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

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*
⟨*proof*⟩

lemma *iso-botf*: *mono* $\bot$
⟨*proof*⟩

lemma *iso-topf*: *mono* $\top$
⟨*proof*⟩

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

lemma *iso-fcomp*: *mono f* $\Longrightarrow$ *mono g* $\Longrightarrow$ *mono* $(f \circ g)$
⟨*proof*⟩

lemma *iso-fSup*:
fixes F :: $('a$::*order* $\Rightarrow$ $'b$::*complete-lattice*$)$ *set*
shows $(\forall f \in F.$ *mono* $f) \Longrightarrow$ *mono* $(\bigsqcup F)$
⟨*proof*⟩

lemma *iso-fsup*: *mono f* $\Longrightarrow$ *mono g* $\Longrightarrow$ *mono* $(f \sqcup g)$
⟨*proof*⟩

lemma *iso-fInf*:
fixes F :: $('a$::*order* $\Rightarrow$ $'b$::*complete-lattice*$)$ *set*
shows $\forall f \in F.$ *mono* $f \Longrightarrow$ *mono* $(\bigsqcap F)$
⟨*proof*⟩

lemma *iso-finf*: *mono f* $\Longrightarrow$ *mono g* $\Longrightarrow$ *mono* $(f \sqcap g)$
⟨*proof*⟩

lemma *fun-isol*: *mono f* $\Longrightarrow g \leq h \Longrightarrow (f \circ g) \leq (f \circ h)$
⟨*proof*⟩

lemma *fun-isor*: *mono f* $\Longrightarrow g \leq h \Longrightarrow (g \circ f) \leq (h \circ f)$
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *times-iso* :: $'a$::*complete-lattice iso* $\Rightarrow$ $'a$ *iso* $\Rightarrow$ $'a$ *iso* **is** $(\circ)$
⟨*proof*⟩

instance
⟨*proof*⟩

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*: $(x::'a::preorder) \leq y \Longrightarrow y \leq f\ z \Longrightarrow x \leq f\ z$
⟨*proof*⟩

lemma *H-iso-cond2*: $mono\ f \Longrightarrow y \leq z \Longrightarrow x \leq f\ y \Longrightarrow x \leq f\ z$
⟨*proof*⟩

lemma *H-iso-seq*: $mono\ f \Longrightarrow x \leq f\ y \Longrightarrow y \leq g\ z \Longrightarrow x \leq f\ (g\ z)$
⟨*proof*⟩

lemma *H-iso-seq-var*: $mono\ f \Longrightarrow x \leq f\ y \Longrightarrow y \leq g\ z \Longrightarrow x \leq (f \circ g)\ z$
⟨*proof*⟩

lemma *H-iso-fInf*:
fixes $F :: ('a \Rightarrow 'b::complete\text{-}lattice)\ set$
shows $(\forall f \in F.\ x \leq f\ y) \Longrightarrow x \leq (\bigsqcap F)\ y$
⟨*proof*⟩

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

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*::$'a \Rightarrow 'a$ ($\circ$)
⟨*proof*⟩

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* ≡ *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* = $f \sqcap (g \circ h)$
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *iso-fiter-fun2*: *mono f* $\Longrightarrow$ *mono g* $\Longrightarrow$ *mono* (*fiter-fun f g*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *fiter-inductl*:
fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'a$
shows $mono\ f \Longrightarrow mono\ g \Longrightarrow h \leq f \sqcap (g \circ h) \Longrightarrow h \leq fiter\ f\ g$
⟨*proof*⟩

lemma *fiter-fusion*:
fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'a$
assumes $mono\ f$
and $mono\ g$
shows $fiter\ f\ g = fiter\text{-}id\ g \circ f$
⟨*proof*⟩

lemma *fpower-supdistl*:
fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'b{::}complete\text{-}lattice$
shows $mono\ f \Longrightarrow f \circ fstar\ g \leq (\bigsqcap i.\ f \circ fpower\ g\ i)$
⟨*proof*⟩

lemma *fpower-distr*: $fstar\ f \circ g = (\bigsqcap i.\ fpower\ f\ i \circ g)$
⟨*proof*⟩

lemma *fpower-Sup-subcomm*: $mono\ f \Longrightarrow f \circ fstar\ f \leq fstar\ f \circ f$
⟨*proof*⟩

lemma *fpower-inductl*:
fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'a$
shows $mono\ f \Longrightarrow mono\ g \Longrightarrow h \leq g \sqcap (f \circ h) \Longrightarrow h \leq fpower\ f\ i \circ g$
⟨*proof*⟩

lemma *fpower-inductr*:
fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'a$
shows $mono\ f \Longrightarrow mono\ g \Longrightarrow h \leq g \sqcap (h \circ f) \Longrightarrow h \leq g \circ fpower\ f\ i$
⟨*proof*⟩

lemma *fiter-fstar*: $mono\ f \Longrightarrow fiter\text{-}id\ f \leq fstar\ f$
⟨*proof*⟩

lemma *iso-fiter-ext*:
fixes $f :: 'a{::}order \Rightarrow 'b{::}complete\text{-}lattice$
shows $mono\ f \Longrightarrow mono\ (\lambda x.\ y \sqcap f\ x)$
⟨*proof*⟩

lemma *fstar-pred-char*:
fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'a$
shows $mono\ f \Longrightarrow fiter\text{-}id\ f\ x = gfp\ (\lambda y.\ x \sqcap f\ y)$
⟨*proof*⟩

2.5 Propositional Hoare Logic Completed

lemma *H-weak-loop*: $mono\ f \Longrightarrow x \leq f\ x \Longrightarrow x \leq fiter\text{-}id\ f\ x$
⟨*proof*⟩

lemma *iso-fiter*: $mono\ f \Longrightarrow mono\ (fiter\text{-}id\ f)$
⟨*proof*⟩

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* ⇒ $'a$ ⇒ $'a$ **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* ⇒ $'a$ ⇒ $'a$ (‹⌊-⌋›) **where**
$\lfloor x \rfloor\ y = -x \sqcup y$

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

lemma *bqtran-iso*: $mono\ \lfloor x \rfloor$
⟨*proof*⟩

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

lemma *loop-iso*: $mono\ f \Longrightarrow mono\ (fiter\text{-}id\ (\lfloor x \rfloor \circ f) \circ \lfloor y \rfloor)$
⟨*proof*⟩

lemma *H-iso-cond*: $mono\ f \Longrightarrow mono\ g \Longrightarrow\ p \sqcap x \leq f\ y \Longrightarrow q \sqcap x \leq g\ y \Longrightarrow x \leq (inf\ (\lfloor p \rfloor \circ f)\ (\lfloor q \rfloor \circ g))\ y$
⟨*proof*⟩

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*⟩

lemma *btran-spec*: $x \leq \lfloor y \rfloor\ (x \sqcap y)$
⟨*proof*⟩

lemma *btran-neg-spec*: $x \leq \lfloor -y \rfloor\ (x - y)$
⟨*proof*⟩

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

lemma *Ri-least*: $mono\ f \Longrightarrow x \leq f\ y \Longrightarrow Ri\ x\ y\ z \leq f\ z$
⟨*proof*⟩

lemma *Ri-spec*: $x \leq Ri\ x\ y\ y$
⟨*proof*⟩

lemma *Ri-spec-var*: $(\forall z.\ Ri\ x\ y\ z \leq f\ z) \Longrightarrow x \leq f\ y$
⟨*proof*⟩

lemma *Ri-prop*: $mono\ f \Longrightarrow x \leq f\ y \longleftrightarrow (\forall z.\ Ri\ x\ y\ z \leq f\ z)$
⟨*proof*⟩

lemma *iso-Ri*: $mono\ (Ri\ x\ y)$
⟨*proof*⟩

lemma *Ri-weaken*: $x \leq x' \Longrightarrow\ y' \leq y \Longrightarrow Ri\ x\ y\ z \leq Ri\ x'\ y'\ z$
⟨*proof*⟩

lemma *Ri-seq*: $Ri\ x\ y\ z \leq Ri\ x\ w\ (Ri\ w\ y\ z)$
⟨*proof*⟩

lemma *Ri-seq-var*: $Ri\ x\ y\ z \leq ((Ri\ x\ w) \circ (Ri\ w\ y))\ z$
⟨*proof*⟩

lemma *Ri-Inf*: $Ri\ (\bigsqcap\ X)\ y\ z \leq \bigsqcap \{Ri\ x\ y\ z \mid x.\ x \in X\}$
⟨*proof*⟩

lemma *Ri-weak-iter*: $Ri\ x\ x\ y\ \leq fiter\text{-}id\ (Ri\ x\ x)\ y$
⟨*proof*⟩

lemma *Ri-cond*: $Ri\ x\ y\ z \leq (inf\ (\lfloor p \rfloor \circ (Ri\ (p \sqcap x)\ y))\ ((\lfloor q \rfloor \circ (Ri\ (q \sqcap x)\ y))))\ z$
⟨*proof*⟩

