Relation Algebra

Alasdair Armstrong, Simon Foster, Georg Struth, Tjark Weber

March 17, 2025

Abstract

Tarski's algebra of binary relations is formalised along the lines of the standard textbooks of Maddux and Schmidt and Ströhlein. This includes relation-algebraic concepts such as subidentities, vectors and a domain operation as well as various notions associated to functions. Relation algebras are also expanded by a reflexive transitive closure operation, and they are linked with Kleene algebras and models of binary relations and Boolean matrices.

Contents

1 Introductory Remarks **2**

2 (More) Boolean Algebra **2**
2.1 Laws of Boolean Algebra 2
2.2 Boolean Algebras with Operators 4

3 Relation Algebra **8**

4 Vectors **13**

5 Tests **14**
5.1 Tests 15
5.2 Test Complements 17

6 Functions **18**
6.1 Functions 18
6.2 Points and Rectangles 23
6.3 Antidomain 23

7 Direct Products **25**

8 Reflexive Transitive Closure **29**

9 Models of Relation Algebra **30**
9.1 Binary Relations 30
9.2 Infinite Boolean Matrices 30

1 Introductory Remarks

These theory files are only sparsely commented. Background information can be found in Tarski's original article [4] and in the books by Maddux [2] and Schmidt and Ströhlein [3]. We briefly discuss proof automation and the formalisation of direct products in [1].

2 (More) Boolean Algebra

theory *More-Boolean-Algebra*
imports *Main*
begin

2.1 Laws of Boolean Algebra

The following laws of Boolean algebra support relational proofs. We might add laws for the binary minus since that would make certain theorems look more nicely. These are currently not so well supported.

context *boolean-algebra*
begin

no-notation
times (**infixl** ‹·› *70*)
and *plus* (**infixl** ‹+› *65*)
and *Groups.zero-class.zero* (‹*0*›)
and *Groups.one-class.one* (‹*1*›)

notation
inf (**infixl** ‹·› *70*)
and *sup* (**infixl** ‹+› *65*)
and *bot* (‹*0*›)
and *top* (‹*1*›)

lemma *meet-assoc*: $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
by (*metis inf-assoc*)

lemma *aux4* [*simp*]: $x \cdot y + x \cdot -y = x$
by (*metis inf-sup-distrib1 inf-top-right sup-compl-top*)

lemma *aux4-comm* [*simp*]: $x \cdot -y + x \cdot y = x$
by (*metis aux4 sup.commute*)

lemma *aux6* [*simp*]: $(x + y) \cdot -x = y \cdot -x$
by (*metis inf-compl-bot inf-sup-distrib2 sup-bot-left*)

lemma *aux6-var* [*simp*]: $(-x + y) \cdot x = x \cdot y$
by (*metis compl-inf-bot inf-commute inf-sup-distrib2 sup-bot-left*)

lemma *aux9* [*simp*]: $x + -x \cdot y = x + y$
by (*metis aux4 aux6 inf.commute inf-sup-absorb*)

lemma *join-iso*: $x \leq y \Longrightarrow x + z \leq y + z$
by (*metis eq-refl sup-mono*)

lemma *join-isol*: $x \leq y \Longrightarrow z + x \leq z + y$
by (*metis join-iso sup.commute*)

lemma *join-double-iso*: $x \leq y \Longrightarrow w + x + z \leq w + y + z$
by (*metis le-iff-inf sup-inf-distrib1 sup-inf-distrib2*)

lemma *comp-anti*: $x \leq y \longleftrightarrow -y \leq -x$
by (*metis compl-le-swap2 double-compl*)

lemma *meet-iso*: $x \leq y \Longrightarrow x \cdot z \leq y \cdot z$
by (*metis eq-refl inf-mono*)

lemma *meet-isor*: $x \leq y \Longrightarrow z \cdot x \leq z \cdot y$
by (*metis inf.commute meet-iso*)

lemma *meet-double-iso*: $x \leq y \Longrightarrow w \cdot x \cdot z \leq w \cdot y \cdot z$
by (*metis meet-iso meet-isor*)

lemma *de-morgan-3* [*simp*]: $-(-x \cdot -y) = x + y$
by (*metis compl-sup double-compl*)

lemma *subdist-2-var*: $x + y \cdot z \leq x + y$
by (*metis eq-refl inf-le1 sup-mono*)

lemma *dist-alt*: $[\![x + z = y + z; x \cdot z = y \cdot z]\!] \Longrightarrow x = y$
by (*metis aux4 aux6 sup.commute*)

Finally we prove the Galois connections for complementation.

lemma *galois-aux*: $x \cdot y = 0 \longleftrightarrow x \leq -y$
by (*metis aux6 compl-sup double-compl inf.commute le-iff-inf sup-bot-right sup-compl-top*)

lemma *galois-aux2*: $x \cdot -y = 0 \longleftrightarrow x \leq y$
by (*metis double-compl galois-aux*)

lemma *galois-1*: $x \cdot -y \leq z \longleftrightarrow x \leq y + z$
apply (*rule iffI*)
apply (*metis inf-le2 join-iso le-iff-sup le-supE join-isol aux4*)

apply (*metis meet-iso aux6 le-infE*)
done

lemma *galois-2*: $x \leq y + -z \longleftrightarrow x \cdot z \leq y$
apply (*rule iffI*)
apply (*metis compl-sup double-compl galois-1 inf.commute*)
apply (*metis inf.commute order-trans subdist-2-var aux4 join-iso*)
done

lemma *galois-aux3*: $x + y = 1 \longleftrightarrow -x \leq y$
by (*metis galois-1 inf-top-left top-unique*)

lemma *galois-aux4*: $-x + y = 1 \longleftrightarrow x \leq y$
by (*metis double-compl galois-aux3*)

2.2 Boolean Algebras with Operators

We follow Jónsson and Tarski to define pairs of conjugate functions on Boolean algebras. We also consider material from Maddux's article. This gives rise to a Galois connection and the notion of Boolean algebras with operators.

We do not explicitly define families of functions over Boolean algebras as a type class.

This development should certainly be expanded do deal with complete Boolean algebras one the one hand and other lattices on the other hand.

Boolean algebras with operators and their variants can be applied in various ways. The prime example are relation algebras. The modular laws, for instance, can be derived by instantiation. Other applications are antidomain semirings where modal operators satisfy conjugations and Galois connections, and algebras of predicate transformers.

