_{\mathcal{F}}\ (f \circ_K g) = fb_{\mathcal{F}}\ f \circ fb_{\mathcal{F}}\ g$
proof−
  have $fb_{\mathcal{F}}\ (f \circ_K g) = \partial_F\ (bd_{\mathcal{F}}\ (op_K\ (f \circ_K g)))$
    by (*simp add*: *ffb-def*)
  also have ... $= \partial_F\ (bd_{\mathcal{F}}\ (op_K\ g \circ_K op_K\ f))$
    by (*simp add*: *kop-contrav*)
  also have ... $= \partial_F\ (bd_{\mathcal{F}}\ (op_K\ f) \circ bd_{\mathcal{F}}\ (op_K\ g))$
    by (*simp add*: *fbd-comp-pres*)
  also have ... $= \partial_F\ (bd_{\mathcal{F}}\ (op_K\ f)) \circ \partial_F\ (bd_{\mathcal{F}}\ (op_K\ g))$
    by (*simp add*: *map-dual-func1*)
  finally show *?thesis*
    by (*simp add*: *ffb-def*)
qed

lemma *ffb-eta-pres*: $fb_{\mathcal{F}}\ \eta = id$
  unfolding *ffb-def* **by** *simp*

lemma *ffb-Sup-dual*: *Sup-dual* $fb_{\mathcal{F}}$
  unfolding *ffb-prop-var comp-def fun-eq-iff klift-prop kop-def f2r-def r2f-def converse-def* **by** *fastforce*

lemma *ffb-Sup-dual-var*: $fb_{\mathcal{F}}\ (\bigsqcup F) = (\bigsqcap f \in F.\ fb_{\mathcal{F}}\ f)$
  unfolding *ffb-prop-var comp-def fun-eq-iff klift-prop kop-def f2r-def r2f-def converse-def* **by** *fastforce*

lemma *ffb-sup-dual*: *sup-dual* $fb_{\mathcal{F}}$
using *ffb-Sup-dual Sup-sup-dual* **by** *force*

lemma *ffb-zero-dual*: $fb_{\mathcal{F}}\ \zeta = (\lambda X.\ UNIV)$
unfolding *ffb-prop-var kop-def klift-prop fun-eq-iff f2r-def r2f-def* **by** *simp*

lemma *inf-dual ffb*
oops

Once again, only the Sup-quantale structure is preserved.

lemma *iffb-comp-pres*:
assumes *Inf-pres* φ
assumes *Inf-pres* ψ
shows $fb^-{}_{\mathcal{F}}\ (\varphi \circ \psi) = fb^-{}_{\mathcal{F}}\ \varphi \circ_K fb^-{}_{\mathcal{F}}\ \psi$
by (*metis assms Inf-pres-comp ffb-iffb-galois ffb-kcomp-pres*)

lemma *iffb-id-pres*: $fb^-{}_{\mathcal{F}}\ id = \eta$
unfolding *iffb-def* **by** *force*

lemma *iffb-Inf-dual*:
assumes $\forall \varphi \in \Phi.$ *Inf-pres* φ
shows $(fb^-{}_{\mathcal{F}} \circ Inf)\ \Phi = (Sup \circ \mathcal{P}\ fb^-{}_{\mathcal{F}})\ \Phi$
proof−
have *Inf-pres* $(\bigsqcap \Phi)$
using *Inf-pres-Inf assms* **by** *blast*
hence $(fb_{\mathcal{F}} \circ fb^-{}_{\mathcal{F}})\ (\bigsqcap \Phi) = \bigsqcap (\mathcal{P}\ (fb_{\mathcal{F}} \circ fb^-{}_{\mathcal{F}})\ \Phi)$
by (*metis* (*mono-tags*, *lifting*) *INF-cong INF-identity-eq assms iffb-inv2*)
hence $(fb_{\mathcal{F}} \circ fb^-{}_{\mathcal{F}})\ (\bigsqcap \Phi) = fb_{\mathcal{F}}\ (\bigsqcup (\mathcal{P}\ fb^-{}_{\mathcal{F}}\ \Phi))$
by (*simp add*: *Setcompr-eq-image ffb-Sup-dual-var image-comp*)
thus *?thesis*
by (*simp add*: *ffb-inj-iff*)
qed

lemma *iffb-Sup-dual*: *Sup-dual* $fb^-{}_{\mathcal{F}}$
by (*auto simp*: *iffb-def fun-eq-iff*)

lemma *iffb-inf-dual*:
assumes *Inf-pres* φ
and *Inf-pres* ψ
shows $fb^-{}_{\mathcal{F}}\ (\varphi \sqcap \psi) = fb^-{}_{\mathcal{F}}\ \varphi \sqcup fb^-{}_{\mathcal{F}}\ \psi$
proof −
have *f1*: $\varphi \sqcap \psi = fb_{\mathcal{F}}\ (fb^-{}_{\mathcal{F}}\ \varphi) \sqcap fb_{\mathcal{F}}\ (fb^-{}_{\mathcal{F}}\ \psi)$
using *assms iffb-inv2* **by** *fastforce*
have $\varphi \sqcap \psi \circ Inter = Inter \circ \mathcal{P}\ (\varphi \sqcap \psi)$
using *assms Inf-pres-inf* **by** *blast*
thus *?thesis*
by (*simp add*: *f1 ffb-iffb-galois ffb-sup-dual*)
qed

lemma *iffb-sup-dual*: $fb^-_{\mathcal{F}}\ (\varphi \sqcup \psi) = fb^-_{\mathcal{F}}\ \varphi \sqcap fb^-_{\mathcal{F}}\ \psi$
unfolding *iffb-def* **by** *fastforce*

lemma *iffb-top-pres* [*simp*]: $fb^-_{\mathcal{F}}\ \top = \zeta$
unfolding *iffb-def* **by** *simp*

This establishes the duality between state transformers and weakest liberal preconditions.

5.4 Forward Box Operators from Relations

Once again one can compose isomorphisms, linking weakest liberal preconditions with relational semantics. The isomorphism obtained should by now be obvious.

definition *rfb* :: $('a \times 'b)\ set \Rightarrow 'b\ set \Rightarrow 'a\ set$ ($\langle fb_{\mathcal{R}} \rangle$) **where**
$fb_{\mathcal{R}} = fb_{\mathcal{F}} \circ \mathcal{F}$

definition *irfb* :: $('b\ set \Rightarrow 'a\ set) \Rightarrow ('a \times 'b)\ set$ ($\langle fb^-_{\mathcal{R}} \rangle$) **where**
$fb^-_{\mathcal{R}} = \mathcal{R} \circ fb^-_{\mathcal{F}}$

lemma *rfb-rbd-dual*: $fb_{\mathcal{R}}\ R = \partial_F\ (bd_{\mathcal{R}}\ (R^{-1}))$
by (*simp add*: *rfb-def rbd-def kop-def ffb-def*, *metis r2f-f2r-galois*)

lemma *rbd-rfb-dual*: $bd_{\mathcal{R}}\ R = \partial_F\ (fb_{\mathcal{R}}\ (R^{-1}))$
by (*simp add*: *rfb-def rbd-def kop-def ffb-def*, *metis converse-converse map-dual-dual r2f-f2r-galois*)

lemma *irfb-irbd-dual*: $Inf\text{-}pres\ \varphi \Longrightarrow fb^-_{\mathcal{R}}\ \varphi = ((\smile) \circ bd^-_{\mathcal{R}} \circ \partial_F)\ \varphi$
by (*simp add*: *irfb-def irbd-def iffb-ifbd-dual kop-def r2f-f2r-galois*)