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*⟩

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 $f :: 'a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice$
shows $Sup\text{-}pres\ f \Longrightarrow mono\ f$
⟨*proof*⟩

lemma *Inf-pres-iso*:
fixes $f :: 'a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice$
shows $Inf\text{-}pres\ f \Longrightarrow mono\ f$
⟨*proof*⟩

lemma *sup-pres-iso*:
fixes $f :: 'a::lattice \Rightarrow 'b::lattice$
shows $sup\text{-}pres\ f \Longrightarrow mono\ f$
⟨*proof*⟩

lemma *inf-pres-iso*:
fixes $f :: 'a::lattice \Rightarrow 'b::lattice$
shows $inf\text{-}pres\ f \Longrightarrow mono\ f$
⟨*proof*⟩

lemma *Sup-sup-dual*:
fixes $f :: 'a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice$
shows $Sup\text{-}dual\ f \Longrightarrow sup\text{-}dual\ f$
⟨*proof*⟩

lemma *Inf-inf-dual*:
fixes $f :: 'a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice$
shows $Inf\text{-}dual\ f \Longrightarrow inf\text{-}dual\ f$
⟨*proof*⟩

lemma *Sup-bot-dual*:
fixes $f :: 'a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice$
shows*Sup-dual* $f \Longrightarrow bot\text{-}dual\ f$
⟨*proof*⟩

lemma *Inf-top-dual*:
fixes $f :: 'a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice$
shows $Inf\text{-}dual\ f \Longrightarrow top\text{-}dual\ f$
⟨*proof*⟩

Next I show some basic preservation properties.

lemma *Sup-dual2*: $Sup\text{-}dual\ f \Longrightarrow Inf\text{-}dual\ g \Longrightarrow Sup\text{-}pres\ (g \circ f)$
⟨*proof*⟩

lemma *Inf-dual2*: *Sup-dual f* $\Longrightarrow$ *Inf-dual g* $\Longrightarrow$ *Inf-pres* $(f \circ g)$
⟨*proof*⟩

lemma *Sup-pres-id*: *Sup-pres id*
⟨*proof*⟩

lemma *Inf-pres-id*: *Inf-pres id*
⟨*proof*⟩

lemma *Sup-pres-comp*: *Sup-pres f* $\Longrightarrow$ *Sup-pres g* $\Longrightarrow$ *Sup-pres* $(f \circ g)$
⟨*proof*⟩

lemma *Inf-pres-comp*: *Inf-pres f* $\Longrightarrow$ *Inf-pres g* $\Longrightarrow$ *Inf-pres*$(f \circ g)$
⟨*proof*⟩

lemma *Sup-pres-Sup*:
fixes F :: $('a$::*complete-lattice* $\Rightarrow$ $'b$::*complete-lattice*$)$ *set*
shows $\forall f \in F$. *Sup-pres f* $\Longrightarrow$ *Sup-pres* $(\bigsqcup F)$
⟨*proof*⟩

lemma *Inf-pres-Inf*:
fixes F :: $('a$::*complete-lattice* $\Rightarrow$ $'b$::*complete-lattice*$)$ *set*
shows $\forall f \in F$. *Inf-pres f* $\Longrightarrow$ *Inf-pres* $(\bigsqcap F)$
⟨*proof*⟩

lemma *Sup-pres-sup*:
fixes f :: $'a$::*complete-lattice* $\Rightarrow$ $'b$::*complete-lattice*
shows *Sup-pres f* $\Longrightarrow$ *Sup-pres g* $\Longrightarrow$ *Sup-pres* $(f \sqcup g)$
⟨*proof*⟩

lemma *Inf-pres-inf*:
fixes f :: $'a$::*complete-lattice* $\Rightarrow$ $'b$::*complete-lattice*
shows *Inf-pres f* $\Longrightarrow$ *Inf-pres g* $\Longrightarrow$ *Inf-pres* $(f \sqcap g)$
⟨*proof*⟩

lemma *Sup-pres-botf*: *Sup-pres* $(\lambda x.\ \bot$::$'a$::*complete-lattice*$)$
⟨*proof*⟩

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-lattice*$)$
⟨*proof*⟩

lemma *Sup-pres* $(\lambda x.\ \top$::$'a$::*complete-lattice*$)$
⟨*proof*⟩

lemma *Inf-pres-topf*: *Inf-pres* $(\lambda x.\ \top$::$'a$::*complete-lattice*$)$
⟨*proof*⟩

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 $(uminus\ f = g) = (uminus\ g = f)$
⟨*proof*⟩

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 $(\partial \circ f = g) = (\partial \circ g = f)$
⟨*proof*⟩

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 $(f \circ \partial = g) = (g \circ \partial = f)$
⟨*proof*⟩

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 $(\partial \circ f = \partial \circ g) = (f = g)$
⟨*proof*⟩

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 $(f \circ \partial = g \circ \partial) = (f = g)$
⟨*proof*⟩

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 $(Inf\text{-}pres\ f) = (Sup\text{-}pres\ (\partial_F\ f))$
⟨*proof*⟩

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 $(Sup\text{-}pres\ f) = (Inf\text{-}pres\ (\partial_F\ f))$
⟨*proof*⟩

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 $f :: 'a::complete\text{-}lattice \Rightarrow 'a$
shows $Inf\text{-}pres\ f \Longrightarrow x \leq f\ x \Longrightarrow x \leq fpower\ f\ i\ x$
⟨*proof*⟩

lemma *H-Inf-pres-fstar*:
fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'a$
shows $Inf\text{-}pres\ f \Longrightarrow x \leq f\ x \Longrightarrow x \leq fstar\ f\ x$
⟨*proof*⟩

lemma *fpower-Inf-pres*: $Inf\text{-}pres\ f \Longrightarrow Inf\text{-}pres\ (fpower\ f\ i)$
⟨*proof*⟩

lemma *fstar-Inf-pres*:
fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'a$
shows $Inf\text{-}pres\ f \Longrightarrow Inf\text{-}pres\ (fstar\ f)$
⟨*proof*⟩

lemma *fstar-unfoldl-var* [*simp*]:
fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'a$
shows $Inf\text{-}pres\ f \Longrightarrow x \sqcap f\ (fstar\ f\ x) = fstar\ f\ x$
⟨*proof*⟩

lemma *fstar-fiter-id*: $Inf\text{-}pres\ f \Longrightarrow fstar\ f = fiter\text{-}id\ f$
⟨*proof*⟩

lemma *fstar-unfoldl* [*simp*]:
fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'a$
shows $Inf\text{-}pres\ f \Longrightarrow id \sqcap (f \circ fstar\ f) = fstar\ f$
⟨*proof*⟩

lemma *fpower-Inf-comm*:
fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'a$
shows $Inf\text{-}pres\ f \Longrightarrow f\ (\bigsqcap i.\ fpower\ f\ i\ x) = (\bigsqcap i.\ fpower\ f\ i\ (f\ x))$
⟨*proof*⟩

lemma *fstar-comm*:
fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'a$
shows $Inf\text{-}pres\ f \Longrightarrow f \circ fstar\ f = fstar\ f \circ f$
⟨*proof*⟩

lemma *fstar-unfoldr* [*simp*]:
fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'a$
shows $Inf\text{-}pres\ f \Longrightarrow id \sqcap (fstar\ f \circ f) = fstar\ f$
⟨*proof*⟩

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* = $\{f::'a::complete\text{-}lattice \Rightarrow 'a.\ Inf\text{-}pres\ f\}$
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *times-Inf-pres* :: *'a*::*complete-lattice Inf-pres* ⇒ *'a Inf-pres* ⇒ *'a Inf-pres* **is** (∘)
⟨*proof*⟩

lift-definition *Sup-Inf-pres* :: *'a*::*complete-lattice Inf-pres set* ⇒ *'a Inf-pres* **is** *Inf*
⟨*proof*⟩

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

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

instance
⟨*proof*⟩

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* (‹$\mathcal{P}$›)

4.1 The Powerset Monad

First I recall functoriality of the powerset functor.

lemma *P-func1*: $\mathcal{P}\ (f \circ g) = \mathcal{P}\ f \circ \mathcal{P}\ g$
⟨*proof*⟩

lemma *P-func2*: $\mathcal{P}\ id = id$
⟨*proof*⟩

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*: $\mathcal{P}\ f \circ \eta = \eta \circ id\ f$
⟨*proof*⟩

lemma *mu-nt*: $\mu \circ (\mathcal{P} \circ \mathcal{P})\ f = (\mathcal{P}\ f) \circ \mu$
⟨*proof*⟩

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

lemma *pow-assoc*: $(\mu::'a\ set\ set \Rightarrow 'a\ set) \circ \mathcal{P}\ (\mu::'a\ set\ set \Rightarrow 'a\ set) = (\mu::'a\ set\ set \Rightarrow 'a\ set) \circ (\mu::'a\ set\ set\ set \Rightarrow 'a\ set\ set)$
⟨*proof*⟩

lemma *pow-un1*: $(\mu::'a\ set\ set \Rightarrow 'a\ set) \circ (\mathcal{P}\ (\eta::'a \Rightarrow 'a\ set)) = (id::'a\ set \Rightarrow 'a\ set)$
⟨*proof*⟩

lemma *pow-un2*: $(\mu::'a\ set\ set \Rightarrow 'a\ set) \circ (\eta::'a\ set \Rightarrow 'a\ set\ set) = (id::'a\ set \Rightarrow 'a\ set)$
⟨*proof*⟩