We define conjugation as a predicate which holds if a pair of functions are conjugates.

definition *is-conjugation* :: $('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a) \Rightarrow bool$
where *is-conjugation f g* $\equiv (\forall x\ y\ .\ f\ x \cdot y = 0 \longleftrightarrow x \cdot g\ y = 0)$

We now prove the standard lemmas. First we show that conjugation is symmetric and that conjugates are uniqely defined.

lemma *is-conjugation-sym*: *is-conjugation f g* $\longleftrightarrow$ *is-conjugation g f*
by (*metis inf.commute is-conjugation-def*)

lemma *is-conjugation-unique*: $[\![$*is-conjugation f g*; *is-conjugation f h*$]\!] \Longrightarrow g = h$
by (*metis galois-aux inf.commute double-compl order.eq-iff ext is-conjugation-def*)

Next we show that conjugates give rise to adjoints in a Galois connection.

lemma *conj-galois-1*:

assumes *is-conjugation f g*
shows $f\ x \leq y \longleftrightarrow x \leq -g\ (-y)$
by (*metis assms is-conjugation-def double-compl galois-aux*)

lemma *conj-galois-2*:
assumes *is-conjugation f g*
shows $g\ x \leq y \longleftrightarrow x \leq -f\ (-y)$
by (*metis assms is-conjugation-sym conj-galois-1*)

Now we prove some of the standard properties of adjoints and conjugates. In fact, conjugate functions even distribute over all existing suprema. We display the next proof in detail because it is elegant.

lemma *f-pre-additive*:
assumes *is-conjugation f g*
shows $f\ (x + y) \leq z \longleftrightarrow f\ x + f\ y \leq z$
proof −
have $f\ (x + y) \leq z \longleftrightarrow x + y \leq -g\ (-z)$
by (*metis assms conj-galois-1*)
also have $\ldots \longleftrightarrow x \leq -g\ (-z) \wedge y \leq -g\ (-z)$
by (*metis le-sup-iff*)
also have $\ldots \longleftrightarrow f\ x \leq z \wedge f\ y \leq z$
by (*metis assms conj-galois-1*)
thus *?thesis*
by (*metis le-sup-iff calculation*)
qed

lemma *f-additive*:
assumes *is-conjugation f g*
shows $f\ (sup\ x\ y) = sup\ (f\ x)\ (f\ y)$
by (*metis assms order.eq-iff f-pre-additive*)

lemma *g-pre-additive*:
assumes *is-conjugation f g*
shows $g\ (sup\ x\ y) \leq z \longleftrightarrow sup\ (g\ x)\ (g\ y) \leq z$
by (*metis assms is-conjugation-sym f-pre-additive*)

lemma *g-additive*:
assumes *is-conjugation f g*
shows $g\ (sup\ x\ y) = sup\ (g\ x)\ (g\ y)$
by (*metis assms is-conjugation-sym f-additive*)

Additivity of adjoints obviously implies their isotonicity.

lemma *f-iso*:
assumes *is-conjugation f g*
shows $x \leq y \longrightarrow f\ x \leq f\ y$
by (*metis assms f-additive le-iff-sup*)

lemma *g-iso*:
assumes *is-conjugation f g*

shows $x \leq y \longrightarrow g\ x \leq g\ y$
by (*metis assms is-conjugation-sym f-iso*)

lemma *f-subdist*:
assumes *is-conjugation f g*
shows $f\ (x \cdot y) \leq f\ x$
by (*metis assms f-iso inf-le1*)

lemma *g-subdist*:
assumes *is-conjugation f g*
shows $g\ (x \cdot y) \leq g\ x$
by (*metis assms g-iso inf-le1*)

Next we prove cancellation and strictness laws.

lemma *cancellation-1*:
assumes *is-conjugation f g*
shows $f\ (-g\ x) \leq -x$
by (*metis assms conj-galois-1 double-compl eq-refl*)

lemma *cancellation-2*:
assumes *is-conjugation f g*
shows $g\ (-f\ x) \leq -x$
by (*metis assms is-conjugation-sym cancellation-1*)

lemma *f-strict*:
assumes *is-conjugation f g*
shows $f\ 0 = 0$
by (*metis assms inf.idem inf-bot-left is-conjugation-def*)

lemma *g-strict*:
assumes *is-conjugation f g*
shows $g\ 0 = 0$
by (*metis assms is-conjugation-sym f-strict*)

The following variants of modular laws have more concrete counterparts in relation algebra.

lemma *modular-1-aux*:
assumes *is-conjugation f g*
shows $f\ (x \cdot -g\ y) \cdot y = 0$
by (*metis assms galois-aux inf-le2 is-conjugation-def*)

lemma *modular-2-aux*:
assumes *is-conjugation f g*
shows $g\ (x \cdot -f\ y) \cdot y = 0$
by (*metis assms is-conjugation-sym modular-1-aux*)

lemma *modular-1*:
assumes *is-conjugation f g*
shows $f\ x \cdot y = f\ (x \cdot g\ y) \cdot y$

proof −
have $f\ x \cdot y = f\ (x \cdot g\ y + x \cdot -g\ y) \cdot y$
by (*metis aux4*)
hence $f\ x \cdot y = (f\ (x \cdot g\ y) + f\ (x \cdot -g\ y)) \cdot y$
by (*metis assms f-additive*)
hence $f\ x \cdot y = f\ (x \cdot g\ y) \cdot y + f\ (x \cdot -g\ y) \cdot y$
by (*metis inf.commute inf-sup-distrib1*)
thus *?thesis*
by (*metis assms modular-1-aux sup-bot-right*)
qed

lemma *modular-2*:
assumes *is-conjugation f g*
shows $g\ x \cdot y = g\ (x \cdot f\ y) \cdot y$
by (*metis assms is-conjugation-sym modular-1*)

lemma *conjugate-eq-aux*:
is-conjugation f g $\Longrightarrow f\ (x \cdot -g\ y) \leq f\ x \cdot -y$
by (*metis f-subdist galois-aux le-inf-iff modular-1-aux*)

lemma *conjugate-eq*:
is-conjugation f g $\longleftrightarrow (\forall\, x\ y.\ f\ (x \cdot -g\ y) \leq f\ x \cdot -y \wedge g\ (y \cdot -f\ x) \leq g\ y \cdot -x)$
(**is** *?l* $\longleftrightarrow$ *?r*)
proof
assume *?l* **thus** *?r*
by (*metis is-conjugation-sym conjugate-eq-aux*)
next
assume *r*: *?r*
have $\forall\, x\ y.\ f\ x \cdot y = 0 \longrightarrow x \cdot g\ y = 0$
by (*metis aux4 inf.left-commute inf-absorb1 inf-compl-bot inf-left-idem sup-bot-left r*)
hence $\forall\, x\ y.\ x \cdot g\ y = 0 \longleftrightarrow f\ x \cdot y = 0$
by (*metis aux4 inf.commute inf.left-commute inf-absorb1 inf-compl-bot sup-commute sup-inf-absorb r*)
thus *is-conjugation f g*
by (*metis is-conjugation-def*)
qed

lemma *conjugation-prop1*: *is-conjugation f g* $\Longrightarrow f\ y \cdot z \leq f\ (y \cdot g\ z)$
by (*metis le-infE modular-1 order-refl*)

lemma *conjugation-prop2*: *is-conjugation f g* $\Longrightarrow g\ z \cdot y \leq g\ (z \cdot f\ y)$
by (*metis is-conjugation-sym conjugation-prop1*)

end

end

3 Relation Algebra

theory *Relation-Algebra*
  imports *More-Boolean-Algebra Kleene-Algebra.Kleene-Algebra*
begin

We follow Tarski's original article and Maddux's book, in particular we use their notation. In contrast to Schmidt and Ströhlein we do not assume that the Boolean algebra is complete and we do not consider the Tarski rule in this development.

A main reason for using complete Boolean algebras seems to be that the Knaster-Tarski fixpoint theorem becomes available for defining notions of iteration. In fact, several chapters of Schmidt and Ströhlein's book deal with iteration.