lemma *irbd-irfb-dual*: $Sup\text{-}pres\ \varphi \Longrightarrow bd^-_{\mathcal{R}}\ \varphi = ((\smile) \circ fb^-_{\mathcal{R}} \circ \partial_F)\ \varphi$
by (*simp add*: *irfb-def irbd-def ifbd-iffb-dual kop-def r2f-f2r-galois*)

lemma *rfb-set*: $fb_{\mathcal{R}}\ R\ Y = \{x.\ \forall\, y.\ (x,y) \in R \longrightarrow y \in Y\}$
unfolding *rfb-def ffb-prop-var comp-def klift-def f2r-def r2f-def kop-def* **by** *force*

lemma *rfb-rbd-galois*: $(bd_{\mathcal{R}}\ R) \dashv (fb_{\mathcal{R}}\ R)$
by (*simp add*: *ffb-fbd-galois rbd-def rfb-def*)

lemma *irfb-set*: $fb^-_{\mathcal{R}}\ \varphi = \{(x,\ y).\ \forall\, Y.\ x \in \varphi\ Y \longrightarrow y \in Y\}$
by (*simp add*: *irfb-def iffb-def f2r-def*)

lemma *irfb-inv1* [*simp*]: $fb^-_{\mathcal{R}} \circ fb_{\mathcal{R}} = id$
by (*simp add*: *fun-eq-iff rfb-def irfb-def iffb-inv1 pointfree-idE*)

lemma *irfb-inv2*: $Inf\text{-}pres\ \varphi \Longrightarrow (fb_{\mathcal{R}} \circ fb^-_{\mathcal{R}})\ \varphi = \varphi$
by (*simp add*: *rfb-def irfb-def*, *metis ffb-iffb-galois r2f-f2r-galois*)

lemma *rfb-inj*: *inj* $fb_{\mathcal{R}}$
  by (*simp add*: *rfb-def ffb-inj inj-compose r2f-inj*)

lemma *rfb-inj-iff*: $(fb_{\mathcal{R}}\ R = fb_{\mathcal{R}}\ S) = (R = S)$
  by (*simp add*: *rfb-inj inj-eq*)

lemma *irfb-inj*: *Inf-pres* $\varphi \Longrightarrow$ *Inf-pres* $\psi \Longrightarrow fb^{-}{}_{\mathcal{R}}\ \varphi = fb^{-}{}_{\mathcal{R}}\ \psi \Longrightarrow \varphi = \psi$
  unfolding *irfb-def* **using** *iffb-inj r2f-inj-iff* **by** *fastforce*

lemma *irfb-inf-iff*: *Inf-pres* $\varphi \Longrightarrow$ *Inf-pres* $\psi \Longrightarrow (fb^{-}{}_{\mathcal{R}}\ \varphi = fb^{-}{}_{\mathcal{R}}\ \psi) = (\varphi = \psi)$
  using *irfb-inj* **by** *auto*

lemma *rfb-surj*: *Inf-pres* $\varphi \Longrightarrow (\exists\, R.\ fb_{\mathcal{R}}\ R = \varphi)$
  using *irfb-inv2* **by** *fastforce*

lemma *irfb-surj*: *surj* $fb^{-}{}_{\mathcal{R}}$
  by (*simp add*: *irfb-def comp-surj f2r-surj iffb-surj cong del*: *image-cong-simp*)

lemma *rfb-irfb-galois*: *Inf-pres* $\varphi \Longrightarrow (fb^{-}{}_{\mathcal{R}}\ \varphi = R) = (fb_{\mathcal{R}}\ R = \varphi)$
  by (*simp add*: *irfb-def rfb-def*, *metis ffb-iffb-galois r2f-f2r-galois*)

lemma *rfb-comp-pres*: $fb_{\mathcal{R}}\ (R\ ;\ S) = fb_{\mathcal{R}}\ R \circ fb_{\mathcal{R}}\ S$
  by (*simp add*: *ffb-kcomp-pres r2f-comp-pres rfb-def*)

lemma *rfb-Id-pres* [*simp*]: $fb_{\mathcal{R}}\ Id = id$
  unfolding *rfb-def ffb-prop* **by** *force*

lemma *rfb-Sup-dual*: *Sup-dual* $fb_{\mathcal{R}}$
proof−
  have $fb_{\mathcal{R}} \circ \mu = fb_{\mathcal{F}} \circ \mathcal{F} \circ Sup$
    by (*simp add*: *rfb-def*)
  also have ... $= fb_{\mathcal{F}} \circ Sup \circ \mathcal{P}\ \mathcal{F}$
    by (*metis fun.map-comp r2f-Sup-pres*)
  also have ... $= Inf \circ \mathcal{P}\ fb_{\mathcal{F}} \circ \mathcal{P}\ \mathcal{F}$
    by (*simp add*: *ffb-Sup-dual*)
  also have ... $= Inf \circ \mathcal{P}\ (fb_{\mathcal{F}} \circ \mathcal{F})$
    by (*simp add*: *P-func1 comp-assoc*)
  finally show *?thesis*
    by (*simp add*: *rfb-def*)
qed

lemma *rfb-Sup-dual-var*: $fb_{\mathcal{R}}\ (\bigsqcup \varphi) = \bigsqcap (\mathcal{P}\ fb_{\mathcal{R}})\ \varphi$
  by (*meson comp-eq-dest rfb-Sup-dual*)

lemma *rfb-sup-dual*: *sup-dual* $fb_{\mathcal{R}}$
  by (*simp add*: *rfb-def ffb-sup-dual r2f-sup-pres*)

lemma *inf-dual* $fb_{\mathcal{R}}$
  oops

lemma *rfb-Inf-pres*: *Inf-pres* ($fb_{\mathcal{R}}$ *R*)
unfolding *rfb-def ffb-prop-var comp-def fun-eq-iff klift-def kop-def f2r-def r2f-def converse-def* **by** *auto*

lemma *rfb-inf-pres*: *inf-pres* ($fb_{\mathcal{R}}$ *R*)
unfolding *rfb-def ffb-prop-var comp-def fun-eq-iff klift-def kop-def f2r-def r2f-def converse-def* **by** *auto*

lemma *rfb-zero-pres* [*simp*]: $fb_{\mathcal{R}}$ {} *X* = *UNIV*
unfolding *rfb-def ffb-prop-var comp-def fun-eq-iff klift-def kop-def f2r-def r2f-def converse-def* **by** *auto*

lemma *rfb-zero-pres2* [*simp*]: $fb_{\mathcal{R}}$ *R* {} = − *Domain R*
unfolding *rfb-def ffb-prop-var comp-def fun-eq-iff klift-def kop-def f2r-def r2f-def converse-def* **by** *auto*

lemma *rfb-univ* [*simp*]: $fb_{\mathcal{R}}$ *R UNIV* = *UNIV*
unfolding *rfb-def ffb-prop-var comp-def fun-eq-iff klift-def kop-def f2r-def r2f-def converse-def* **by** *auto*

lemma *rfb-iso*: $X \subseteq Y \Longrightarrow fb_{\mathcal{R}}\ R\ X \subseteq fb_{\mathcal{R}}\ R\ Y$
unfolding *rfb-def ffb-prop-var comp-def fun-eq-iff klift-def kop-def f2r-def r2f-def converse-def* **by** *auto*