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** $\langle\circ_K\rangle$ *75*) **where**
$f \circ_K g = \mu \circ \mathcal{P}\ g \circ f$

lemma *kcomp-prop*: $(f \circ_K g)\ x = (\bigsqcup y \in f\ x.\ g\ y)$
⟨*proof*⟩

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

lemma *klift-prop*: $(f^\dagger)\ X = (\bigsqcup x \in X.\ f\ x)$
⟨*proof*⟩

lemma *kcomp-klift*: $f \circ_K g = g^\dagger \circ f$
⟨*proof*⟩

lemma *klift-prop1*: $(f^\dagger \circ g)^\dagger = f^\dagger \circ g^\dagger$
⟨*proof*⟩

lemma *klift-eta-inv1* [*simp*]: $f^\dagger \circ \eta = f$
⟨*proof*⟩

lemma *klift-eta-pres* [*simp*]: $\eta^\dagger = (id::'a\ set \Rightarrow 'a\ set)$
⟨*proof*⟩

lemma *klift-id-pres* [*simp*]: $id^\dagger = \mu$
⟨*proof*⟩

lemma *kcomp-assoc*: $(f \circ_K g) \circ_K h = f \circ_K (g \circ_K h)$
⟨*proof*⟩

lemma *kcomp-idl* [*simp*]: $\eta \circ_K f = f$
⟨*proof*⟩

lemma *kcomp-idr* [*simp*]: $f \circ_K \eta = f$
⟨*proof*⟩

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

interpretation *kmon*: *monoid-mult* η $(\circ_K)$
⟨*proof*⟩

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*: $\eta \circ (f \circ g) = (\eta \circ g) \circ_K (\eta \circ f)$
⟨*proof*⟩

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*]: $\sigma \circ \mathcal{P}\ (\sigma::'a::complete\text{-}lattice\ set \Rightarrow 'a) = \sigma \circ \mu$
⟨*proof*⟩

lemma *em-id* [*simp*]: $\sigma \circ \eta = (id::'a::complete\text{-}lattice \Rightarrow 'a)$
⟨*proof*⟩

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*]: $\mu \circ \mathcal{P}\ \mu = \mu \circ \mu$
⟨*proof*⟩

lemma *em-mu-id* [*simp*]: $\mu \circ \eta = id$
⟨*proof*⟩

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

class *eilenberg-moore-pow* =
fixes *smap* :: $'a\ set \Rightarrow 'a$
assumes *smap-assoc*: $smap \circ \mathcal{P}\ smap = smap \circ \mu$
and *smap-id*: $smap \circ \eta = id$

begin

definition *sleq* = $(\lambda x\ y.\ smap\ \{x,y\} = y)$

definition *sle* = $(\lambda x\ y.\ sleq\ x\ y \wedge y \neq x)$

lemma *smap-un1*: $smap\ \{x,\ smap\ Y\} = smap\ (\{x\} \cup Y)$
⟨*proof*⟩

lemma *smap-comm*: $smap\ \{x,\ smap\ Y\} = smap\ \{smap\ Y,\ x\}$
⟨*proof*⟩

lemma *smap-un2*: $smap\ \{smap\ X,\ y\} = smap\ (X \cup \{y\})$

⟨*proof*⟩

lemma *sleq-refl*: *sleq x x*
⟨*proof*⟩

lemma *sleq-trans*: *sleq x y* ⟹ *sleq y z* ⟹ *sleq x z*
⟨*proof*⟩

lemma *sleq-antisym*: *sleq x y* ⟹ *sleq y x* ⟹ *x* = *y*
⟨*proof*⟩

lemma *smap-ub*: *x* ∈ *A* ⟹ *sleq x* (*smap A*)
⟨*proof*⟩

lemma *smap-lub*: (⋀*x*. *x* ∈ *A* ⟹ *sleq x z*) ⟹ *sleq* (*smap A*) *z*
⟨*proof*⟩

sublocale *smap-Sup-lat*: *Sup-lattice smap sleq sle*
⟨*proof*⟩

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 ≡ (*λx*::*'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* (‹*F*›) **where**
F R = *Image R* ∘ *η*

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

The functors form a bijective pair.

lemma *r2f2r-inv1* [*simp*]: $\mathcal{R} \circ \mathcal{F} = id$
⟨*proof*⟩

lemma *f2r2f-inv2* [*simp*]: $\mathcal{F} \circ \mathcal{R} = id$
⟨*proof*⟩

lemma *r2f-f2r-galois*: $(\mathcal{R}\ f = R) = (\mathcal{F}\ R = f)$
⟨*proof*⟩

lemma *r2f-f2r-galois-var*: $(\mathcal{R} \circ f = R) = (\mathcal{F} \circ R = f)$
⟨*proof*⟩

lemma *r2f-f2r-galois-var2*: $(f \circ \mathcal{R} = R) = (R \circ \mathcal{F} = f)$
⟨*proof*⟩

lemma *r2f-inj*: *inj* $\mathcal{F}$
⟨*proof*⟩

lemma *f2r-inj*: *inj* $\mathcal{R}$
⟨*proof*⟩

lemma *r2f-mono*: $\forall f\ g.\ \mathcal{F} \circ f = \mathcal{F} \circ g \longrightarrow f = g$
⟨*proof*⟩

lemma *f2r-mono*: $\forall f\ g.\ \mathcal{R} \circ f = \mathcal{R} \circ g \longrightarrow f = g$
⟨*proof*⟩

lemma *r2f-mono-iff*: $(\mathcal{F} \circ f = \mathcal{F} \circ g) = (f = g)$
⟨*proof*⟩

lemma *f2r-mono-iff* : $(\mathcal{R} \circ f = \mathcal{R} \circ g) = (f = g)$
⟨*proof*⟩

lemma *r2f-inj-iff*: $(\mathcal{R}\ f = \mathcal{R}\ g) = (f = g)$
⟨*proof*⟩

lemma *f2r-inj-iff*: $(\mathcal{F}\ R = \mathcal{F}\ S) = (R = S)$
⟨*proof*⟩

lemma *r2f-surj*: *surj* $\mathcal{F}$
⟨*proof*⟩

lemma *f2r-surj*: *surj* $\mathcal{R}$
⟨*proof*⟩

lemma *r2f-epi*: $\forall f\ g.\ f \circ \mathcal{F} = g \circ \mathcal{F} \longrightarrow f = g$
⟨*proof*⟩

lemma *f2r-epi*: $\forall f\ g.\ f \circ \mathcal{R} = g \circ \mathcal{R} \longrightarrow f = g$

⟨*proof*⟩

lemma *r2f-epi-iff*: $(f \circ \mathcal{F} = g \circ \mathcal{F}) = (f = g)$
⟨*proof*⟩

lemma *f2r-epi-iff*: $(f \circ \mathcal{R} = g \circ \mathcal{R}) = (f = g)$
⟨*proof*⟩

lemma *r2f-bij*: *bij* $\mathcal{F}$
⟨*proof*⟩

lemma *f2r-bij*: *bij* $\mathcal{R}$
⟨*proof*⟩

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
⟨*proof*⟩

lemma *f2r-uncurry*: *f2r* f = (*Collect* $\circ$ *case-prod*) (*s2p* $\circ$ f)
⟨*proof*⟩

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*: $\mathcal{F}\ (R\ ;\ S) = \mathcal{F}\ R \circ_K \mathcal{F}\ S$
⟨*proof*⟩

lemma *r2f-Id-pres* [*simp*]: $\mathcal{F}\ Id = \eta$
⟨*proof*⟩

lemma *r2f-Sup-pres*: *Sup-pres* $\mathcal{F}$
⟨*proof*⟩

lemma *r2f-Sup-pres-var*: $\mathcal{F}\ (\bigcup R) = (\bigsqcup r \in R.\ \mathcal{F}\ r)$
⟨*proof*⟩

lemma *r2f-sup-pres*: *sup-pres* $\mathcal{F}$
⟨*proof*⟩

lemma *r2f-Inf-pres*: *Inf-pres* $\mathcal{F}$
⟨*proof*⟩

lemma *r2f-Inf-pres-var*: $\mathcal{F}\ (\bigsqcap R) = (\bigsqcap r \in R.\ \mathcal{F}\ r)$
⟨*proof*⟩

lemma *r2f-inf-pres*: *inf-pres* $\mathcal{F}$
⟨*proof*⟩

lemma *r2f-bot-pres*: *bot-pres* $\mathcal{F}$
⟨*proof*⟩

lemma *r2f-top-pres*: *top-pres* $\mathcal{F}$
⟨*proof*⟩

lemma *r2f-leq*: $(R \subseteq S) = (\mathcal{F}\ R \leq \mathcal{F}\ S)$
⟨*proof*⟩

Dual statements for f2r hold. Can one automate this?

lemma *f2r-kcomp-pres*: $\mathcal{R}\ (f \circ_K g) = \mathcal{R}\ f\ ;\ \mathcal{R}\ g$
⟨*proof*⟩

lemma *f2r-eta-pres* [*simp*]: $\mathcal{R}\ \eta = Id$
⟨*proof*⟩

lemma *f2r-Sup-pres*:*Sup-pres* $\mathcal{R}$
⟨*proof*⟩

lemma *f2r-Sup-pres-var*: $\mathcal{R}\ (\bigsqcup F) = (\bigsqcup f \in F.\ \mathcal{R}\ f)$
⟨*proof*⟩

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

lemma *f2r-Inf-pres*: *Inf-pres* $\mathcal{R}$
⟨*proof*⟩

lemma *f2r-Inf-pres-var*: $\mathcal{R}\ (\bigsqcap F) = (\bigcap f \in F.\ \mathcal{R}\ f)$
⟨*proof*⟩

lemma *f2r-inf-pres*: *inf-pres* $\mathcal{R}$
⟨*proof*⟩

lemma *f2r-bot-pres*: *bot-pres* $\mathcal{R}$
⟨*proof*⟩

lemma *f2r-top-pres*: *top-pres* $\mathcal{R}$
⟨*proof*⟩

lemma *f2r-leq*: $(f \leq g) = (\mathcal{R}\ f \subseteq \mathcal{R}\ g)$
⟨*proof*⟩