We capture iteration in an alternative way by linking relation algebras with Kleene algebras (cf. *relation-algebra-rtc*).

class *relation-algebra* = *boolean-algebra* +
  fixes *composition* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** ‹;› *75*)
    and *converse* :: $'a \Rightarrow 'a$ (‹(-$^\smile$)› [*1000*] *999*)
    and *unit* :: $'a$ (‹*1'*›)
  assumes *comp-assoc*: $(x\ ;\ y)\ ;\ z = x\ ;\ (y\ ;\ z)$
    and *comp-unitr* [*simp*]: $x\ ;\ 1' = x$
    and *comp-distr*: $(x + y)\ ;\ z = x\ ;\ z + y\ ;\ z$
    and *conv-invol* [*simp*]: $(x^\smile)^\smile = x$
    and *conv-add* [*simp*]: $(x + y)^\smile = x^\smile + y^\smile$
    and *conv-contrav* [*simp*]: $(x\ ;\ y)^\smile = y^\smile\ ;\ x^\smile$
    and *comp-res*: $x^\smile\ ;\ -(x\ ;\ y) \leq -y$

We first show that every relation algebra is a dioid. We do not yet treat the zero (the minimal element of the boolean reduct) since the proof of the annihilation laws is rather tricky to automate. Following Maddux we derive them from properties of Boolean algebras with operators.

sublocale *relation-algebra* $\subseteq$ *dioid-one* (+) (;) ($\leq$) ($<$) *1'*
proof
  fix *x y z* :: $'a$
  show $x\ ;\ y\ ;\ z = x\ ;\ (y\ ;\ z)$
    by (*fact comp-assoc*)
  show $x + y + z = x + (y + z)$
    by (*metis sup.commute sup.left-commute*)
  show $x + y = y + x$
    by (*fact sup.commute*)
  show $(x + y)\ ;\ z = x\ ;\ z + y\ ;\ z$
    by (*fact comp-distr*)
  show $x + x = x$
    by (*fact sup.idem*)
  show $x\ ;\ (y + z) = x\ ;\ y + x\ ;\ z$
    by (*metis conv-invol conv-add conv-contrav comp-distr*)

show $x\ ;\ 1' = x$
by (*fact comp-unitr*)
show $1'\ ;\ x = x$
by (*metis comp-unitr conv-contrav conv-invol*)
show $x \leq y \longleftrightarrow x + y = y$
by (*metis sup.commute sup-absorb1 sup-ge1*)
show $x < y \longleftrightarrow x \leq y \wedge x \neq y$
by (*fact less-le*)
qed

context *relation-algebra*
begin

First we prove some basic facts about joins and meets.

lemma *meet-interchange*: $(w \cdot x)\ ;\ (y \cdot z) \leq w\ ;\ y \cdot x\ ;\ z$
by (*metis inf-le1 inf-le2 le-infI mult-isol-var*)

lemma *join-interchange*: $w\ ;\ x + y\ ;\ z \leq (w + y)\ ;\ (x + z)$
using *local.mult-isol-var local.sup.bounded-iff local.sup.cobounded2 local.sup-ge1* **by** *presburger*

We now prove some simple facts about conversion.

lemma *conv-iso*: $x \leq y \longleftrightarrow x^\smile \leq y^\smile$
by (*metis conv-add conv-invol le-iff-sup*)

lemma *conv-zero* [*simp*]: $0^\smile = 0$
by (*metis conv-add conv-invol sup-bot-right sup-eq-bot-iff*)

lemma *conv-one* [*simp*]: $1^\smile = 1$
by (*metis conv-add conv-invol sup-top-left sup-top-right*)

lemma *conv-compl-aux*: $(-x)^\smile = (-x)^\smile + (-x^\smile)$
by (*metis aux9 conv-add conv-one double-compl galois-aux4 inf.commute less-eq-def sup.commute sup-top-left*)

lemma *conv-compl*: $(-x)^\smile = -(x^\smile)$
by (*metis add-commute conv-add conv-compl-aux conv-invol*)

lemma *comp-res-aux* [*simp*]: $x^\smile\ ;\ -(x\ ;\ y) \cdot y = 0$
by (*metis comp-res galois-aux*)

lemma *conv-e* [*simp*]: $1'^\smile = 1'$
by (*metis comp-unitr conv-contrav conv-invol*)

lemma *conv-times* [*simp*]: $(x \cdot y)^\smile = x^\smile \cdot y^\smile$
by (*metis compl-inf double-compl conv-add conv-compl*)

The next lemmas show that conversion is self-conjugate in the sense of Boolean algebra with operators.

lemma *conv-self-conjugate*: $x^\smile \cdot y = 0 \longleftrightarrow x \cdot y^\smile = 0$
by (*metis conv-invol conv-times conv-zero*)

lemma *conv-self-conjugate-var*: *is-conjugation converse converse*
by (*metis conv-self-conjugate is-conjugation-def*)

The following lemmas link the relative product and meet.

lemma *one-idem-mult* [*simp*]: $1 \mathbin{;} 1 = 1$
by (*metis compl-eq-compl-iff galois-aux2 inf.commute inf-top-right mult-1-left mult-isor top-greatest*)

lemma *mult-subdistl*: $x \mathbin{;} (y \cdot z) \leq x \mathbin{;} y$
by (*metis inf-le1 mult-isol*)

lemma *mult-subdistr*: $(x \cdot y) \mathbin{;} z \leq x \mathbin{;} z$
by (*metis inf-le1 mult-isor*)

lemma *mult-subdistr-var*: $(x \cdot y) \mathbin{;} z \leq x \mathbin{;} z \cdot y \mathbin{;} z$
by (*metis inf.commute le-inf-iff mult-subdistr*)

The following lemmas deal with variants of the Peirce law, the Schröder laws and the Dedekind law. Some of them are obtained from Boolean algebras with operators by instantiation, using conjugation properties. However, Isabelle does not always pick up this relationship.

lemma *peirce-1*: $x \mathbin{;} y \cdot z^\smile = 0 \Longrightarrow y \mathbin{;} z \cdot x^\smile = 0$
by (*metis compl-le-swap1 conv-contrav conv-self-conjugate galois-aux comp-res conv-invol galois-aux mult-isol order-trans*)

lemma *peirce*: $x \mathbin{;} y \cdot z^\smile = 0 \longleftrightarrow y \mathbin{;} z \cdot x^\smile = 0$
by (*metis peirce-1*)

lemma *schroeder-1*: $x \mathbin{;} y \cdot z = 0 \longleftrightarrow y \cdot x^\smile \mathbin{;} z = 0$
by (*metis conv-invol peirce conv-contrav conv-invol conv-self-conjugate inf.commute*)

lemma *schroeder-2*: $y \mathbin{;} x \cdot z = 0 \longleftrightarrow y \cdot z \mathbin{;} x^\smile = 0$
by (*metis conv-invol peirce schroeder-1*)

The following two conjugation properties between multiplication with elements and their converses are used for deriving modular laws of relation algebra from those of Boolean algebras with operators.

lemma *schroeder-1-var*: *is-conjugation* (*composition x*) (*composition* ($x^\smile$))
by (*metis schroeder-1 is-conjugation-def*)

lemma *schroeder-2-var*: *is-conjugation* ($\lambda x.\ x \mathbin{;} y$) ($\lambda x.\ x \mathbin{;} y^\smile$)
by (*unfold is-conjugation-def*, *metis schroeder-2*)

The following Galois connections define residuals. They link relation algebras with action algebras. This could be further explored and formalised.