lemma *irfb-comp-pres*:
assumes *Inf-pres* φ
assumes *Inf-pres* ψ
shows $fb^{-}{}_{\mathcal{R}}\ (\varphi \circ \psi) = fb^{-}{}_{\mathcal{R}}\ \varphi\ ;\ fb^{-}{}_{\mathcal{R}}\ \psi$
by (*metis assms rfb-Inf-pres rfb-comp-pres rfb-irfb-galois*)

lemma *irfb-id-pres* [*simp*]: $fb^{-}{}_{\mathcal{R}}$ *id* = *Id*
by (*simp add*: *rfb-irfb-galois*)

lemma *irfb-Sup-dual*: *Sup-dual* $fb^{-}{}_{\mathcal{R}}$
by (*auto simp*: *fun-eq-iff irfb-def iffb-def f2r-def*)

lemma *irfb-Inf-dual*:
assumes $\forall \varphi \in \Phi$. *Inf-pres* φ
shows $(fb^{-}{}_{\mathcal{R}} \circ Inf)\ \Phi = (Sup \circ \mathcal{P}\ fb^{-}{}_{\mathcal{R}})\ \Phi$
proof−
have *Inf-pres* ($\bigsqcap \Phi$)
using *Inf-pres-Inf assms* **by** *blast*
hence $(fb_{\mathcal{R}} \circ fb^{-}{}_{\mathcal{R}})\ (\bigsqcap \Phi) = \bigsqcap (\mathcal{P}\ (fb_{\mathcal{R}} \circ fb^{-}{}_{\mathcal{R}})\ \Phi)$
by (*smt INF-identity-eq Sup.SUP-cong assms irfb-inv2*)
also have ... $= \bigsqcap (\mathcal{P}\ fb_{\mathcal{R}}\ (\mathcal{P}\ fb^{-}{}_{\mathcal{R}}\ \Phi))$
by (*simp add*: *image-comp*)
also have ... $= fb_{\mathcal{R}}\ (\bigsqcup (\mathcal{P}\ fb^{-}{}_{\mathcal{R}}\ \Phi))$
by (*simp add*: *rfb-Sup-dual-var*)

finally have $(fb_{\mathcal{R}} \circ fb^{-}{}_{\mathcal{R}})\ (\bigsqcap \Phi) = fb_{\mathcal{R}}\ (\bigsqcup (\mathcal{P}\ fb^{-}{}_{\mathcal{R}}\ \Phi))$.
thus *?thesis*
by (*simp add*: *rfb-inj-iff*)
qed

lemma *irfb-sup-dual*: *sup-dual* $fb^{-}{}_{\mathcal{R}}$
by (*force simp*: *fun-eq-iff irfb-def iffb-def f2r-def*)

lemma *irfb-inf-dual*:
assumes *Inf-pres* φ
and *Inf-pres* ψ
shows $fb^{-}{}_{\mathcal{R}}\ (\varphi \sqcap \psi) = fb^{-}{}_{\mathcal{R}}\ \varphi \sqcup fb^{-}{}_{\mathcal{R}}\ \psi$
by (*metis assms rfb-Inf-pres rfb-irfb-galois rfb-sup-dual*)

lemma *irfb-top-pres* [*simp*]: $bd^{-}{}_{\mathcal{R}}\ \top = \mathit{UNIV}$
unfolding *irbd-def f2r-def* **by** *auto*

Finally, the adjunctions between the predicate transformers considered so far are revisited.

lemma *ffb-fbd-galois-var*: $(bd_{\mathcal{F}}\ f\ X \subseteq Y) = (X \subseteq fb_{\mathcal{F}}\ f\ Y)$
by (*meson adj-def ffb-fbd-galois*)

lemma *rfb-rbd-galois-var*: $(bd_{\mathcal{R}}\ R\ X \subseteq Y) = (X \subseteq fb_{\mathcal{R}}\ R\ Y)$
by (*meson adj-def rfb-rbd-galois*)

lemma *ffb-fbd*: $fb_{\mathcal{F}}\ f\ Y = \bigcup\{X.\ bd_{\mathcal{F}}\ f\ X \subseteq Y\}$
using *ffb-fbd-galois-var* **by** *fastforce*

lemma *rfb-rbd*: $fb_{\mathcal{R}}\ R\ Y = \bigcup\{X.\ bd_{\mathcal{R}}\ R\ X \subseteq Y\}$
using *rfb-rbd-galois-var* **by** *fastforce*

lemma *fbd-ffb*: $bd_{\mathcal{F}}\ f\ X = \bigcap\{Y.\ X \subseteq fb_{\mathcal{F}}\ f\ Y\}$
using *ffb-fbd-galois-var* **by** *fastforce*

lemma *rbd-rfb*: $bd_{\mathcal{R}}\ R\ X = \bigcap\{Y.\ X \subseteq fb_{\mathcal{R}}\ R\ Y\}$
using *rfb-rbd-galois-var* **by** *fastforce*

5.5 The Remaining Modalities

Finally I set up the remaining dual transformers: forward diamonds and backward boxes. Most properties are not repeated, only some symmetries and dualities are spelled out.

First, forward diamond operators are introduced, from state transformers and relations; together with their inverses.

definition *ffd* :: $('a \Rightarrow 'b\ set) \Rightarrow 'b\ set \Rightarrow 'a\ set$ ($\langle fd_{\mathcal{F}} \rangle$) **where**
$fd_{\mathcal{F}} = bd_{\mathcal{F}} \circ op_K$

definition *iffd* :: $('b\ set \Rightarrow 'a\ set) \Rightarrow 'a \Rightarrow 'b\ set$ (‹$fd^{-}{}_{\mathcal{F}}$›) **where**
$fd^{-}{}_{\mathcal{F}} = op_K \circ bd^{-}{}_{\mathcal{F}}$

definition *rfd* :: $('a \times 'b)\ set \Rightarrow 'b\ set \Rightarrow 'a\ set$ (‹$fd_{\mathcal{R}}$›) **where**
$fd_{\mathcal{R}} = fd_{\mathcal{F}} \circ \mathcal{F}$

definition *irfd* :: $('b\ set \Rightarrow 'a\ set) \Rightarrow ('a \times 'b)\ set$ (‹$fd^{-}{}_{\mathcal{R}}$›) **where**
$fd^{-}{}_{\mathcal{R}} = \mathcal{R} \circ fd^{-}{}_{\mathcal{F}}$

Second, I introduce forward boxes and their inverses.

definition *fbb* :: $('a \Rightarrow 'b\ set) \Rightarrow 'a\ set \Rightarrow 'b\ set$ (‹$bb_{\mathcal{F}}$›) **where**
$bb_{\mathcal{F}} = fb_{\mathcal{F}} \circ op_K$

definition *ifbb* :: $('a\ \ set \Rightarrow 'b\ set) \Rightarrow 'a \Rightarrow 'b\ set$ (‹$bb^{-}{}_{\mathcal{F}}$›) **where**
$bb^{-}{}_{\mathcal{F}} = op_K \circ fb^{-}{}_{\mathcal{F}}$

definition *rbb* :: $('a \times 'b)\ set \Rightarrow 'a\ set \Rightarrow 'b\ set$ (‹$bb_{\mathcal{R}}$›) **where**
$bb_{\mathcal{R}} = bb_{\mathcal{F}} \circ \mathcal{F}$

definition *irbb* :: $('a\ \ set \Rightarrow 'b\ set) \Rightarrow ('a \times 'b)\ set$ (‹$bb^{-}{}_{\mathcal{R}}$›) **where**
$bb^{-}{}_{\mathcal{R}} = \mathcal{R} \circ bb^{-}{}_{\mathcal{F}}$