Relational subidentities are isomorphic to particular Kleisli arrows.

lemma *r2f-Id-on1*: $\mathcal{F}\ (Id\text{-}on\ X) = (\lambda x.\ \mathit{if}\ x \in X\ \mathit{then}\ \{x\}\ \mathit{else}\ \{\})$

⟨*proof*⟩

lemma *r2f-Id-on2*: $\mathcal{F}\ (\mathit{Id\text{-}on}\ X) \circ_K f = (\lambda x.\ \mathit{if}\ x \in X\ \mathit{then}\ f\ x\ \mathit{else}\ \{\})$
⟨*proof*⟩

lemma *r2f-Id-on3*: $f \circ_K \mathcal{F}\ (\mathit{Id\text{-}on}\ X) = (\lambda x.\ X \cap f\ x)$
⟨*proof*⟩

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* ($\langle\smile\rangle$)

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

Kop is a contravariant functor.

lemma *kop-contrav*: $op_K\ (f \circ_K g) = op_K\ g \circ_K op_K\ f$
⟨*proof*⟩

lemma *kop-func2* [*simp*]: $op_K\ \eta = \eta$
⟨*proof*⟩

lemma *converse-idem* [*simp*]: $(\smile) \circ (\smile) = id$
⟨*proof*⟩

lemma *converse-galois*: $((\smile) \circ f = g) = ((\smile) \circ g = f)$
⟨*proof*⟩

lemma *converse-galois2*: $(f \circ (\smile) = g) = (g \circ (\smile) = f)$
⟨*proof*⟩

lemma *converse-mono-iff*: $((\smile) \circ f = (\smile) \circ g) = (f = g)$
⟨*proof*⟩

lemma *converse-epi-iff*: $(f \circ (\smile) = g \circ (\smile)) = (f = g)$
⟨*proof*⟩

lemma *kop-idem* [*simp*]: $op_K \circ op_K = id$
⟨*proof*⟩

lemma *kop-galois*: $(op_K\ f = g) = (op_K\ g = f)$
⟨*proof*⟩

lemma *kop-galois-var*: $(op_K \circ f = g) = (op_K \circ g = f)$
⟨*proof*⟩

lemma *kop-galois-var2*: $(f \circ op_K = g) = (g \circ op_K = f)$
⟨*proof*⟩

lemma *kop-inj*: *inj* op_K
⟨*proof*⟩

lemma *kop-inj-iff*: $(op_K\ f = op_K\ g) = (f = g)$
⟨*proof*⟩

lemma *kop-surj*: *surj* op_K
⟨*proof*⟩

lemma *kop-bij*: *bij* op_K
⟨*proof*⟩

lemma *kop-mono*: $(op_K \circ f = op_K \circ g) \Longrightarrow (f = g)$
⟨*proof*⟩

lemma *kop-mono-iff*: $(op_K \circ f = op_K \circ g) = (f = g)$
⟨*proof*⟩

lemma *kop-epi*: $(f \circ op_K = g \circ op_K) \Longrightarrow (f = g)$
⟨*proof*⟩

lemma *kop-epi-iff*: $(f \circ op_K = g \circ op_K) = (f = g)$
⟨*proof*⟩

lemma *Sup-pres-kop*: *Sup-pres* op_K
⟨*proof*⟩

lemma *Inf-pres-kop*: *Inf-pres* op_K
⟨*proof*⟩

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*: $bd_{\mathcal{F}}\ f\ X = \{y.\ \exists x.\ y \in f\ x \wedge x \in X\}$
⟨*proof*⟩

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

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*⟩

lemma *fbd-ifbd-inv2-inv*: $(bd_{\mathcal{F}} \circ bd^-_{\mathcal{F}})\ \varphi = \varphi \Longrightarrow$ *Sup-pres* φ
⟨*proof*⟩

lemma *fbd-ifbd-inv2-iff*: $((bd_{\mathcal{F}} \circ bd^-_{\mathcal{F}})\ \varphi = \varphi) = (\textit{Sup-pres}\ \varphi)$
⟨*proof*⟩

lemma *fbd-inj*: *inj* $bd_{\mathcal{F}}$
⟨*proof*⟩

lemma *fbd-inj-iff*: $(bd_{\mathcal{F}}\ f = bd_{\mathcal{F}}\ g) = (f = g)$
⟨*proof*⟩

lemma *ifbd-inj*: *Sup-pres* $\varphi \Longrightarrow$ *Sup-pres* $\psi \Longrightarrow bd^-_{\mathcal{F}}\ \varphi = bd^-_{\mathcal{F}}\ \psi \Longrightarrow \varphi = \psi$
⟨*proof*⟩

lemma *ifbd-inj-iff*: *Sup-pres* $\varphi \Longrightarrow$ *Sup-pres* $\psi \Longrightarrow (bd^-_{\mathcal{F}}\ \varphi = bd^-_{\mathcal{F}}\ \psi) = (\varphi = \psi)$
⟨*proof*⟩

lemma *fbd-ifbd-galois*: *Sup-pres* $\varphi \Longrightarrow (bd^-_{\mathcal{F}}\ \varphi = f) = (bd_{\mathcal{F}}\ f = \varphi)$
⟨*proof*⟩

lemma *fbd-surj*: *Sup-pres* $\varphi \Longrightarrow (\exists f.\ bd_{\mathcal{F}}\ f = \varphi)$

⟨*proof*⟩

lemma *ifbd-surj*: *surj* $bd^-{}_{\mathcal{F}}$
⟨*proof*⟩

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*: $bd_{\mathcal{F}}\ (f \circ_K g) = bd_{\mathcal{F}}\ g \circ bd_{\mathcal{F}}\ f$
⟨*proof*⟩

lemma *fbd-Sup-pres*: *Sup-pres* $bd_{\mathcal{F}}$
⟨*proof*⟩

lemma *fbd-sup-pres*: *sup-pres* $bd_{\mathcal{F}}$
⟨*proof*⟩

lemma *fbd-Disj*: *Sup-pres* $(bd_{\mathcal{F}}\ f)$
⟨*proof*⟩

lemma *fbd-disj*: *sup-pres* $(bd_{\mathcal{F}}\ f)$
⟨*proof*⟩

lemma *fbd-bot-pres*: *bot-pres* $bd_{\mathcal{F}}$
⟨*proof*⟩

lemma *fbd-zero-pres2* [*simp*]: $bd_{\mathcal{F}}\ f\ \{\} = \{\}$
⟨*proof*⟩

lemma *fbd-iso*: $X \subseteq Y \longrightarrow bd_{\mathcal{F}}\ f\ X \subseteq bd_{\mathcal{F}}\ f\ Y$
⟨*proof*⟩

The following counterexamples show that Infs are not preserved.

lemma *top-pres* $bd_{\mathcal{F}}$
⟨*proof*⟩

lemma *inf-pres* $bd_{\mathcal{F}}$
⟨*proof*⟩

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

lemma *ifbd-comp-pres*: *Sup-pres* $\varphi \Longrightarrow bd^-{}_{\mathcal{F}}\ (\varphi \circ \psi) = bd^-{}_{\mathcal{F}}\ \psi \circ_K bd^-{}_{\mathcal{F}}\ \varphi$
⟨*proof*⟩

lemma *ifbd-Sup-pres*: *Sup-pres* $bd^-{}_{\mathcal{F}}$
⟨*proof*⟩

lemma *ifbd-sup-pres*: *sup-pres* $bd^-{}_{\mathcal{F}}$
⟨*proof*⟩

lemma *ifbd-Inf-pres*: *Inf-pres* $bd^-_{\mathcal{F}}$
⟨*proof*⟩

lemma *ifbd-inf-pres*: *inf-pres* $bd^-_{\mathcal{F}}$
⟨*proof*⟩

lemma *ifbd-bot-pres*: *bot-pres* $bd^-_{\mathcal{F}}$
⟨*proof*⟩

lemma *ifbd-top-pres*: *top-pres* $bd^-_{\mathcal{F}}$
⟨*proof*⟩

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*: $bd_{\mathcal{R}} = (`)$
⟨*proof*⟩

lemma *rbd-set*: $bd_{\mathcal{R}}\ R\ X = \{y.\ \exists x \in X.\ (x,y) \in R\}$
⟨*proof*⟩

lemma *irbd-set*: $bd^-_{\mathcal{R}}\ \varphi = \{(x,y).\ y \in (\varphi \circ \eta)\ x\}$
⟨*proof*⟩

lemma *irbd-set-var*: $bd^-_{\mathcal{R}}\ \varphi = \{(x,y).\ y \in \varphi\ \{x\}\}$
⟨*proof*⟩

lemma *rbd-irbd-inv1* [*simp*]: $bd^-_{\mathcal{R}} \circ bd_{\mathcal{R}} = id$
⟨*proof*⟩

lemma *irbd-rbd-inv2*: *Sup-pres* $\varphi \Longrightarrow (bd_{\mathcal{R}} \circ bd^{-}{}_{\mathcal{R}})\ \varphi = \varphi$
⟨*proof*⟩

lemma *irbd-rbd-inv2-inv*: $(bd_{\mathcal{R}} \circ bd^{-}{}_{\mathcal{R}})\ \varphi = \varphi \Longrightarrow$ *Sup-pres* φ
⟨*proof*⟩