lemma *conv-galois-1*: $x\ ;\ y \leq z \longleftrightarrow y \leq -(x^\smile\ ;\ -z)$
by (*metis galois-aux galois-aux2 schroeder-1*)

lemma *conv-galois-2*: $y\ ;\ x \leq z \longleftrightarrow y \leq -(-z\ ;\ x^\smile)$
by (*metis galois-aux galois-aux2 schroeder-2*)

Variants of the modular law for relation algebras can now be instantiated from Boolean algebras with operators.

lemma *modular-1-aux'*: $x\ ;\ (y \cdot -(x^\smile\ ;\ z)) \cdot z = 0$
by (*metis schroeder-1-var modular-1-aux*)

lemma *modular-2-aux'*: $(y \cdot -(z\ ;\ x^\smile))\ ;\ x \cdot z = 0$
by (*metis modular-1-aux schroeder-2-var*)

lemma *modular-1'*: $x\ ;\ y \cdot z = x\ ;\ (y \cdot x^\smile\ ;\ z) \cdot z$
by (*metis schroeder-1-var modular-1*)

lemma *modular-2'*: $y\ ;\ x \cdot z = (y \cdot z\ ;\ x^\smile)\ ;\ x \cdot z$
proof −
  have $y\ ;\ x \cdot z = (y \cdot z\ ;\ x^\smile)\ ;\ x \cdot z + (y \cdot -(z\ ;\ x^\smile))\ ;\ x \cdot z$
    by (*metis aux4 distrib-right inf.commute inf-sup-distrib1*)
  thus *?thesis*
    by (*metis sup-bot-right modular-2-aux'*)
qed

lemma *modular-1-var*: $x\ ;\ y \cdot z \leq x\ ;\ (y \cdot x^\smile\ ;\ z)$
by (*metis inf.commute inf-le2 modular-1'*)

lemma *modular-2-var*: $x\ ;\ y \cdot z \leq (x \cdot z\ ;\ y^\smile)\ ;\ y$
by (*metis inf.commute inf-le2 modular-2'*)

lemma *modular-var-2*: $x\ ;\ y \leq x\ ;\ (y \cdot x^\smile\ ;\ 1)$
by (*metis inf-top-right modular-1-var*)

lemma *modular-var-3*: $x\ ;\ y \leq (x \cdot 1\ ;\ y^\smile)\ ;\ y$
by (*metis inf-top-right modular-2-var*)

The modular laws are used to prove the Dedekind rule.

lemma *dedekind*: $x\ ;\ y \cdot z \leq (x \cdot z\ ;\ y^\smile)\ ;\ (y \cdot x^\smile\ ;\ z)$
proof −
  have $x\ ;\ y \cdot z \leq (x \cdot z\ ;\ y^\smile)\ ;\ (y \cdot ((x \cdot z\ ;\ y^\smile)^\smile\ ;\ z))$
    by (*metis modular-2' modular-1-var*)
  also have $\ldots \leq (x \cdot z\ ;\ y^\smile)\ ;\ (y \cdot x^\smile\ ;\ z)$
    by (*metis conv-iso inf-le1 inf-mono mult-isol mult-isor order-refl*)
  thus *?thesis*
    by (*metis calculation order-trans*)
qed

lemma *dedekind-var-1*: $x\ ;\ y \leq (x \cdot 1\ ;\ y^\smile)\ ;\ (y \cdot x^\smile\ ;\ 1)$

by (*metis dedekind inf.commute inf-top-left*)

end

The Schröder laws allow us, finally, to prove the annihilation laws for zero. We formalise this by proving that relation algebras form dioids with zero.

sublocale *relation-algebra* < *dioid-one-zero* (+) (;) *1′ 0* (≤) (<)
proof
fix $x :: 'a$
show $0 + x = x$
by (*fact sup-bot-left*)
show $0 \ ; x = 0$
by (*metis f-strict schroeder-2-var*)
show $x \ ; 0 = 0$
by (*metis f-strict schroeder-1-var*)
qed

context *relation-algebra*
begin

Next we prove miscellaneous properties which we found in the books of Maddux and Schmidt and Ströhlein. Most of them do not carry any meaningful names.

lemma *ra-1*: $(x \cdot y \ ; 1) \ ; z = x \ ; z \cdot y \ ; 1$
proof (*rule order.antisym*)
show $x \ ; z \cdot y \ ; 1 \leq (x \cdot y \ ; 1) \ ; z$
by (*metis modular-2-var comp-assoc order-trans eq-refl inf-mono inf-top-left mult-isor mult-subdistl*)
show $(x \cdot y \ ; 1) \ ; z \leq x \ ; z \cdot y \ ; 1$
by (*metis inf.commute inf-greatest inf-top-left mult.assoc mult-subdistl mult-subdistr order-trans*)
qed

lemma *ra-2*: $x \ ; (z \cdot y \ ; 1) = (x \cdot (y \ ; 1)^\smile) \ ; z$
proof (*rule order.antisym*)
have $(x \cdot 1 \ ; (z \cdot y \ ; 1)^\smile) \ ; z \leq (x \cdot (y \ ; 1)^\smile) \ ; z$
by (*metis conv-contrav conv-invol conv-one conv-times inf.idem inf.left-commute le-iff-inf meet-assoc mult-isor ra-1*)
thus $x \ ; (z \cdot y \ ; 1) \leq (x \cdot (y \ ; 1)^\smile) \ ; z$
by (*metis modular-var-3 mult-subdistl order-trans*)
next
have $x \ ; (z \cdot ((x \cdot (y \ ; 1)^\smile)^\smile \ ; 1)) \leq x \ ; (z \cdot y \ ; 1)$
by (*metis conv-invol conv-times eq-refl inf-le2 inf-mono mult.assoc mult-isol mult-isor one-idem-mult*)
thus $x \ ; (z \cdot y \ ; 1) \geq (x \cdot (y \ ; 1)^\smile) \ ; z$
by (*metis modular-var-2 mult-subdistr order-trans*)
qed

lemma *one-conv*: $1' \cdot x \ ; 1 = 1' \cdot x \ ; x^\smile$

by (*metis inf.commute inf-top-left modular-1′ mult.right-neutral*)

lemma *maddux-12*: $-(y \mathbin{;} x) \mathbin{;} x^\smile \leq -y$
by (*metis galois-aux inf.commute inf-compl-bot schroeder-2*)

lemma *maddux-141*: $x \mathbin{;} y \cdot z = 0 \longleftrightarrow x^\smile \mathbin{;} z \cdot y = 0$
by (*metis inf.commute schroeder-1*)

lemma *maddux-142*: $x^\smile \mathbin{;} z \cdot y = 0 \longleftrightarrow z \mathbin{;} y^\smile \cdot x = 0$
by (*metis inf.commute schroeder-1 schroeder-2*)

lemmas *maddux-16* = *modular-1-var*

lemmas *maddux-17* = *modular-2-var*

lemma *maddux-20*: $x \leq x \mathbin{;} 1$
by (*metis inf-top-left mult.right-neutral mult-subdistl*)

lemma *maddux-21*: $x \leq 1 \mathbin{;} x$
by (*metis mult-isor mult-onel top-greatest*)

lemma *maddux-23*: $x \mathbin{;} y \cdot -(x \mathbin{;} z) = x \mathbin{;} (y \cdot -z) \cdot -(x \mathbin{;} z)$
apply (*rule order.antisym*)
apply (*metis local.aux6 local.aux6-var local.aux9 local.compl-inf-bot local.compl-sup-top local.compl-unique local.distrib-left local.galois-2 local.sup-ge2*)
using *local.meet-iso local.mult-subdistl* **by** *blast*

lemma *maddux-24*: $-(x \mathbin{;} y) + x \mathbin{;} z = -(x \mathbin{;} (y \cdot -z)) + x \mathbin{;} z$
by (*metis de-morgan-3 double-compl maddux-23*)

lemma *one-compl*: $-(x \mathbin{;} 1) \mathbin{;} 1 = -(x \mathbin{;} 1)$
by (*metis order.antisym conv-one maddux-12 mult.assoc one-idem-mult maddux-20*)

lemma *ss-p18*: $x \mathbin{;} 1 = 0 \longleftrightarrow x = 0$
by (*metis annil le-bot maddux-20*)

end

This finishes our development of the basic laws of relation algebras. The next sections are devoted to special elements such as vectors, test or subidentities, and, in particular, functions.