Forward and backward operators of the same type (box or diamond) are related by opposition.

lemma *rfd-rbd*: $fd_{\mathcal{R}} = bd_{\mathcal{R}} \circ (\smile)$
by (*simp add*: *rfd-def rbd-def ffd-def kop-def comp-assoc*)

lemma *irfd-irbd*: $fd^{-}{}_{\mathcal{R}} = (\smile) \circ bd^{-}{}_{\mathcal{R}}$
by (*simp add*: *irfd-def iffd-def kop-def irbd-def comp-assoc*[*symmetric*])

lemma *fbd-ffd*: $bd_{\mathcal{F}} = fd_{\mathcal{F}} \circ op_K$
by (*simp add*: *ffd-def kop-def converse-def f2r-def r2f-def klift-def fun-eq-iff*)

lemma *rbb-rfb*: $bb_{\mathcal{R}} = fb_{\mathcal{R}} \circ (\smile)$
by (*simp add*: *rfb-def rbb-def*, *metis fbb-def kop-def r2f-f2r-galois-var2 rewriteR-comp-comp2*)

lemma *irbb-irfb*: $bb^{-}{}_{\mathcal{R}} = (\smile) \circ fb^{-}{}_{\mathcal{R}}$
proof−
have $bb^{-}{}_{\mathcal{R}} = \mathcal{R} \circ op_K \circ fb^{-}{}_{\mathcal{F}}$
by (*simp add*: *irbb-def ifbb-def o-assoc*)
also have ... $= \mathcal{R} \circ \mathcal{F} \circ (\smile) \circ \mathcal{R} \circ fb^{-}{}_{\mathcal{F}}$
by (*simp add*: *kop-def o-assoc*)
also have ... $= (\smile) \circ fb^{-}{}_{\mathcal{R}}$
by (*simp add*: *comp-assoc irfb-def*)
finally show *?thesis*.
qed

Complementation is a natural isomorphism between forwards and backward operators of different type.

lemma *ffd-ffb-demorgan*: $\partial \circ fd_{\mathcal{F}}\ f = fb_{\mathcal{F}}\ f \circ \partial$
by (*simp add*: *comp-assoc ffb-prop ffd-def*)

lemma *iffd-iffb-demorgan*: *Sup-pres* $\varphi \Longrightarrow fd^{-}{}_{\mathcal{F}}\ \varphi = (fb^{-}{}_{\mathcal{F}} \circ \partial_F)\ \varphi$
by (*smt Sup-pres-Inf-pres comp-apply iffb-ifbd-dual iffd-def map-dual-dual*)

lemma *ffb-ffd-demorgan*: $\partial \circ fb_{\mathcal{F}}\ f = fd_{\mathcal{F}}\ f \circ \partial$
by (*simp add*: *ffb-prop ffd-def rewriteL-comp-comp*)

lemma *iffb-iffd-demorgan*: *Inf-pres* $\varphi \Longrightarrow fb^{-}{}_{\mathcal{F}}\ \varphi = (fd^{-}{}_{\mathcal{F}} \circ \partial_F)\ \varphi$
by (*simp add*: *iffb-ifbd-dual iffd-def*)

lemma *rfd-rfb-demorgan*: $\partial \circ fd_{\mathcal{R}}\ R = fb_{\mathcal{R}}\ R \circ \partial$
by (*simp add*: *rfb-def rfd-def ffd-ffb-demorgan*)

lemma *irfd-irfb-demorgan*: *Sup-pres* $\varphi \Longrightarrow fd^{-}{}_{\mathcal{R}}\ \varphi = (fb^{-}{}_{\mathcal{R}} \circ \partial_F)\ \varphi$
by (*simp add*: *irfb-def irfd-def iffd-iffb-demorgan*)

lemma *rfb-rfd-demorgan*: $\partial \circ fb_{\mathcal{R}}\ R = fd_{\mathcal{R}}\ R \circ \partial$
by (*simp add*: *ffb-ffd-demorgan rfb-def rfd-def*)

lemma *irfb-irfd-demorgan*: *Inf-pres* $\varphi \Longrightarrow fb^{-}{}_{\mathcal{R}}\ \varphi = (fd^{-}{}_{\mathcal{R}} \circ \partial_F)\ \varphi$
by (*simp add*: *irfb-irbd-dual irfd-irbd*)

lemma *fbd-fbb-demorgan*: $\partial \circ bd_{\mathcal{F}}\ f = bb_{\mathcal{F}}\ f \circ \partial$
by (*simp add*: *fbb-def fbd-ffd ffd-ffb-demorgan*)

lemma *ifbd-ifbb-demorgan*: *Sup-pres* $\varphi \Longrightarrow bd^{-}{}_{\mathcal{F}}\ \varphi = (bb^{-}{}_{\mathcal{F}} \circ \partial_F)\ \varphi$
by (*simp add*: *ifbd-iffb-dual ifbb-def*)

lemma *fbb-fbd-demorgan*: $\partial \circ bb_{\mathcal{F}}\ R = bd_{\mathcal{F}}\ R \circ \partial$
by (*simp add*: *fbb-def fbd-ffd ffb-ffd-demorgan*)

lemma *ifbb-ifbd-demorgan*: *Inf-pres* $\varphi \Longrightarrow bb^{-}{}_{\mathcal{F}}\ \varphi = (bd^{-}{}_{\mathcal{F}} \circ \partial_F)\ \varphi$
proof−
assume *h*: *Inf-pres* φ
have $bb^{-}{}_{\mathcal{F}}\ \varphi = (op_K \circ fb^{-}{}_{\mathcal{F}})\ \varphi$
by (*simp add*: *ifbb-def*)
also have ... $= (op_K \circ op_K \circ bd^{-}{}_{\mathcal{F}})\ (\partial_F\ \varphi)$
by (*metis comp-apply h iffb-ifbd-dual*)
also have ... $= (bd^{-}{}_{\mathcal{F}} \circ \partial_F)\ \varphi$
by *auto*
finally show *?thesis*.
qed

lemma *rbd-rbb-demorgan*: $\partial \circ bd_{\mathcal{R}}\ R = bb_{\mathcal{R}}\ R \circ \partial$
by (*simp add*: *rbb-def rbd-def fbd-fbb-demorgan*)

lemma *irbd-irbb-demorgan*: *Sup-pres* $\varphi \Longrightarrow bd^{-}{}_{\mathcal{R}}\ \varphi = (bb^{-}{}_{\mathcal{R}} \circ \partial_F)\ \varphi$

by (*simp add*: *irbb-irfb irbd-irfb-dual*)

lemma *rbb-rbd-demorgan*: $\partial \circ bb_{\mathcal{R}}\ R = bd_{\mathcal{R}}\ R \circ \partial$
by (*simp add*: *rbb-def rbd-def fbb-fbd-demorgan*)

lemma *irbb-irbd-demorgan*: *Inf-pres* $\varphi \Longrightarrow bb^{-}{}_{\mathcal{R}}\ \varphi = (bd^{-}{}_{\mathcal{R}} \circ \partial_F)\ \varphi$
by (*simp add*: *irbb-def irbd-def ifbb-ifbd-demorgan*)

Further symmetries arise by combination.