lemma *irbd-rbd-inv2-iff*: $((bd_{\mathcal{R}} \circ bd^{-}{}_{\mathcal{R}})\ \varphi = \varphi) = (\textit{Sup-pres}\ \varphi)$
⟨*proof*⟩

lemma *rbd-inj*: *inj* $bd_{\mathcal{R}}$
⟨*proof*⟩

lemma *rbd-translate*: $(bd_{\mathcal{R}}\ R = bd_{\mathcal{R}}\ S) = (R = S)$
⟨*proof*⟩

lemma *irbd-inj*: *Sup-pres* $\varphi \Longrightarrow$ *Sup-pres* $\psi \Longrightarrow bd^{-}{}_{\mathcal{R}}\ \varphi = bd^{-}{}_{\mathcal{R}}\ \psi \Longrightarrow \varphi = \psi$
⟨*proof*⟩

lemma *irbd-inj-iff*: *Sup-pres* $\varphi \Longrightarrow$ *Sup-pres* $\psi \Longrightarrow (bd^{-}{}_{\mathcal{R}}\ \varphi = bd^{-}{}_{\mathcal{R}}\ \psi) = (\varphi = \psi)$
⟨*proof*⟩

lemma *rbd-surj*: *Sup-pres* $\varphi \Longrightarrow (\exists\, R.\ bd_{\mathcal{R}}\ R = \varphi)$
⟨*proof*⟩

lemma *irbd-surj*: *surj* $bd^{-}{}_{\mathcal{R}}$
⟨*proof*⟩

lemma *rbd-irbd-galois*: *Sup-pres* $\varphi \Longrightarrow (\varphi = bd_{\mathcal{R}}\ R) = (R = bd^{-}{}_{\mathcal{R}}\ \varphi)$
⟨*proof*⟩

lemma *rbd-comp-pres*: $bd_{\mathcal{R}}\ (R\ ;\ S) = bd_{\mathcal{R}}\ S \circ bd_{\mathcal{R}}\ R$
⟨*proof*⟩

lemma *rbd-Id-pres*: $bd_{\mathcal{R}}\ Id = id$
⟨*proof*⟩

lemma *rbd-Un-pres*: *Sup-pres* $bd_{\mathcal{R}}$
⟨*proof*⟩

lemma *rbd-un-pres*: *sup-pres* $bd_{\mathcal{R}}$
⟨*proof*⟩

lemma *inf-pres* $bd_{\mathcal{R}}$
⟨*proof*⟩

lemma *rbd-disj*: *Sup-pres* $(bd_{\mathcal{R}}\ R)$
⟨*proof*⟩

lemma *rbd-disj2*: *sup-pres* ($bd_{\mathcal{R}}$ *R*)
⟨*proof*⟩

lemma *rbd-bot-pres*: *bot-pres* $bd_{\mathcal{R}}$
⟨*proof*⟩

lemma *rbd-zero-pres2* [*simp*]: $bd_{\mathcal{R}}$ *R* {} = {}
⟨*proof*⟩

lemma *rbd-univ*: $bd_{\mathcal{R}}$ *R UNIV* = *Range R*
⟨*proof*⟩

lemma *rbd-iso*: *X* ⊆ *Y* ⟹ $bd_{\mathcal{R}}$ *R X* ⊆ $bd_{\mathcal{R}}$ *R Y*
⟨*proof*⟩

lemma *irbd-comp-pres*: *Sup-pres* φ ⟹ $bd^-{}_{\mathcal{R}}$ (φ ∘ ψ) = $bd^-{}_{\mathcal{R}}$ ψ ; $bd^-{}_{\mathcal{R}}$ φ
⟨*proof*⟩

lemma *irbd-id-pres* [*simp*]: $bd^-{}_{\mathcal{R}}$ *id* = *Id*
⟨*proof*⟩

lemma *irbd-Sup-pres*: *Sup-pres* $bd^-{}_{\mathcal{R}}$
⟨*proof*⟩

lemma *irbd-sup-pres*: *sup-pres* $bd^-{}_{\mathcal{R}}$
⟨*proof*⟩

lemma *irbd-Inf-pres*: *Inf-pres* $bd^-{}_{\mathcal{R}}$
⟨*proof*⟩

lemma *irbd-inf-pres*: *inf-pres* $bd^-{}_{\mathcal{R}}$
⟨*proof*⟩

lemma *irbd-bot-pres*: *bot-pres* $bd^-{}_{\mathcal{R}}$
⟨*proof*⟩

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* ⇒ '*b set*) ⇒ '*b set* ⇒ '*a set* (‹*fb*$_{\mathcal{F}}$›) **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*: $fb_{\mathcal{F}}\ f = \partial \circ bd_{\mathcal{F}}\ (op_K\ f) \circ \partial$
⟨*proof*⟩

lemma *ffb-prop-var*: $fb_{\mathcal{F}}\ f = uminus \circ bd_{\mathcal{F}}\ (op_K\ f) \circ uminus$
⟨*proof*⟩

lemma *ffb-fbd-dual*: $\partial \circ fb_{\mathcal{F}}\ f = bd_{\mathcal{F}}\ (op_K\ f) \circ \partial$
⟨*proof*⟩

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

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

lemma *ffb-set*: $fb_{\mathcal{F}}\ f = (\lambda Y.\ \{x.\ f\ x \subseteq\ Y\})$
⟨*proof*⟩

Forward boxes and backward diamonds are adjoints.

lemma *ffb-fbd-galois*: $(bd_{\mathcal{F}}\ f) \dashv (fb_{\mathcal{F}}\ f)$
⟨*proof*⟩

lemma *iffb-inv1*: $fb^-{}_{\mathcal{F}} \circ fb_{\mathcal{F}} = id$
⟨*proof*⟩

lemma *iffb-inv2-aux*: *Inf-pres* $\varphi \Longrightarrow \bigsqcap\{X.\ x \in \varphi\ X\} \subseteq\ Y \Longrightarrow x \in \varphi\ Y$
⟨*proof*⟩

lemma *iffb-inv2*: *Inf-pres* $\varphi \Longrightarrow (fb_{\mathcal{F}} \circ fb^-{}_{\mathcal{F}})\ \varphi = \varphi$
⟨*proof*⟩

lemma *iffb-inv2-inv*: $(fb_{\mathcal{F}} \circ fb^-{}_{\mathcal{F}})\ \varphi = \varphi \Longrightarrow$ *Inf-pres* φ
⟨*proof*⟩

lemma *iffb-inv2-iff*: $((fb_{\mathcal{F}} \circ fb^-{}_{\mathcal{F}})\ \varphi = \varphi) = (\textit{Inf-pres}\ \varphi)$
⟨*proof*⟩

lemma *ffb-inj*: *inj* $fb_{\mathcal{F}}$
⟨*proof*⟩

lemma *ffb-inj-iff*: $(fb_{\mathcal{F}}\ f = fb_{\mathcal{F}}\ g) = (f = g)$
⟨*proof*⟩

lemma *ffb-iffb-galois*: $Inf\text{-}pres\ \varphi \Longrightarrow (ffb^{-}_{\mathcal{F}}\ \varphi = f) = (fb_{\mathcal{F}}\ f = \varphi)$
⟨*proof*⟩

lemma *iffb-inj*: $Inf\text{-}pres\ \varphi \Longrightarrow Inf\text{-}pres\ \psi \Longrightarrow ffb^{-}_{\mathcal{F}}\ \varphi = ffb^{-}_{\mathcal{F}}\ \psi \Longrightarrow \varphi = \psi$
⟨*proof*⟩

lemma *iffb-inj-iff*: $Inf\text{-}pres\ \varphi \Longrightarrow Inf\text{-}pres\ \psi \Longrightarrow (ffb^{-}_{\mathcal{F}}\ \varphi = ffb^{-}_{\mathcal{F}}\ \psi) = (\varphi = \psi)$
⟨*proof*⟩

lemma *ffb-surj*: $Inf\text{-}pres\ \varphi \Longrightarrow (\exists f.\ fb_{\mathcal{F}}\ f = \varphi)$
⟨*proof*⟩

lemma *iffb-surj*: $surj\ ffb^{-}_{\mathcal{F}}$
⟨*proof*⟩

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

lemma *iffb-ifbd-dual*: $Inf\text{-}pres\ \varphi \Longrightarrow ffb^{-}_{\mathcal{F}}\ \varphi = (op_K \circ bd^{-}_{\mathcal{F}} \circ \partial_F)\ \varphi$
⟨*proof*⟩

lemma *fbd-ffb-dual*: $\partial_F \circ fb_{\mathcal{F}} \circ op_K = bd_{\mathcal{F}}$
⟨*proof*⟩

lemma *ffbd-ffb-dual-var*: $\partial \circ bd_{\mathcal{F}}\ f = fb_{\mathcal{F}}\ (op_K\ f) \circ \partial$
⟨*proof*⟩

lemma *ifbd-iffb-dual*: $Sup\text{-}pres\ \varphi \Longrightarrow bd^{-}_{\mathcal{F}}\ \varphi = (op_K \circ ffb^{-}_{\mathcal{F}} \circ \partial_F)\ \varphi$
⟨*proof*⟩

lemma *ffb-kcomp-pres*: $fb_{\mathcal{F}}\ (f \circ_K g) = fb_{\mathcal{F}}\ f \circ fb_{\mathcal{F}}\ g$
⟨*proof*⟩