end

4 Vectors

theory *Relation-Algebra-Vectors*
  imports *Relation-Algebra*
begin

Vectors can be used for modelling sets of states. In this section we follow Maddux's book to derive some of their most important properties.

context *relation-algebra*
begin

definition *is-vector* :: $'a \Rightarrow bool$
  where $is\text{-}vector\ x \equiv x = x\ ;\ 1$

lemma *vector-compl*: $is\text{-}vector\ x \Longrightarrow is\text{-}vector\ (-x)$
by (*metis one-compl is-vector-def*)

lemma *vector-add*: $[\![is\text{-}vector\ x;\ is\text{-}vector\ y]\!] \Longrightarrow is\text{-}vector\ (x + y)$
by (*metis comp-distr is-vector-def*)

lemma *vector-mult*: $[\![is\text{-}vector\ x;\ is\text{-}vector\ y]\!] \Longrightarrow is\text{-}vector\ (x \cdot y)$
by (*metis ra-1 is-vector-def*)

lemma *vector-comp*: $[\![is\text{-}vector\ x;\ is\text{-}vector\ y]\!] \Longrightarrow is\text{-}vector\ (x\ ;\ y)$
by (*metis comp-assoc is-vector-def*)

lemma *vector-1*: $is\text{-}vector\ x \Longrightarrow (x \cdot y)\ ;\ z = x \cdot y\ ;\ z$
by (*metis inf.commute ra-1 is-vector-def*)

lemma *vector-1-comm*: $is\text{-}vector\ y \Longrightarrow (x \cdot y)\ ;\ z = x\ ;\ z \cdot y$
by (*metis ra-1 is-vector-def*)

lemma *vector-2*: $is\text{-}vector\ y \Longrightarrow (x \cdot y^\smile)\ ;\ z = x\ ;\ (y \cdot z)$
by (*metis inf.commute ra-2 is-vector-def*)

lemma *vector-2-var*: $is\text{-}vector\ y \Longrightarrow (x \cdot y^\smile)\ ;\ z = (x \cdot y^\smile)\ ;\ (y \cdot z)$
by (*metis inf.left-idem vector-2*)

lemma *vector-idem* [*simp*]: $is\text{-}vector\ x \Longrightarrow x\ ;\ x = x$
by (*metis inf-absorb2 inf-top-left maddux-21 vector-1-comm*)

lemma *vector-rectangle* [*simp*]: $is\text{-}vector\ x \Longrightarrow x\ ;\ 1\ ;\ x = x$
by (*metis vector-idem is-vector-def*)

lemma *vector-3* [*simp*]: $is\text{-}vector\ x \Longrightarrow (x \cdot 1')\ ;\ y = x \cdot y$
by (*metis inf.commute mult.left-neutral vector-1*)

end

end

5 Tests

theory *Relation-Algebra-Tests*

imports *Relation-Algebra*
begin

5.1 Tests

Tests or subidentities provide another way of modelling sets. Once more we prove the basic properties, most of which stem from Maddux's book.

context *relation-algebra*
begin

definition *is-test* :: $'a \Rightarrow bool$
  where *is-test* $x \equiv x \leq 1'$

lemma *test-conv*: *is-test* $x \Longrightarrow$ *is-test* $(x^\smile)$
by (*metis conv-e conv-iso is-test-def*)

lemma *test-conv-var*: *is-test* $x \Longrightarrow x^\smile \leq 1'$
by (*metis test-conv is-test-def*)

lemma *test-eq-conv* [*simp*]: *is-test* $x \Longrightarrow x^\smile = x$
proof (*rule order.antisym*)
  assume *hyp*: *is-test* x
  hence $x \leq x \ ; (1' \cdot x^\smile \ ; 1')$
   by (*metis inf.commute inf-absorb2 inf-le2 modular-1' mult.right-neutral is-test-def*)
  thus $x \leq x^\smile$
   by (*metis comp-unitr conv-contrav conv-invol order.eq-iff hyp inf-absorb2 mult-subdistl test-conv-var*)
  thus $x^\smile \leq x$
    by (*metis conv-invol conv-times le-iff-inf*)
qed

lemma *test-sum*: $[\![$*is-test* x; *is-test* $y]\!] \Longrightarrow$ *is-test* $(x + y)$
by (*simp add*: *is-test-def*)

lemma *test-prod*: $[\![$*is-test* x; *is-test* $y]\!] \Longrightarrow$ *is-test* $(x \cdot y)$
by (*metis le-infI2 is-test-def*)

lemma *test-comp*: $[\![$*is-test* x; *is-test* $y]\!] \Longrightarrow$ *is-test* $(x \ ; y)$
by (*metis mult-isol comp-unitr order-trans is-test-def*)

lemma *test-comp-eq-mult*:
  assumes *is-test* x
    and *is-test* y
  shows $x \ ; y = x \cdot y$
proof (*rule order.antisym*)
  show $x \ ; y \leq x \cdot y$
   by (*metis assms comp-unitr inf-absorb2 le-inf-iff mult-onel mult-subdistl mult-subdistr is-test-def*)
next

have $x \cdot y \leq x \ ; \ (1' \cdot x^{\smile} \ ; \ y)$
by (*metis comp-unitr modular-1-var*)
thus $x \cdot y \leq x \ ; \ y$
by (*metis assms*(*1*) *inf-absorb2 le-infI2 mult.left-neutral mult-isol mult-subdistr order-trans test-eq-conv is-test-def*)
qed

lemma *test-1* [*simp*]: *is-test* $x \Longrightarrow x \ ; \ 1 \cdot y = x \ ; \ y$
by (*metis inf.commute inf.idem inf-absorb2 mult.left-neutral one-conv ra-1 test-comp-eq-mult test-eq-conv is-test-def*)

lemma *maddux-32* [*simp*]: *is-test* $x \Longrightarrow -(x \ ; \ 1) \cdot 1' = -x \cdot 1'$
proof (*rule order.antisym*)
assume *is-test x*
show $-(x \ ; \ 1) \cdot 1' \leq -x \cdot 1'$
by (*metis maddux-20 comp-anti inf.commute meet-isor*)
next
assume *is-test x*
have *one*: $x \ ; \ 1 \cdot (-x \cdot 1') \leq x \ ; \ x^{\smile} \ ; \ (-x \cdot 1')$
by (*metis maddux-16 inf-top-left mult.assoc*)
hence *two*: $x \ ; \ 1 \cdot (-x \cdot 1') \leq -x$
by (*metis inf.commute inf-le1 le-infE*)
hence $x \ ; \ 1 \cdot (-x \cdot 1') \leq x$
by (*metis one inf.commute le-infE meet-iso one-conv* ‹*is-test x*› *order.eq-iff test-1 test-eq-conv*)
hence $x \ ; \ 1 \cdot (-x \cdot 1') = 0$
by (*metis two galois-aux2 le-iff-inf*)
thus $-x \cdot 1' \leq -(x \ ; \ 1) \cdot 1'$
by (*metis double-compl galois-aux2 inf.commute inf-le1 le-inf-iff*)
qed