lemma *ffd-fbb-dual*: $\partial \circ fd_{\mathcal{F}}\ f = bb_{\mathcal{F}}\ (op_K\ f) \circ \partial$
by (*simp add*: *fbd-fbb-demorgan ffd-def*)

lemma *iffd-ifbb-dual*: *Sup-pres* $\varphi \Longrightarrow fd^{-}{}_{\mathcal{F}}\ \varphi = (op_K \circ bb^{-}{}_{\mathcal{F}} \circ \partial_F)\ \varphi$
by (*simp add*: *ifbd-ifbb-demorgan iffd-def*)

lemma *fbb-ffd-dual*: $\partial \circ bb_{\mathcal{F}}\ f = fd_{\mathcal{F}}\ (op_K\ f) \circ \partial$
by (*simp add*: *fbd-ffd fbb-fbd-demorgan*)

lemma *ifbb-iffd-dual*: *Inf-pres* $\varphi \Longrightarrow bb^{-}{}_{\mathcal{F}}\ \varphi = (op_K \circ fd^{-}{}_{\mathcal{F}} \circ \partial_F)\ \varphi$
by (*simp add*: *ifbb-def iffb-iffd-demorgan*)

lemma *rfd-rbb-dual*: $\partial \circ fd_{\mathcal{R}}\ R = bb_{\mathcal{R}}\ (R^{-1}) \circ \partial$
by (*metis fun-dual1 map-dual-def rbd-rbb-demorgan rfb-rbd-dual rfd-rfb-demorgan*)

lemma *ifd-ibb-dual*: *Sup-pres* $\varphi \Longrightarrow fd^{-}{}_{\mathcal{R}}\ \varphi = ((\smile) \circ bb^{-}{}_{\mathcal{R}} \circ \partial_F)\ \varphi$
by (*simp add*: *irbb-irfb irbd-irfb-dual irfd-irbd*)

lemma *rbb-rfd-dual*: $\partial \circ bb_{\mathcal{R}}\ R = fd_{\mathcal{R}}\ (R^{-1}) \circ \partial$
by (*simp add*: *rbb-rfb rfb-rfd-demorgan*)

lemma *irbb-irfd-dual*: *Inf-pres* $\varphi \Longrightarrow bb^{-}{}_{\mathcal{R}}\ \varphi = ((\smile) \circ fd^{-}{}_{\mathcal{R}} \circ \partial_F)\ \varphi$
by (*simp add*: *irbb-irfb irfb-irbd-dual irfd-irbd*)

lemma *ffd-iffd-galois*: *Sup-pres* $\varphi \Longrightarrow (\varphi = fd_{\mathcal{F}}\ f) = (f = fd^{-}{}_{\mathcal{F}}\ \varphi)$
unfolding *ffd-def iffd-def* **by** (*metis comp-apply fbd-surj klift-eta-inv1 kop-galois*)

lemma *rfd-irfd-galois*: *Sup-pres* $\varphi \Longrightarrow (\varphi = fd_{\mathcal{R}}\ R) = (R = fd^{-}{}_{\mathcal{R}}\ \varphi)$
unfolding *irfd-def rfd-def* **by** (*metis comp-apply ffd-iffd-galois r2f-f2r-galois*)

lemma *fbb-ifbb-galois*: *Inf-pres* $\varphi \Longrightarrow (\varphi = bb_{\mathcal{F}}\ f) = (f = bb^{-}{}_{\mathcal{F}}\ \varphi)$
unfolding *fbb-def iffb-def* **by** (*metis* (*no-types*, *lifting*) *comp-apply ffb-iffb-galois ifbb-ifbd-demorgan iffb-ifbd-dual kop-galois*)

lemma *rbb-irbb-galois*: *Inf-pres* $\varphi \Longrightarrow (\varphi = bb_{\mathcal{R}}\ R) = (R = bb^{-}{}_{\mathcal{R}}\ \varphi)$
apply (*simp add*: *rbb-def irbb-def*) **using** *fbb-ifbb-galois r2f-f2r-galois* **by** *blast*

Next I spell out the missing adjunctions.

lemma *ffd-ffb-adj*: $fd_{\mathcal{F}}\ f \dashv bb_{\mathcal{F}}\ f$
by (*simp add*: *fbb-def ffb-fbd-galois ffd-def*)

lemma *ffd-fbb-galois*: $(fd_{\mathcal{F}}\ f\ X \subseteq Y) = (X \subseteq bb_{\mathcal{F}}\ f\ Y)$
  by (*simp add*: *fbb-def ffb-fbd-galois-var ffd-def*)

lemma *rfd-rfb-adj*: $fd_{\mathcal{R}}\ f \dashv bb_{\mathcal{R}}\ f$
  by (*simp add*: *ffd-ffb-adj rbb-def rfd-def*)

lemma *rfd-rbb-galois*: $(fd_{\mathcal{R}}\ R\ X \subseteq Y) = (X \subseteq bb_{\mathcal{R}}\ R\ Y)$
  by (*simp add*: *ffd-fbb-galois rbb-def rfd-def*)

Finally, forward and backward operators of the same type are linked by conjugation.

lemma *ffd-fbd-conjugation*: $(fd_{\mathcal{F}}\ f\ X \cap Y = \{\}) = (X \cap bd_{\mathcal{F}}\ f\ Y = \{\})$
proof−
  have $(fd_{\mathcal{F}}\ f\ X \cap Y = \{\}) = (fd_{\mathcal{F}}\ f\ X \subseteq -Y)$
    by (*simp add*: *disjoint-eq-subset-Compl*)
  also have ... $= (X \subseteq bb_{\mathcal{F}}\ f\ (-Y))$
    by (*simp add*: *ffd-fbb-galois*)
  also have ... $= (X \cap -\ bb_{\mathcal{F}}\ f\ (-Y) = \{\})$
    by (*simp add*: *disjoint-eq-subset-Compl*)
  also have ... $= (X \cap \partial\ (bb_{\mathcal{F}}\ f\ (\partial\ Y)) = \{\})$
    by (*simp add*: *dual-set-def*)
  finally show *?thesis*
   by (*metis* (*no-types*, *opaque-lifting*) *comp-apply fbb-fbd-demorgan invol-dual-var*)
qed

lemma *rfd-rbd-conjugation*: $((fd_{\mathcal{R}}\ R\ X) \cap Y = \{\}) = (X \cap (bd_{\mathcal{R}}\ R\ Y) = \{\})$
  by (*simp add*: *rbd-def rfd-def ffd-fbd-conjugation*)

lemma *ffb-fbb-conjugation*: $((fb_{\mathcal{F}}\ f\ X) \cup Y = UNIV) = (X \cup (bb_{\mathcal{F}}\ f\ Y) = UNIV)$

proof−
  have $((fb_{\mathcal{F}}\ f\ X) \cup Y = UNIV) = (-Y \subseteq fb_{\mathcal{F}}\ f\ X)$
    by *blast*
  also have ... $= (bd_{\mathcal{F}}\ f\ (\partial\ Y) \subseteq X)$
    by (*simp add*: *ffb-fbd-galois-var dual-set-def*)
  also have ... $= (\partial\ (bb_{\mathcal{F}}\ f\ Y) \subseteq X)$
    by (*metis comp-def fbb-fbd-demorgan*)
  also have ... $= (X \cup (bb_{\mathcal{F}}\ f\ Y) = UNIV)$
    by (*metis compl-le-swap2 dual-set-def sup-shunt*)
    finally show *?thesis***.**
qed

lemma *rfb-rbb-conjugation*: $((fb_{\mathcal{R}}\ R\ X) \cup Y = UNIV) = (X \cup (bb_{\mathcal{R}}\ R\ Y) = UNIV)$
  by (*simp add*: *rfb-def rbb-def ffb-fbb-conjugation*)

end

6 The Quantaloid of Kleisli Arrows

theory *Kleisli-Quantaloid*

imports *Kleisli-Transformers*
begin

This component formalises the quantalic structure of Kleisli arrows or state transformers, that is, the homset of the Kleisli category. Of course, by the previous isomorphisms, this is reflected at least partially in the Eilenberg-Moore algebras, via the comparison functor. The main result is that Kleisli arrows form a quantaloid, hence essentially a typed quantale. Some emphasis is on the star. This component thus complements that in which the quantaloid structure of Sup- and Inf-preserving transformers has been formalised.