lemma *ffb-eta-pres*: $fb_{\mathcal{F}}\ \eta = id$
⟨*proof*⟩

lemma *ffb-Sup-dual*: $Sup\text{-}dual\ fb_{\mathcal{F}}$
⟨*proof*⟩

lemma *ffb-Sup-dual-var*: $fb_{\mathcal{F}}\ (\bigsqcup F) = (\bigsqcap f \in F.\ fb_{\mathcal{F}}\ f)$
⟨*proof*⟩

lemma *ffb-sup-dual*: $sup\text{-}dual\ fb_{\mathcal{F}}$
⟨*proof*⟩

lemma *ffb-zero-dual*: $fb_{\mathcal{F}}\ \zeta = (\lambda X.\ UNIV)$
⟨*proof*⟩

lemma *inf-dual ffb*
⟨*proof*⟩

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

lemma *iffb-comp-pres*:
assumes *Inf-pres* φ
assumes *Inf-pres* ψ
shows $fb^-_{\mathcal{F}}\ (\varphi \circ \psi) = fb^-_{\mathcal{F}}\ \varphi \circ_K fb^-_{\mathcal{F}}\ \psi$
⟨*proof*⟩

lemma *iffb-id-pres*: $fb^-_{\mathcal{F}}\ id = \eta$
⟨*proof*⟩

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*⟩

lemma *iffb-Sup-dual*: *Sup-dual* $fb^-_{\mathcal{F}}$
⟨*proof*⟩

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*⟩

lemma *iffb-sup-dual*: $fb^-_{\mathcal{F}}\ (\varphi \sqcup \psi) = fb^-_{\mathcal{F}}\ \varphi \sqcap fb^-_{\mathcal{F}}\ \psi$
⟨*proof*⟩

lemma *iffb-top-pres* [*simp*]: $fb^-_{\mathcal{F}}\ \top = \zeta$
⟨*proof*⟩

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*: $fb_{\mathcal{R}}\ R = \partial_F\ (bd_{\mathcal{R}}\ (R^{-1}))$
⟨*proof*⟩

lemma *rbd-rfb-dual*: $bd_{\mathcal{R}}\ R = \partial_F\ (fb_{\mathcal{R}}\ (R^{-1}))$
⟨*proof*⟩

lemma *irfb-irbd-dual*: *Inf-pres* $\varphi \Longrightarrow fb^{-}{}_{\mathcal{R}}\ \varphi = ((\smile) \circ bd^{-}{}_{\mathcal{R}}\ \circ \partial_F)\ \varphi$
⟨*proof*⟩

lemma *irbd-irfb-dual*: *Sup-pres* $\varphi \Longrightarrow bd^{-}{}_{\mathcal{R}}\ \varphi = ((\smile) \circ fb^{-}{}_{\mathcal{R}} \circ \partial_F)\ \varphi$
⟨*proof*⟩

lemma *rfb-set*: $fb_{\mathcal{R}}\ R\ Y = \{x.\ \forall\, y.\ (x,y) \in R \longrightarrow y \in Y\}$
⟨*proof*⟩

lemma *rfb-rbd-galois*: $(bd_{\mathcal{R}}\ R) \dashv (fb_{\mathcal{R}}\ R)$
⟨*proof*⟩

lemma *irfb-set*: $fb^{-}{}_{\mathcal{R}}\ \varphi = \{(x,\ y).\ \forall\, Y.\ x \in \varphi\ Y \longrightarrow y \in Y\}$
⟨*proof*⟩

lemma *irfb-inv1* [*simp*]: $fb^{-}{}_{\mathcal{R}} \circ fb_{\mathcal{R}} = id$
⟨*proof*⟩

lemma *irfb-inv2*: *Inf-pres* $\varphi \Longrightarrow (fb_{\mathcal{R}} \circ fb^{-}{}_{\mathcal{R}})\ \varphi = \varphi$
⟨*proof*⟩

lemma *rfb-inj*: *inj* $fb_{\mathcal{R}}$
⟨*proof*⟩

lemma *rfb-inj-iff*: $(fb_{\mathcal{R}}\ R = fb_{\mathcal{R}}\ S) = (R = S)$
⟨*proof*⟩

lemma *irfb-inj*: *Inf-pres* $\varphi \Longrightarrow$ *Inf-pres* $\psi \Longrightarrow fb^{-}{}_{\mathcal{R}}\ \varphi = fb^{-}{}_{\mathcal{R}}\ \psi \Longrightarrow \varphi = \psi$
⟨*proof*⟩

lemma *irfb-inf-iff*: *Inf-pres* $\varphi \Longrightarrow$ *Inf-pres* $\psi \Longrightarrow (fb^{-}{}_{\mathcal{R}}\ \varphi = fb^{-}{}_{\mathcal{R}}\ \psi) = (\varphi = \psi)$
⟨*proof*⟩

lemma *rfb-surj*: *Inf-pres* $\varphi \Longrightarrow (\exists\, R.\ fb_{\mathcal{R}}\ R = \varphi)$
⟨*proof*⟩

lemma *irfb-surj*: *surj* $fb^{-}{}_{\mathcal{R}}$
⟨*proof*⟩

lemma *rfb-irfb-galois*: *Inf-pres* $\varphi \Longrightarrow (fb^{-}{}_{\mathcal{R}}\ \varphi = R) = (fb_{\mathcal{R}}\ R = \varphi)$
⟨*proof*⟩

lemma *rfb-comp-pres*: $fb_{\mathcal{R}}\ (R\ ;\ S) = fb_{\mathcal{R}}\ R \circ fb_{\mathcal{R}}\ S$
⟨*proof*⟩

lemma *rfb-Id-pres* [*simp*]: $fb_{\mathcal{R}}\ Id = id$
⟨*proof*⟩

lemma *rfb-Sup-dual*: *Sup-dual* $fb_{\mathcal{R}}$
⟨*proof*⟩

lemma *rfb-Sup-dual-var*: $fb_{\mathcal{R}}\ (\bigsqcup \varphi) = \bigsqcap (\mathcal{P}\ fb_{\mathcal{R}})\ \varphi$
⟨*proof*⟩

lemma *rfb-sup-dual*: *sup-dual* $fb_{\mathcal{R}}$
⟨*proof*⟩

lemma *inf-dual* $fb_{\mathcal{R}}$
⟨*proof*⟩

lemma *rfb-Inf-pres*: *Inf-pres* $(fb_{\mathcal{R}}\ R)$
⟨*proof*⟩

lemma *rfb-inf-pres*: *inf-pres* $(fb_{\mathcal{R}}\ R)$
⟨*proof*⟩

lemma *rfb-zero-pres* [*simp*]: $fb_{\mathcal{R}}\ \{\}\ X = UNIV$
⟨*proof*⟩

lemma *rfb-zero-pres2* [*simp*]: $fb_{\mathcal{R}}\ R\ \{\} = -\ Domain\ R$
⟨*proof*⟩

lemma *rfb-univ* [*simp*]: $fb_{\mathcal{R}}\ R\ UNIV = UNIV$
⟨*proof*⟩

lemma *rfb-iso*: $X \subseteq Y \Longrightarrow fb_{\mathcal{R}}\ R\ X \subseteq fb_{\mathcal{R}}\ R\ Y$
⟨*proof*⟩

lemma *irfb-comp-pres*:
assumes *Inf-pres* φ
assumes *Inf-pres* ψ
shows $fb^{-}{}_{\mathcal{R}}\ (\varphi \circ \psi) = fb^{-}{}_{\mathcal{R}}\ \varphi\ ;\ fb^{-}{}_{\mathcal{R}}\ \psi$
⟨*proof*⟩

lemma *irfb-id-pres* [*simp*]: $fb^{-}{}_{\mathcal{R}}\ id = Id$
⟨*proof*⟩

lemma *irfb-Sup-dual*: *Sup-dual* $fb^{-}{}_{\mathcal{R}}$
⟨*proof*⟩

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*⟩

lemma *irfb-sup-dual*: *sup-dual* $fb^{-}{}_{\mathcal{R}}$
⟨*proof*⟩

lemma *irfb-inf-dual*:
assumes *Inf-pres* φ
and *Inf-pres* ψ
shows $fb^-_{\mathcal{R}}\ (\varphi \sqcap \psi) = fb^-_{\mathcal{R}}\ \varphi \sqcup fb^-_{\mathcal{R}}\ \psi$
⟨*proof*⟩

lemma *irfb-top-pres* [*simp*]: $bd^-_{\mathcal{R}}\ \top = UNIV$
⟨*proof*⟩

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

lemma *ffb-fbd-galois-var*: $(bd_{\mathcal{F}}\ f\ X \subseteq Y) = (X \subseteq fb_{\mathcal{F}}\ f\ Y)$
⟨*proof*⟩

lemma *rfb-rbd-galois-var*: $(bd_{\mathcal{R}}\ R\ X \subseteq Y) = (X \subseteq fb_{\mathcal{R}}\ R\ Y)$
⟨*proof*⟩

lemma *ffb-fbd*: $fb_{\mathcal{F}}\ f\ Y = \bigcup\{X.\ bd_{\mathcal{F}}\ f\ X \subseteq Y\}$
⟨*proof*⟩

lemma *rfb-rbd*: $fb_{\mathcal{R}}\ R\ Y = \bigcup\{X.\ bd_{\mathcal{R}}\ R\ X \subseteq Y\}$
⟨*proof*⟩

lemma *fbd-ffb*: $bd_{\mathcal{F}}\ f\ X = \bigcap\{Y.\ X \subseteq fb_{\mathcal{F}}\ f\ Y\}$
⟨*proof*⟩

lemma *rbd-rfb*: $bd_{\mathcal{R}}\ R\ X = \bigcap\{Y.\ X \subseteq fb_{\mathcal{R}}\ R\ Y\}$
⟨*proof*⟩

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

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

definition *irfd* :: $('b\ set \Rightarrow 'a\ set) \Rightarrow ('a \times 'b)\ set$ ($\langle fd^-{}_{\mathcal{R}}\rangle$) **where**

$fd^-_{\mathcal{R}} = \mathcal{R} \circ fd^-_{\mathcal{F}}$

Second, I introduce forward boxes and their inverses.