lemma *test-distr-1* :
assumes *is-test x*
and *is-test y*
shows $x \ ; \ z \cdot y \ ; \ z = (x \cdot y) \ ; \ z$
proof (*rule order.antisym*)
have $x \ ; \ z \cdot y \ ; \ z \leq x \ ; \ 1 \cdot y \ ; \ z$
by (*metis inf-top-left meet-iso mult-subdistl*)
also have $\ldots = x \ ; \ y \ ; \ z$
by (*metis assms*(*1*) *mult.assoc test-1*)
finally show $x \ ; \ z \cdot y \ ; \ z \leq (x \cdot y) \ ; \ z$
by (*metis assms test-comp-eq-mult*)
next
show $(x \cdot y) \ ; \ z \leq x \ ; \ z \cdot y \ ; \ z$
by (*metis mult-subdistr-var*)
qed

lemma *maddux-35*: *is-test* $x \Longrightarrow x \ ; \ y \cdot -z = x \ ; \ y \cdot -(x \ ; \ z)$
proof (*rule order.antisym*)

assume *is-test x*
show $x \ ; \ y \cdot -z \leq x \ ; \ y \cdot -(x \ ; \ z)$
by (*metis ‹is-test x› comp-anti mult-isor mult-onel is-test-def inf.commute inf-le2 le-infI le-infI1*)
have *one*: $x \ ; \ y \cdot -(x \ ; \ z) \leq x \ ; \ (y \cdot -z)$
by (*metis order.eq-iff le-infE maddux-23*)
hence *two*: $x \ ; \ y \cdot -(x \ ; \ z) \leq x \ ; \ y$
by (*metis inf-le1*)
have $x \ ; \ y \cdot -(x \ ; \ z) \leq \ x \ ; \ -z$
using *one*
by (*metis galois-1 le-iff-sup distrib-left sup-compl-top sup-top-right*)
hence $x \ ; \ y \cdot -(x \ ; \ z) \leq -z$
by (*metis ‹is-test x› mult-isor mult-onel is-test-def order-trans*)
thus $x \ ; \ y \cdot -(x \ ; \ z) \leq x \ ; \ y \cdot -z$
using *two*
by (*metis le-inf-iff*)
qed

5.2 Test Complements

Text complements are complements of elements that are "pushed below" the multiplicative unit.

definition $tc :: 'a \Rightarrow 'a$
where $tc \ x = 1' \cdot -x$

lemma *test-compl-1* [*simp*]: *is-test* $x \Longrightarrow x + tc \ x = 1'$
by (*metis is-test-def local.aux4 local.inf.absorb-iff1 local.inf-commute tc-def*)

lemma *test-compl-2* [*simp*]: *is-test* $x \Longrightarrow x \cdot tc \ x = 0$
by (*metis galois-aux inf.commute inf-le2 tc-def*)

lemma *test-test-compl*: *is-test* $x \Longrightarrow$ *is-test* $(tc \ x)$
by (*simp add*: *is-test-def tc-def*)

lemma *test-compl-de-morgan-1*: $tc \ (x + y) = tc \ x \cdot tc \ y$
by (*metis compl-sup inf.left-commute inf.left-idem meet-assoc tc-def*)

lemma *test-compl-de-morgan-2*: $tc \ (x \cdot y) = tc \ x + tc \ y$
by (*metis compl-inf inf-sup-distrib1 tc-def*)

lemma *test-compl-three* [*simp*]: $tc \ (tc \ (tc \ x)) = tc \ x$
by (*metis aux4 aux6 de-morgan-3 inf.commute inf-sup-absorb tc-def*)

lemma *test-compl-double* [*simp*]: *is-test* $x \Longrightarrow tc \ (tc \ x) = x$
by (*metis aux6-var compl-inf double-compl inf.commute le-iff-inf tc-def is-test-def*)

end

end

6 Functions

theory *Relation-Algebra-Functions*
  imports *Relation-Algebra-Vectors Relation-Algebra-Tests*
begin

6.1 Functions

This section collects the most important properties of functions. Most of them can be found in the books by Maddux and by Schmidt and Ströhlein. The main material is on partial and total functions, injections, surjections, bijections.

context *relation-algebra*
begin

definition *is-p-fun* :: *'a* ⇒ *bool*
  where *is-p-fun x* ≡ $x^\smile$; *x* ≤ *1'*

definition *is-total* :: *'a* ⇒ *bool*
  where *is-total x* ≡ *1'* ≤ *x* ; $x^\smile$

definition *is-map* :: *'a* ⇒ *bool*
  where *is-map x* ≡ *is-p-fun x* ∧ *is-total x*

definition *is-inj* :: *'a* ⇒ *bool*
  where *is-inj x* ≡ *x* ; $x^\smile$ ≤ *1'*

definition *is-sur* :: *'a* ⇒ *bool*
  where *is-sur x* ≡ *1'* ≤ $x^\smile$; *x*

We distinguish between partial and total bijections. As usual we call the latter just bijections.

definition *is-p-bij* :: *'a* ⇒ *bool*
  where *is-p-bij x* ≡ *is-p-fun x* ∧ *is-inj x* ∧ *is-sur x*

definition *is-bij* :: *'a* ⇒ *bool*
  where *is-bij x* ≡ *is-map x* ∧ *is-inj x* ∧ *is-sur x*

Our first set of lemmas relates the various concepts.

lemma *inj-p-fun*: *is-inj x* ⟷ *is-p-fun* ($x^\smile$)
by (*metis conv-invol is-inj-def is-p-fun-def*)

lemma *p-fun-inj*: *is-p-fun x* ⟷ *is-inj* ($x^\smile$)
by (*metis conv-invol inj-p-fun*)

lemma *sur-total*: *is-sur x* ⟷ *is-total* ($x^\smile$)
by (*metis conv-invol is-sur-def is-total-def*)

lemma *total-sur*: *is-total* $x \longleftrightarrow$ *is-sur* $(x^\smile)$
by (*metis conv-invol sur-total*)

lemma *bij-conv*: *is-bij* $x \longleftrightarrow$ *is-bij* $(x^\smile)$
by (*metis is-bij-def inj-p-fun is-map-def p-fun-inj sur-total total-sur*)

Next we show that tests are partial injections.

lemma *test-is-inj-fun*: *is-test* $x \Longrightarrow$ (*is-p-fun* $x \wedge$ *is-inj* x)
by (*metis is-inj-def p-fun-inj test-comp test-eq-conv is-test-def*)

Next we show composition properties.