The first set of lemmas shows that Kleisli arrows form a typed dioid, that is, a typed idempotent semiring.

lemma *ksup-assoc*: $((f{::}'a \Rightarrow 'b\ set) \sqcup g) \sqcup h = f \sqcup (g \sqcup h)$
  unfolding *sup.assoc* **by** *simp*

lemma *ksup-comm*: $(f{::}'a => 'b\ set) \sqcup g = g \sqcup f$
  by (*simp add*: *sup.commute*)

lemma *ksup-idem* [*simp*]: $(f{::}'a \Rightarrow 'b\ set) \sqcup f = f$
  by *simp*

lemma *kcomp-distl*: $f \circ_K (g \sqcup h) = (f \circ_K g) \sqcup (f \circ_K h)$
 unfolding *kcomp-klift fun-eq-iff comp-def sup-fun-def* **by** (*simp add*: *UN-Un-distrib klift-prop*)

lemma *kcomp-distr*: $(f \sqcup g) \circ_K h = (f \circ_K h) \sqcup (g \circ_K h)$
  by (*simp add*: *kcomp-klift fun-eq-iff klift-def*)

lemma *ksup-zerol* [*simp*]: $\zeta \sqcup f = f$
  by *force*

lemma *ksup-annil* [*simp*]: $\zeta \circ_K f = \zeta$
  by (*force simp*: *kcomp-klift klift-def*)

lemma *ksup-annir* [*simp*]: $f \circ_K \zeta = \zeta$
  by (*force simp*: *kcomp-klift klift-def*)

Associativity of Kleisli composition has already been proved.

The next laws establish typed quantales — or quantaloids.

lemma *kSup-distl*: $f \circ_K (\bigsqcup G) = (\bigsqcup g \in G.\ f \circ_K g)$
proof−
  have $f \circ_K (\bigsqcup G) = ((klift \circ Sup)\ G) \circ f$

by (*simp add*: *kcomp-klift*)
also have ... = ($\bigsqcup g \in G.$ (*klift g*)) $\circ f$
by (*simp add*: *fbd-Sup-pres fun-eq-iff*)
also have ... = ($\bigsqcup g \in G.$ (*klift g*) $\circ f$)
by *auto*
finally show *?thesis*
by (*simp add*: *kcomp-klift*)
qed

lemma *kSup-distr*: ($\bigsqcup F$) $\circ_K$ g = ($\bigsqcup f \in F.$ $f \circ_K g$)
unfolding *kcomp-klift fun-eq-iff comp-def* **by** (*simp add*: *klift-prop*)

lemma *kcomp-isol*: $f \leq g \Longrightarrow h \circ_K f \leq h \circ_K g$
by (*force simp*: *kcomp-klift le-fun-def klift-def*)

lemma *kcomp-isor*: $f \leq g \Longrightarrow f \circ_K h \leq g \circ_K h$
by (*force simp*: *kcomp-klift le-fun-def klift-def*)

6.1 Kleene Star

The Kleene star can be defined in any quantale or quantaloid by iteration. For Kleisli arrows, laws for the star can be obtained via the isomorphism to binary relations, where the star is the reflexive-transitive closure operation.

abbreviation *kpower* ≡ *kmon.power*

lemma *r2f-pow*: $\mathcal{F}$ (R ^^ i) = *kpower* ($\mathcal{F}$ R) i
by (*induct i*, *simp*, *metis power.power.power-Suc r2f-comp-pres relpow.simps*(*2*) *relpow-commute*)

lemma *f2r-kpower*: $\mathcal{R}$ (*kpower f i*) = ($\mathcal{R}$ f) ^^ i
by (*induct i*, *simp*, *metis f2r2f-inv2 pointfree-idE r2f2r-inv1 r2f-pow*)

definition *kstar f* = ($\bigsqcup i.$ *kpower f i*)

lemma *r2f-rtrancl-hom*: $\mathcal{F}$ (*rtrancl R*) = *kstar* ($\mathcal{F}$ R)
proof−
have $\mathcal{F}$ (*rtrancl R*) = $\mathcal{F}$ ($\bigcup i.$ R ^^ i)
by (*simp add*: *full-SetCompr-eq rtrancl-is-UN-relpow*)
also have ... = ($\bigsqcup i.$ *kpower* ($\mathcal{F}$ R) i)
by (*auto simp*: *r2f-Sup-pres-var r2f-pow*)
finally show *?thesis*
by (*simp add*: *kstar-def*)
qed

lemma *r2f-rtrancl-hom-var*: $\mathcal{F} \circ$ *rtrancl* = *kstar* $\circ \mathcal{F}$
by *standard* (*simp add*: *r2f-rtrancl-hom*)

lemma *f2r-kstar-hom*: $\mathcal{R}$ (*kstar f*) = *rtrancl* ($\mathcal{R}$ f)
by (*metis r2f-f2r-galois r2f-rtrancl-hom*)

lemma *f2r-kstar-hom-var*: $\mathcal{R} \circ$ *kstar* = *rtrancl* $\circ\ \mathcal{R}$
by *standard* (*simp add*: *f2r-kstar-hom*)

lemma *kstar-unfoldl-eq*: $\eta \sqcup f \circ_K$ *kstar f* = *kstar f*
proof −
have $\mathcal{R}$ (*kstar f*) = ($\mathcal{R}$ η) $\cup$ ($\mathcal{R}$ f)* ; $\mathcal{R}$ f
using *f2r-kstar-hom rtrancl-unfold*
by (*metis f2r-eta-pres*)
thus *?thesis*
by (*metis f2r-kcomp-pres f2r-kstar-hom f2r-sup-pres r2f-inj-iff r-comp-rtrancl-eq*)
qed

lemma *kstar-unfoldl*: $\eta \sqcup f \circ_K$ *kstar f* $\leq$ *kstar f*
by (*simp add*: *kstar-unfoldl-eq*)

lemma *kstar-unfoldr-eq*: $\eta \sqcup$ (*kstar f*) $\circ_K f$ = *kstar f*
by (*metis* (*no-types*) *f2r2f-inv2 f2r-kcomp-pres f2r-kstar-hom kstar-unfoldl-eq pointfree-idE r-comp-rtrancl-eq*)

lemma *kstar-unfoldr*: $\eta \sqcup$ (*kstar f*) $\circ_K f \leq$ *kstar f*
by (*simp add*: *kstar-unfoldr-eq*)

Relational induction laws seem to be missing in Isabelle Main. So I derive functional laws directly.