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

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

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

definition *irbb* :: $('a\ \ set \Rightarrow 'b\ set) \Rightarrow ('a \times 'b)\ set\ (\langle bb^-_{\mathcal{R}} \rangle)$ **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*: $fd_{\mathcal{R}} = bd_{\mathcal{R}} \circ (\smile)$
⟨*proof*⟩

lemma *irfd-irbd*: $fd^-_{\mathcal{R}} = (\smile) \circ bd^-_{\mathcal{R}}$
⟨*proof*⟩

lemma *fbd-ffd*: $bd_{\mathcal{F}} = fd_{\mathcal{F}} \circ op_K$
⟨*proof*⟩

lemma *rbb-rfb*: $bb_{\mathcal{R}} = fb_{\mathcal{R}} \circ (\smile)$
⟨*proof*⟩

lemma *irbb-irfb*: $bb^-_{\mathcal{R}} = (\smile) \circ fb^-_{\mathcal{R}}$
⟨*proof*⟩

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

lemma *ffd-ffb-demorgan*: $\partial \circ fd_{\mathcal{F}}\ f = fb_{\mathcal{F}}\ f \circ \partial$
⟨*proof*⟩

lemma *iffd-iffb-demorgan*: *Sup-pres* $\varphi \Longrightarrow fd^-_{\mathcal{F}}\ \varphi = (fb^-_{\mathcal{F}} \circ \partial_F)\ \varphi$
⟨*proof*⟩

lemma *ffb-ffd-demorgan*: $\partial \circ fb_{\mathcal{F}}\ f = fd_{\mathcal{F}}\ f \circ \partial$
⟨*proof*⟩

lemma *iffb-iffd-demorgan*: *Inf-pres* $\varphi \Longrightarrow fb^-_{\mathcal{F}}\ \varphi = (fd^-_{\mathcal{F}} \circ \partial_F)\ \varphi$
⟨*proof*⟩

lemma *rfd-rfb-demorgan*: $\partial \circ fd_{\mathcal{R}}\ R = fb_{\mathcal{R}}\ R \circ \partial$
⟨*proof*⟩

lemma *irfd-irfb-demorgan*: *Sup-pres* $\varphi \Longrightarrow fd^-_{\mathcal{R}}\ \varphi = (fb^-_{\mathcal{R}} \circ \partial_F)\ \varphi$
⟨*proof*⟩

lemma *rfb-rfd-demorgan*: $\partial \circ fb_{\mathcal{R}}\ R = fd_{\mathcal{R}}\ R \circ \partial$
⟨*proof*⟩

lemma *irfb-irfd-demorgan*: *Inf-pres* $\varphi \Longrightarrow fb^-_{\mathcal{R}}\ \varphi\ = (fd^-_{\mathcal{R}} \circ \partial_F)\ \varphi$
⟨*proof*⟩

lemma *fbd-fbb-demorgan*: $\partial \circ bd_{\mathcal{F}}\ f = bb_{\mathcal{F}}\ f \circ \partial$
⟨*proof*⟩

lemma *ifbd-ifbb-demorgan*: *Sup-pres* $\varphi \Longrightarrow bd^-_{\mathcal{F}}\ \varphi = (bb^-_{\mathcal{F}} \circ \partial_F)\ \varphi$
⟨*proof*⟩

lemma *fbb-fbd-demorgan*: $\partial \circ bb_{\mathcal{F}}\ R = bd_{\mathcal{F}}\ R \circ \partial$
⟨*proof*⟩

lemma *ifbb-ifbd-demorgan*: *Inf-pres* $\varphi \Longrightarrow bb^-_{\mathcal{F}}\ \varphi = (bd^-_{\mathcal{F}} \circ \partial_F)\ \varphi$
⟨*proof*⟩

lemma *rbd-rbb-demorgan*: $\partial \circ bd_{\mathcal{R}}\ R = bb_{\mathcal{R}}\ R \circ \partial$
⟨*proof*⟩

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

lemma *rbb-rbd-demorgan*: $\partial \circ bb_{\mathcal{R}}\ R = bd_{\mathcal{R}}\ R \circ \partial$
⟨*proof*⟩

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

Further symmetries arise by combination.

lemma *ffd-fbb-dual*: $\partial \circ fd_{\mathcal{F}}\ f = bb_{\mathcal{F}}\ (op_K\ f) \circ \partial$
⟨*proof*⟩

lemma *iffd-ifbb-dual*: *Sup-pres* $\varphi \Longrightarrow fd^-_{\mathcal{F}}\ \varphi = (op_K \circ bb^-_{\mathcal{F}} \circ \partial_F)\ \varphi$
⟨*proof*⟩

lemma *fbb-ffd-dual*: $\partial \circ bb_{\mathcal{F}}\ f = fd_{\mathcal{F}}\ (op_K\ f) \circ \partial$
⟨*proof*⟩

lemma *ifbb-iffd-dual*: *Inf-pres* $\varphi \Longrightarrow bb^-_{\mathcal{F}}\ \varphi = (op_K \circ fd^-_{\mathcal{F}} \circ \partial_F)\ \varphi$
⟨*proof*⟩

lemma *rfd-rbb-dual*: $\partial \circ fd_{\mathcal{R}}\ R = bb_{\mathcal{R}}\ (R^{-1}) \circ \partial$
⟨*proof*⟩

lemma *ifd-ibb-dual*: *Sup-pres* $\varphi \Longrightarrow fd^-{}_{\mathcal{R}}\ \varphi = ((\smile) \circ bb^-{}_{\mathcal{R}} \circ \partial_F)\ \varphi$
⟨*proof*⟩

lemma *rbb-rfd-dual*: $\partial \circ bb_{\mathcal{R}}\ R = fd_{\mathcal{R}}\ (R^{-1}) \circ \partial$
⟨*proof*⟩

lemma *irbb-irfd-dual*: *Inf-pres* $\varphi \Longrightarrow bb^-{}_{\mathcal{R}}\ \varphi = ((\smile) \circ fd^-{}_{\mathcal{R}} \circ \partial_F)\ \varphi$
⟨*proof*⟩

lemma *ffd-iffd-galois*: *Sup-pres* $\varphi \Longrightarrow (\varphi = fd_{\mathcal{F}}\ f) = (f = fd^-{}_{\mathcal{F}}\ \varphi)$
⟨*proof*⟩

lemma *rfd-irfd-galois*: *Sup-pres* $\varphi \Longrightarrow (\varphi = fd_{\mathcal{R}}\ R) = (R = fd^-{}_{\mathcal{R}}\ \varphi)$
⟨*proof*⟩

lemma *fbb-ifbb-galois*: *Inf-pres* $\varphi \Longrightarrow (\varphi = bb_{\mathcal{F}}\ f) = (f = bb^-{}_{\mathcal{F}}\ \varphi)$
⟨*proof*⟩

lemma *rbb-irbb-galois*: *Inf-pres* $\varphi \Longrightarrow (\varphi = bb_{\mathcal{R}}\ R) = (R = bb^-{}_{\mathcal{R}}\ \varphi)$
⟨*proof*⟩

Next I spell out the missing adjunctions.

lemma *ffd-ffb-adj*: $fd_{\mathcal{F}}\ f \dashv bb_{\mathcal{F}}\ f$
⟨*proof*⟩

lemma *ffd-fbb-galois*: $(fd_{\mathcal{F}}\ f\ X \subseteq Y) = (X \subseteq bb_{\mathcal{F}}\ f\ Y)$
⟨*proof*⟩

lemma *rfd-rfb-adj*: $fd_{\mathcal{R}}\ f \dashv bb_{\mathcal{R}}\ f$
⟨*proof*⟩

lemma *rfd-rbb-galois*: $(fd_{\mathcal{R}}\ R\ X \subseteq Y) = (X \subseteq bb_{\mathcal{R}}\ R\ Y)$
⟨*proof*⟩

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*⟩

lemma *rfd-rbd-conjugation*: $((fd_{\mathcal{R}}\ R\ X) \cap Y = \{\}) = (X \cap (bd_{\mathcal{R}}\ R\ Y) = \{\})$
⟨*proof*⟩

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

⟨*proof*⟩

lemma *rfb-rbb-conjugation*: $((fb_{\mathcal{R}}\ R\ X) \cup Y = UNIV) = (X \cup (bb_{\mathcal{R}}\ R\ Y) = UNIV)$