lemma *p-fun-comp*:
assumes *is-p-fun* x **and** *is-p-fun* y
shows *is-p-fun* $(x ; y)$
proof (*unfold is-p-fun-def*)
have $(x ; y)^\smile ; x ; y = y^\smile ; x^\smile ; x ; y$
by (*metis conv-contrav mult.assoc*)
also have $... \leq y^\smile ; y$
by (*metis assms(1) is-p-fun-def mult-double-iso mult.right-neutral mult.assoc*)
finally show $(x ; y)^\smile ; (x ; y) \leq 1'$
by (*metis assms(2) order-trans is-p-fun-def mult.assoc*)
qed

lemma *p-fun-mult-var*: $x^\smile ; x \leq 1' \Longrightarrow (x \cdot y)^\smile ; (x \cdot y) \leq 1'$
by (*metis conv-times inf-le1 mult-isol-var order-trans*)

lemma *inj-compose*:
assumes *is-inj* x **and** *is-inj* y
shows *is-inj* $(x ; y)$
by (*metis assms conv-contrav inj-p-fun p-fun-comp*)

lemma *inj-mult-var*: $x^\smile ; x \leq 1' \Longrightarrow (x \cdot y)^\smile ; (x \cdot y) \leq 1'$
by (*metis p-fun-mult-var*)

lemma *total-comp*:
assumes *is-total* x **and** *is-total* y
shows *is-total* $(x ; y)$
by (*metis assms inf-top-left le-iff-inf mult.assoc mult.right-neutral one-conv ra-2 is-total-def*)

lemma *total-add-var*: $1' \leq x^\smile ; x \Longrightarrow 1' \leq (x + y)^\smile ; (x + y)$
by (*metis local.conv-add local.inf.absorb2 local.inf.coboundedI1 local.join-interchange local.le-sup-iff*)

lemma *sur-comp*:
assumes *is-sur* x **and** *is-sur* y
shows *is-sur* $(x ; y)$
by (*metis assms conv-contrav sur-total total-comp*)

lemma *sur-sum-var*: $1' \leq x^\smile \ ; \ x \Longrightarrow 1' \leq (x + y)^\smile \ ; \ (x + y)$
by (*metis total-add-var*)

lemma *map-comp*:
assumes *is-map x* **and** *is-map y*
shows *is-map* $(x \ ; \ y)$
by (*metis assms is-map-def p-fun-comp total-comp*)

lemma *bij-comp*:
assumes *is-bij x* **and** *is-bij y*
shows *is-bij* $(x \ ; \ y)$
by (*metis assms is-bij-def inj-compose map-comp sur-comp*)

We now show that (partial) functions, unlike relations, distribute over meets from the left.

lemma *p-fun-distl*: *is-p-fun* $x \Longrightarrow x \ ; \ (y \cdot z) = x \ ; \ y \cdot x \ ; \ z$
proof −
assume *is-p-fun x*
hence $x \ ; \ (z \cdot ((x^\smile \ ; \ x) \ ; \ y)) \leq x \ ; \ (z \cdot y)$
by (*metis is-p-fun-def inf-le1 le-infI le-infI2 mult-isol mult-isor mult-onel*)
hence $x \ ; \ y \cdot x \ ; \ z \leq x \ ; \ (z \cdot y)$
by (*metis inf.commute mult.assoc order-trans modular-1-var*)
thus $x \ ; \ (y \cdot z) = x \ ; \ y \cdot x \ ; \ z$
by (*metis order.eq-iff inf.commute le-infI mult-subdistl*)
qed

lemma *map-distl*: *is-map* $x \Longrightarrow x \ ; \ (y \cdot z) = x \ ; \ y \cdot x \ ; \ z$
by (*metis is-map-def p-fun-distl*)

Next we prove simple properties of functions which arise in equivalent definitions of those concepts.

lemma *p-fun-zero*: *is-p-fun* $x \Longrightarrow x \ ; \ y \cdot x \ ; \ -y = 0$
by (*metis annir inf-compl-bot p-fun-distl*)

lemma *total-one*: *is-total* $x \Longrightarrow x \ ; \ 1 = 1$
by (*metis conv-invol conv-one inf-top-left le-iff-inf mult.right-neutral one-conv ra-2 is-total-def*)

lemma *total-1*: *is-total* $x \Longrightarrow (\forall\, y.\ y \ ; \ x = 0 \longrightarrow y = 0)$
by (*metis conv-invol conv-zero inf-bot-left inf-top-left peirce total-one*)

lemma *surj-one*: *is-sur* $x \Longrightarrow 1 \ ; \ x = 1$
by (*metis conv-contrav conv-invol conv-one sur-total total-one*)

lemma *surj-1*: *is-sur* $x \Longrightarrow (\forall\, y.\ x \ ; \ y = 0 \longrightarrow y = 0)$
by (*metis comp-res-aux compl-bot-eq conv-contrav conv-one inf.commute inf-top-right surj-one*)

lemma *bij-is-maprop*:

assumes *is-bij x* **and** *is-map x*
shows $x^\smile \mathbin{;} x = 1' \wedge x \mathbin{;} x^\smile = 1'$
by (*metis assms is-bij-def order.eq-iff is-inj-def is-map-def is-p-fun-def is-sur-def is-total-def*)

We now provide alternative definitions for functions. These can be found in Schmidt and Ströhlein's book.

lemma *p-fun-def-var*: $\textit{is-p-fun}\ x \longleftrightarrow x \mathbin{;} -(1') \leq -x$
by (*metis conv-galois-1 double-compl galois-aux inf.commute is-p-fun-def*)

lemma *total-def-var-1*: $\textit{is-total}\ x \longleftrightarrow x \mathbin{;} 1 = 1$
by (*metis inf-top-right le-iff-inf one-conv total-one is-total-def*)

lemma *total-def-var-2*: $\textit{is-total}\ x \longleftrightarrow -x \leq x \mathbin{;} -(1')$
by (*metis total-def-var-1 distrib-left sup-compl-top mult.right-neutral galois-aux3*)

lemma *sur-def-var1*: $\textit{is-sur}\ x \longleftrightarrow 1 \mathbin{;} x = 1$
by (*metis conv-contrav conv-one sur-total surj-one total-def-var-1*)

lemma *sur-def-var2*: $\textit{is-sur}\ x \longleftrightarrow -x \leq -(1') \mathbin{;} x$
by (*metis sur-total total-def-var-2 conv-compl conv-contrav conv-e conv-iso*)

lemma *inj-def-var1*: $\textit{is-inj}\ x \longleftrightarrow -(1') \mathbin{;} x \leq -x$
by (*metis conv-galois-2 double-compl galois-aux inf.commute is-inj-def*)

lemma *is-maprop*: $\textit{is-map}\ x \longleftrightarrow x \mathbin{;} -(1') = -x$
by (*metis order.eq-iff is-map-def p-fun-def-var total-def-var-2*)

Finally we prove miscellaneous properties of functions.