lemma *kpower-inductl*: $f \circ_K g \leq g \Longrightarrow$ *kpower f i* $\circ_K g \leq g$
by (*induct i*, *simp-all add*: *kcomp-assoc kcomp-isol order-subst2*)

lemma *kpower-inductl-var*: $h \sqcup f \circ_K g \leq g \Longrightarrow$ *kpower f i* $\circ_K h \leq g$
proof −
assume *h1*: $h \sqcup f \circ_K g \leq g$
then have *h2*: $f \circ_K g \leq g$
using *le-sup-iff* **by** *blast*
have $h \leq g$
using *h1* **by** *simp*
then show *?thesis*
using *h2 kcomp-isol kpower-inductl order-trans* **by** *blast*
qed

lemma *kstar-inductl*: $h \sqcup f \circ_K g \leq g \Longrightarrow$ *kstar f* $\circ_K h \leq g$
apply (*simp add*: *kstar-def kSup-distr*, *rule Sup-least*)
using *kpower-inductl-var* **by** *fastforce*

lemma *kpower-inductr*: $g \circ_K f \leq g \Longrightarrow g \circ_K$ *kpower f i* $\leq g$
apply (*induct i*, *simp-all*)
by (*metis* (*mono-tags*, *lifting*) *dual-order.trans kcomp-assoc kcomp-isor*)

lemma *kpower-inductr-var*: $h \sqcup g \circ_K f \leq g \Longrightarrow h \circ_K$ *kpower f i* $\leq g$
by (*metis* (*no-types*) *dual-order.trans kcomp-isor kpower-inductr le-sup-iff*)

lemma *kstar-inductr*: $h \sqcup g \circ_K f \leq g \Longrightarrow h \circ_K \mathit{kstar}\ f \leq g$
apply (*simp add*: *kstar-def kSup-distl*, *rule Sup-least*)
using *kpower-inductr-var* **by** *fastforce*

lemma *kpower-prop*: $f \leq \eta \Longrightarrow \mathit{kpower}\ f\ i \leq \eta$
by (*metis kcomp-idl kpower-inductr*)

lemma *kstar-prop*: $f \leq \eta \Longrightarrow \mathit{kstar}\ f \leq \eta$
by (*simp add*: *SUP-le-iff kpower-prop kstar-def*)

6.2 Antidomain

Next I define an antidomain operation and prove the axioms of antidomain semirings [5, 3].

definition $\mathit{kad}\ f = (\lambda x.\ \mathit{if}\ (f\ x = \{\})\ \mathit{then}\ \{x\}\ \mathit{else}\ \{\})$

definition $\mathit{ad\text{-}rel}\ R = \{(x,x)\ |x.\ \neg(\exists\, y.\ (x,y) \in R)\}$

lemma *f2r-ad-fun-hom*: $\mathcal{R}\ (\mathit{kad}\ f) = \mathit{ad\text{-}rel}\ (\mathcal{R}\ f)$
apply (*simp add*: *kad-def ad-rel-def f2r-def*, *safe*)
by *simp-all* (*meson empty-iff singletonD*)

lemma *f2r-ad-fun-hom-var*: $\mathcal{R} \circ \mathit{kad} = \mathit{ad\text{-}rel} \circ \mathcal{R}$
by *standard* (*simp add*: *f2r-ad-fun-hom*)

lemma *r2f-ad-rel-hom*: $\mathcal{F}\ (\mathit{ad\text{-}rel}\ R) = \mathit{kad}\ (\mathcal{F}\ R)$
by (*force simp add*: *kad-def ad-rel-def r2f-def fun-eq-iff*)

lemma *r2f-ad-rel-hom-var*: $\mathcal{F} \circ \mathit{ad\text{-}rel} = \mathit{kad} \circ \mathcal{F}$
by *standard* (*simp add*: *r2f-ad-rel-hom*)

lemma *ad-fun-as1* [*simp*]: $(\mathit{kad}\ f) \circ_K f = \zeta$
by (*simp add*: *kad-def kcomp-def fun-eq-iff*)

lemma *ad-fun-as2* [*simp*]: $\mathit{kad}\ (f \circ_K g) \sqcup \mathit{kad}\ (f \circ_K \mathit{kad}\ (\mathit{kad}\ g)) = \mathit{kad}\ (f \circ_K \mathit{kad}\ (\mathit{kad}\ g))$
by (*force simp*: *kad-def kcomp-def fun-eq-iff*)

lemma *ad-fun-as3* [*simp*]: $\mathit{kad}\ (\mathit{kad}\ f) \sqcup \mathit{kad}\ f = \eta$
by (*simp add*: *kad-def fun-eq-iff*)

definition $\mathit{set2fun}\ X = (\lambda x.\ \mathit{if}\ (x \in X)\ \mathit{then}\ \{x\}\ \mathit{else}\ \{\})$

definition $\mathit{p2fun} = \mathit{set2fun} \circ \mathit{Collect}$

lemma *ffb-ad-fun*: $fb_{\mathcal{F}}\ f\ X = \{x.\ (\mathit{kad}\ (f \circ_K \mathit{kad}\ (\mathit{set2fun}\ X)))\ x \neq \{\}\}$
unfolding *ffb-prop-var klift-def kop-def fun-eq-iff comp-def f2r-def r2f-def converse-def kad-def kcomp-def set2fun-def*

by *auto*

lemma *ffb-ad-fun2*: *set2fun* ($fb_{\mathcal{F}}$ *f X*) = *kad* (*f* $\circ_K$ *kad* (*set2fun X*))
by *standard* (*subst ffb-ad-fun*, *subst set2fun-def*, *simp add*: *kad-def*)

The final statements check that the relational forward diamond is consistent with the Kleene-algebraic definition.

lemma *fb-ad-rel*: $fb_{\mathcal{R}}$ *R X* = *Domain* (*ad-rel* (*R* ; *ad-rel* (*Id-on X*)))
unfolding *rfb-def ffb-prop-var klift-def comp-def r2f-def kop-def f2r-def converse-def Domain-def Id-on-def ad-rel-def*
by *auto*

lemma *fb-ad-rel2*: *Id-on* ($fb_{\mathcal{R}}$ *R X*) = *ad-rel* (*R* ; *ad-rel* (*Id-on X*))
unfolding *rfb-def ffb-prop-var klift-def comp-def r2f-def kop-def f2r-def converse-def Domain-def Id-on-def ad-rel-def*
by *auto*

end

7 The Quantale of Kleisli Arrows

theory *Kleisli-Quantale*
imports *Kleisli-Quantaloid*
Quantales.Quantale-Star

begin

This component revisits the results of the quantaloid one in the single-typed setting, that is, in the context of quantales. An instance proof, showing that Kleisli arrows (or state transformers) form quantales, is its main result. Facts proved for quantales are thus made available for state transformers.

typedef *'a nd-fun* = {*f*::*'a* ⇒ *'a set*. *f* ∈ *UNIV*}
by *simp*

setup-lifting *type-definition-nd-fun*

Definitions are lifted to gain access to the Kleisli categories.