⟨*proof*⟩

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*: $((f::'a \Rightarrow 'b\ set) \sqcup g) \sqcup h = f \sqcup (g \sqcup h)$
⟨*proof*⟩

lemma *ksup-comm*: $(f::'a => 'b\ set) \sqcup g = g \sqcup f$
⟨*proof*⟩

lemma *ksup-idem* [*simp*]: $(f::'a \Rightarrow 'b\ set) \sqcup f = f$
⟨*proof*⟩

lemma *kcomp-distl*: $f \circ_K (g \sqcup h) = (f \circ_K g) \sqcup (f \circ_K h)$
⟨*proof*⟩

lemma *kcomp-distr*: $(f \sqcup g) \circ_K h = (f \circ_K h) \sqcup (g \circ_K h)$
⟨*proof*⟩

lemma *ksup-zerol* [*simp*]: $\zeta \sqcup f = f$
⟨*proof*⟩

lemma *ksup-annil* [*simp*]: $\zeta \circ_K f = \zeta$
⟨*proof*⟩

lemma *ksup-annir* [*simp*]: $f \circ_K \zeta = \zeta$
⟨*proof*⟩

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*⟩

lemma *kSup-distr*: $(\bigsqcup F) \circ_K g = (\bigsqcup f \in F.\ f \circ_K g)$
⟨*proof*⟩

lemma *kcomp-isol*: $f \leq g \Longrightarrow h \circ_K f \leq h \circ_K g$
⟨*proof*⟩

lemma *kcomp-isor*: $f \leq g \Longrightarrow f \circ_K h \leq g \circ_K h$
⟨*proof*⟩

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*: $\mathcal{F}\ (R\ ^{\frown\frown}\ i) = kpower\ (\mathcal{F}\ R)\ i$
⟨*proof*⟩

lemma *f2r-kpower*: $\mathcal{R}\ (kpower\ f\ i) = (\mathcal{R}\ f)\ ^{\frown\frown}\ i$
⟨*proof*⟩

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

lemma *r2f-rtrancl-hom*: $\mathcal{F}\ (rtrancl\ R) = kstar\ (\mathcal{F}\ R)$
⟨*proof*⟩

lemma *r2f-rtrancl-hom-var*: $\mathcal{F} \circ rtrancl = kstar \circ \mathcal{F}$
⟨*proof*⟩

lemma *f2r-kstar-hom*: $\mathcal{R}\ (kstar\ f) = rtrancl\ (\mathcal{R}\ f)$
⟨*proof*⟩

lemma *f2r-kstar-hom-var*: $\mathcal{R} \circ kstar = rtrancl \circ \mathcal{R}$
⟨*proof*⟩

lemma *kstar-unfoldl-eq*: $\eta \sqcup f \circ_K kstar\ f = kstar\ f$
⟨*proof*⟩

lemma *kstar-unfoldl*: $\eta \sqcup f \circ_K \quad kstar\ f \leq kstar\ f$
⟨*proof*⟩

lemma *kstar-unfoldr-eq*: $\eta \sqcup (kstar\ f) \circ_K f = kstar\ f$
⟨*proof*⟩

lemma *kstar-unfoldr*: $\eta \sqcup (kstar\ f) \circ_K f \leq kstar\ f$

⟨*proof*⟩

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

lemma *kpower-inductl*: $f \circ_K g \leq g \Longrightarrow kpower\ f\ i \circ_K g \leq g$
⟨*proof*⟩

lemma *kpower-inductl-var*: $h \sqcup f \circ_K g \leq g \Longrightarrow kpower\ f\ i \circ_K h \leq g$
⟨*proof*⟩

lemma *kstar-inductl*: $h \sqcup f \circ_K g \leq g \Longrightarrow kstar\ f \circ_K h \leq g$
⟨*proof*⟩

lemma *kpower-inductr*: $g \circ_K f \leq g \Longrightarrow g \circ_K kpower\ f\ i \leq g$
⟨*proof*⟩

lemma *kpower-inductr-var*: $h \sqcup g \circ_K f \leq g \Longrightarrow h \circ_K kpower\ f\ i \leq g$
⟨*proof*⟩

lemma *kstar-inductr*: $h \sqcup g \circ_K f \leq g \Longrightarrow h \circ_K kstar\ f \leq g$
⟨*proof*⟩

lemma *kpower-prop*: $f \leq \eta \Longrightarrow kpower\ f\ i \leq \eta$
⟨*proof*⟩

lemma *kstar-prop*: $f \leq \eta \Longrightarrow kstar\ f \leq \eta$
⟨*proof*⟩

6.2 Antidomain

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

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

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

lemma *f2r-ad-fun-hom*: $\mathcal{R}\ (kad\ f) = ad\text{-}rel\ (\mathcal{R}\ f)$
⟨*proof*⟩

lemma *f2r-ad-fun-hom-var*:$\mathcal{R} \circ kad = ad\text{-}rel \circ \mathcal{R}$
⟨*proof*⟩

lemma *r2f-ad-rel-hom*: $\mathcal{F}\ (ad\text{-}rel\ R) = kad\ (\mathcal{F}\ R)$
⟨*proof*⟩

lemma *r2f-ad-rel-hom-var*:$\mathcal{F} \circ ad\text{-}rel = kad \circ \mathcal{F}$
⟨*proof*⟩

lemma *ad-fun-as1* [*simp*]: $(kad\ f) \circ_K f = \zeta$
⟨*proof*⟩

lemma *ad-fun-as2* [*simp*]: $kad\ (f \circ_K g) \sqcup kad\ (f \circ_K kad\ (kad\ g)) = kad\ (f \circ_K kad\ (kad\ g))$
⟨*proof*⟩

lemma *ad-fun-as3* [*simp*]: $kad\ (kad\ f) \sqcup kad\ f = \eta$
⟨*proof*⟩

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

definition $p2fun = set2fun \circ Collect$

lemma *ffb-ad-fun*: $fb_{\mathcal{F}}\ f\ X = \{x.\ (kad\ (f \circ_K kad\ (set2fun\ X)))\ x \neq \{\}\}$
⟨*proof*⟩

lemma *ffb-ad-fun2*: $set2fun\ (fb_{\mathcal{F}}\ f\ X) = kad\ (f \circ_K kad\ (set2fun\ X))$
⟨*proof*⟩

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

lemma *fb-ad-rel*: $fb_{\mathcal{R}}\ R\ X = Domain\ (ad\text{-}rel\ (R\ ;\ ad\text{-}rel\ (Id\text{-}on\ X)))$
⟨*proof*⟩

lemma *fb-ad-rel2*: $Id\text{-}on\ (fb_{\mathcal{R}}\ R\ X) = ad\text{-}rel\ (R\ ;\ ad\text{-}rel\ (Id\text{-}on\ X))$
⟨*proof*⟩

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 $'a\ nd\text{-}fun = \{f::'a \Rightarrow 'a\ set.\ f \in UNIV\}$
⟨*proof*⟩

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* ∘ $\mathcal{F}\langle proof\rangle$

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

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

lemma *r2f2r-inv*: *r2fnd* ∘ *f2rnd* = *id*
$\langle proof\rangle$

lemma *f2r2f-inv*: *f2rnd* ∘ *r2fnd* = *id*
$\langle proof\rangle$

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

lift-definition *one-nd-fun* :: *'a nd-fun* **is** *Abs-nd-fun* $\eta\langle proof\rangle$

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

instance
$\langle proof\rangle$

end

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

lift-definition *less-eq-nd-fun* :: *'a nd-fun* ⇒ *'a nd-fun* ⇒ *bool* **is** $\lambda f\ g.$ *Rep-nd-fun f* ≤ *Rep-nd-fun g*$\langle proof\rangle$

lift-definition *less-nd-fun* :: *'a nd-fun* ⇒ *'a nd-fun* ⇒ *bool* **is** $\lambda f\ g.$ *Rep-nd-fun f* ≤ *Rep-nd-fun g* ∧ *f* ≠ *g*$\langle proof\rangle$

instance
$\langle proof\rangle$

end

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

lift-definition *Sup-nd-fun* :: *'a nd-fun set* ⇒ *'a nd-fun* **is** *Abs-nd-fun* ∘ *Sup* ∘ $\mathcal{P}$ *Rep-nd-fun*$\langle proof\rangle$

instance
$\langle proof\rangle$

end

lemma *Abs-comp-hom*: *Abs-nd-fun* $(f \circ_K g) =$ *Abs-nd-fun* $f \cdot$ *Abs-nd-fun* g
⟨*proof*⟩

lemma *Rep-comp-hom*: *Rep-nd-fun* $(f \cdot g) =$ *Rep-nd-fun* $f \circ_K$ *Rep-nd-fun* g
⟨*proof*⟩

instance *nd-fun* :: (*type*) *unital-Sup-quantale*
⟨*proof*⟩

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* ∘ $\mathcal{P}$ *Rep-nd-fun*⟨*proof*⟩

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

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*)⟨*proof*⟩

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

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*)⟨*proof*⟩

instance
⟨*proof*⟩

end

instance *nd-fun* :: (*type*) *unital-quantale*
⟨*proof*⟩

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

lemma *fun-star-unfoldl-eq*: (*1*::*'a nd-fun*) ⊔ $f \cdot$ *qstar f* = *qstar f*
⟨*proof*⟩

lemma *fun-star-unfoldl*: (*1*::*'a nd-fun*) ⊔ $f \cdot$ *qstar f* ≤ *qstar f*
⟨*proof*⟩

lemma *fun-star-unfoldr-eq*: (*1*::*'a nd-fun*) ⊔ (*qstar f*) $\cdot f =$ *qstar f*
⟨*proof*⟩

lemma *fun-star-unfoldr*: (*1*::*'a nd-fun*) ⊔ *qstar f* $\cdot f \leq$ *qstar f*

⟨*proof*⟩

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

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

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.