lift-definition *r2fnd* :: *'a rel* ⇒ *'a nd-fun* **is** *Abs-nd-fun* ∘ $\mathcal{F}$**.**

lift-definition *f2rnd* :: *'a nd-fun* ⇒ *'a rel* **is** $\mathcal{R}$ ∘ *Rep-nd-fun***.**

declare *Rep-nd-fun-inverse* [*simp*]

lemma *r2f2r-inv*: *r2fnd* ∘ *f2rnd* = *id*
by *transfer* (*simp add*: *fun-eq-iff pointfree-idE*)

lemma *f2r2f-inv*: *f2rnd* ∘ *r2fnd* = *id*
by *transfer* (*simp add*: *fun-eq-iff r2f-def f2r-def Abs-nd-fun-inverse*)

instantiation *nd-fun* :: (*type*) *monoid-mult*
begin

lift-definition *one-nd-fun* :: *'a nd-fun* **is** *Abs-nd-fun* η**.**

lift-definition *times-nd-fun* :: *'a*::*type nd-fun* $\Rightarrow$ *'a*::*type nd-fun* $\Rightarrow$ *'a*::*type nd-fun* **is** $\lambda f\ g$. *Abs-nd-fun* (*Rep-nd-fun f* $\circ_K$ *Rep-nd-fun g*)**.**

instance
by *intro-classes* (*transfer*, *simp add*: *Abs-nd-fun-inverse kcomp-assoc*)+

end

instantiation *nd-fun* :: (*type*) *order-lean*
begin

lift-definition *less-eq-nd-fun* :: *'a nd-fun* $\Rightarrow$ *'a nd-fun* $\Rightarrow$ *bool* **is** $\lambda f\ g$. *Rep-nd-fun f* $\leq$ *Rep-nd-fun g***.**

lift-definition *less-nd-fun* :: *'a nd-fun* $\Rightarrow$ *'a nd-fun* $\Rightarrow$ *bool* **is** $\lambda f\ g$. *Rep-nd-fun f* $\leq$ *Rep-nd-fun g* $\wedge f \neq g$**.**

instance
apply *intro-classes*
apply (*transfer*, *simp*)
apply *transfer* **using** *order.trans* **apply** *blast*
by (*simp add*: *Rep-nd-fun-inject less-eq-nd-fun.abs-eq*)

end

instantiation *nd-fun* :: (*type*) *Sup-lattice*
begin

lift-definition *Sup-nd-fun* :: *'a nd-fun set* $\Rightarrow$ *'a nd-fun* **is** *Abs-nd-fun* $\circ$ *Sup* $\circ$ $\mathcal{P}$ *Rep-nd-fun***.**

instance
by (*intro-classes*; *transfer*, *simp-all add*: *Abs-nd-fun-inverse Sup-upper sup-absorb2 Sup-le-iff*)

end

lemma *Abs-comp-hom*: *Abs-nd-fun* ($f \circ_K g$) = *Abs-nd-fun f* $\cdot$ *Abs-nd-fun g*
by *transfer* (*simp add*: *Abs-nd-fun-inverse*)

lemma *Rep-comp-hom*: *Rep-nd-fun* ($f \cdot g$) = *Rep-nd-fun f* $\circ_K$ *Rep-nd-fun g*
by (*simp add*: *Abs-nd-fun-inverse times-nd-fun.abs-eq*)

instance *nd-fun* :: (*type*) *unital-Sup-quantale*
by (*intro-classes*; *transfer*, *simp-all*) (*smt Abs-comp-hom Rep-comp-hom Rep-nd-fun-inverse SUP-cong image-image kSup-distr kSup-distl*)+

Unfortunately, this is not it yet. To benefit from Isabelle's theorems for orderings, lattices, Kleene algebras and quantales, Isabelle's complete lattices need to be in scope. Somewhat annoyingly, this requires more work...

instantiation *nd-fun* :: (*type*) *complete-lattice*
begin

lift-definition *Inf-nd-fun* :: *'a nd-fun set* ⇒ *'a nd-fun* **is** *Abs-nd-fun* ∘ *Inf* ∘ *𝒫 Rep-nd-fun*.

lift-definition *bot-nd-fun* :: *'a::type nd-fun* **is** *Abs-nd-fun* (*Sup* {}).

lift-definition *sup-nd-fun* :: *'a::type nd-fun* ⇒ *'a::type nd-fun* ⇒ *'a::type nd-fun* **is** *λf g. Abs-nd-fun* (*Rep-nd-fun f* ⊔ *Rep-nd-fun g*).

lift-definition *top-nd-fun* :: *'a::type nd-fun* **is** *Abs-nd-fun* (*Inf* {}).

lift-definition *inf-nd-fun* :: *'a::type nd-fun* ⇒ *'a::type nd-fun* ⇒ *'a::type nd-fun* **is** *λf g. Abs-nd-fun* (*Rep-nd-fun f* ⊓ *Rep-nd-fun g*).

instance
apply *intro-classes*
apply *transfer* **using** *Rep-nd-fun-inject dual-order.antisym* **apply** *blast*
apply (*transfer*, *simp*)
apply (*transfer*, *simp*)
apply (*simp add*: *Abs-nd-fun-inverse*)
by (*transfer*; *simp-all add*: *Abs-nd-fun-inverse Sup-le-iff SUP-upper2 le-INF-iff Inf-lower*)+

end

instance *nd-fun* :: (*type*) *unital-quantale*
apply *intro-classes*
using *supq.Sup-distr* **apply** *fastforce*
by (*simp add*: *supq.Sup-distl*)

Now, theorems for the Kleene star, which come from quantales, are finally in scope.

lemma *fun-star-unfoldl-eq*: (*1::'a nd-fun*) ⊔ *f* · *qstar f* = *qstar f*
by (*simp add*: *qstar-comm*)

lemma *fun-star-unfoldl*: (*1::'a nd-fun*) ⊔ *f* · *qstar f* ≤ *qstar f*
using *qstar-unfoldl* **by** *blast*

lemma *fun-star-unfoldr-eq*: (*1::'a nd-fun*) ⊔ (*qstar f*) · *f* = *qstar f*

by *simp*

lemma *fun-star-unfoldr*: $(1::'a\ nd\text{-}fun) \sqcup qstar\ f \cdot f \leq qstar\ f$
by (*simp add*: *fun-star-unfoldr-eq*)

lemma *fun-star-inductl*: $(h::'a\ nd\text{-}fun) \sqcup f \cdot g \leq g \Longrightarrow qstar\ f \cdot\ h \leq g$
using *qstar-inductl* **by** *blast*

lemma *fun-star-inductr*: $(h::'a\ nd\text{-}fun) \sqcup g \cdot f \leq g \Longrightarrow h \cdot qstar\ f \leq g$
by (*simp add*: *qstar-inductr*)

end

References

[1] A. Armstrong, V. B. F. Gomes, and G. Struth. Building program construction and verification tools from algebraic principles. *Formal Aspects of Computing*, 28(2):265–293, 2016.

[2] R. Back and J. von Wright. *Refinement Calculus - A Systematic Introduction*. Springer, 1998.

[3] J. Desharnais and G. Struth. Internal axioms for domain semirings. *Science of Computer Programming*, 76(3):181–203, 2011.

[4] H. Furusawa and G. Struth. Binary multirelations. *Archive of Formal Proofs*, 2015.

[5] V. B. F. Gomes, W. Guttmann, P. Höfner, G. Struth, and T. Weber. Kleene algebras with domain. *Archive of Formal Proofs*, 2016.

[6] V. B. F. Gomes and G. Struth. Program construction and verification components based on kleene algebra. *Archive of Formal Proofs*, 2016.

[7] B. Jacobs. A recipe for state-and-effect triangles. *Logical Methods in Computer Science*, 13(2), 2017.

[8] A. McIver and C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, 2005.

[9] V. Preoteasa. Algebra of monotonic boolean transformers. *Archive of Formal Proofs*, 2011.

[10] K. I. Rosenthal. *Quantales and their Applications*. Longman Scientific & Technical, 1990.