lemma *ss-422iii*: $\textit{is-p-fun}\ y \Longrightarrow (x \cdot z \mathbin{;} y^\smile) \mathbin{;} y = x \mathbin{;} y \cdot z$

proof (*rule order.antisym*)
assume *is-p-fun y*
show $x \mathbin{;} y \cdot z \leq (x \cdot z \mathbin{;} y^\smile) \mathbin{;} y$
by (*metis maddux-17*)
have $(x \cdot z \mathbin{;} y^\smile) \mathbin{;} y \leq x \mathbin{;} y \cdot (z \mathbin{;} (y^\smile \mathbin{;} y))$
by (*metis mult-subdistr-var mult.assoc*)
also have $\ldots \leq x \mathbin{;} y \cdot z \mathbin{;} 1'$
by (*metis ‹is-p-fun y› inf-absorb2 inf-le1 le-infI le-infI2 mult-subdistl is-p-fun-def*)
finally show $(x \cdot z \mathbin{;} y^\smile) \mathbin{;} y \leq x \mathbin{;} y \cdot z$
by (*metis mult.right-neutral*)
qed

lemma *p-fun-compl*: $\textit{is-p-fun}\ x \Longrightarrow x \mathbin{;} -y \leq -(x;\ y)$
by (*metis annir galois-aux inf.commute inf-compl-bot p-fun-distl*)

lemma *ss-422v*: $\textit{is-p-fun}\ x \Longrightarrow x \mathbin{;} -y = x \mathbin{;} 1 \cdot -(x \mathbin{;} y)$
by (*metis inf.commute inf-absorb2 inf-top-left maddux-23 p-fun-compl*)

The next property is a Galois connection.

lemma *ss43iii*: *is-map* $x \longleftrightarrow (\forall\, y.\ x\ ;\ -y = -(x\ ;\ y))$
by *standard* (*metis inf-top-left is-map-def ss-422v total-one*, *metis is-maprop mult.right-neutral*)

Next we prove a lemma from Schmidt and Ströhlein's book and some of its consequences. We show the proof in detail since the textbook proof uses Tarski's rule which we omit.

lemma *ss423*: *is-map* $x \Longrightarrow y\ ;\ x \leq z \longleftrightarrow y \leq z\ ;\ x^\smile$
proof
  assume *is-map* x **and** $y\ ;\ x \leq z$
  hence $y \leq y\ ;\ x\ ;\ x^\smile$
    by (*metis is-map-def mult-1-right mult.assoc mult-isol is-total-def*)
  thus $y \leq z\ ;\ x^\smile$
    by (*metis* ‹$y\ ;\ x \leq z$› *mult-isor order-trans*)
next
  assume *is-map* x **and** $y \leq z\ ;\ x^\smile$
  hence $y\ ;\ x \leq z\ ;\ x^\smile\ ;\ x$
    by (*metis mult-isor*)
  also have $\ldots \leq z\ ;\ 1'$
    by (*metis* ‹*is-map* x› *is-map-def mult.assoc mult-isol is-p-fun-def*)
  finally show $y\ ;\ x \leq z$
    by (*metis mult-1-right*)
qed

lemma *ss424i*: *is-total* $x \longleftrightarrow (\forall\, y.\ -(x\ ;\ y) \leq x\ ;\ -y)$
by (*metis galois-aux3 distrib-left sup-compl-top total-def-var-1*)

lemma *ss434ii*: *is-p-fun* $x \longleftrightarrow (\forall\, y.\ x\ ;\ -y \leq -(x\ ;\ y))$
by (*metis mult.right-neutral p-fun-compl p-fun-def-var*)

lemma *is-maprop1*: *is-map* $x \Longrightarrow (y \leq x\ ;\ z\ ;\ x^\smile \longleftrightarrow y\ ;\ x \leq x\ ;\ z)$
by (*metis ss423*)

lemma *is-maprop2*: *is-map* $x \Longrightarrow (y\ ;\ x \leq x\ ;\ z \longleftrightarrow x^\smile\ ;\ y;\ x \leq z)$
by *standard* (*metis galois-aux2 inf-commute mult.assoc schroeder-1 ss43iii*, *metis conv-contrav conv-invol conv-iso mult.assoc ss423*)

lemma *is-maprop3*: *is-map* $x \Longrightarrow (x^\smile\ ;\ y;\ x \leq z \longleftrightarrow x^\smile\ ;\ y \leq z\ ;\ x^\smile)$
by (*metis ss423*)

lemma *p-fun-sur-id* [*simp*]:
  assumes *is-p-fun* x **and** *is-sur* x
  shows $x^\smile\ ;\ x = 1'$
by (*metis assms order.eq-iff is-p-fun-def is-sur-def*)

lemma *total-inj-id* [*simp*]:
  assumes *is-total* x **and** *is-inj* x
  shows $x\ ;\ x^\smile = 1'$
by (*metis assms conv-invol inj-p-fun p-fun-sur-id sur-total*)

lemma *bij-inv-1* [*simp*]: *is-bij x* $\Longrightarrow$ *x* ; $x^\smile$ = *1′*
by (*metis bij-is-maprop is-bij-def*)

lemma *bij-inv-2* [*simp*]: *is-bij x* $\Longrightarrow$ $x^\smile$; *x* = *1′*
by (*metis bij-is-maprop is-bij-def*)

lemma *bij-inv-comm*: *is-bij x* $\Longrightarrow$ *x* ; $x^\smile$ = $x^\smile$; *x*
by (*metis bij-inv-1 bij-inv-2*)

lemma *is-bijrop*: *is-bij x* $\Longrightarrow$ (*y* = *x* ; *z* $\longleftrightarrow$ *z* = $x^\smile$; *y*)
by (*metis bij-inv-1 bij-inv-2 mult.assoc mult.left-neutral*)

lemma *inj-map-monomorph*: ⟦*is-inj x*; *is-map x*⟧ $\Longrightarrow$ (∀ *y z*. *y* ; *x* = *z* ; *x* $\longrightarrow$ *y* = *z*)
by (*metis is-map-def mult.assoc mult.right-neutral total-inj-id*)

lemma *sur-map-epimorph*: ⟦*is-sur x*; *is-map x*⟧ $\Longrightarrow$ (∀ *y z*. *x* ; *y* = *x* ; *z* $\longrightarrow$ *y* = *z*)
by (*metis order.eq-iff mult.assoc mult.left-neutral ss423 is-sur-def*)

6.2 Points and Rectangles

Finally here is a section on points and rectangles. This is only a beginning.

definition *is-point* :: *′a* ⇒ *bool*
where *is-point x* ≡ *is-vector x* ∧ *is-inj x* ∧ *x* ≠ *0*

definition *is-rectangle* :: *′a* ⇒ *bool*
where *is-rectangle x* ≡ *x* ; *1* ; *x* ≤ *x*

lemma *rectangle-eq* [*simp*]: *is-rectangle x* $\longleftrightarrow$ *x* ; *1* ; *x* = *x*
by (*metis conv-one dedekind order.eq-iff inf-top-left mult.assoc one-idem-mult is-rectangle-def*)

6.3 Antidomain

This section needs to be linked with domain semirings. We essentially prove the antidomain semiring axioms. Then we have the abstract properties at our disposition.

definition *antidom* :: *′a* ⇒ *′a* (‹*a*›)
where *a x* = *1′* · (−(*x* ; *1*))

definition *dom* :: *′a* ⇒ *′a* (‹*d*›)
where *d x* = *a* (*a x*)

lemma *antidom-test-comp* [*simp*]: *a x* = *tc* (*x* ; *1*)
by (*metis antidom-def tc-def*)

lemma *dom-def-aux*: *d x* = *1′* · *x* ; *1*

by (*metis antidom-test-comp dom-def double-compl inf-top-left mult.left-neutral one-compl ra-1 tc-def*)

lemma *dom-def-aux-var*: $d\ x = 1' \cdot x\ ;\ x^\smile$
by (*metis dom-def-aux one-conv*)

lemma *antidom-dom* [*simp*]: $a\ (d\ x) = a\ x$
by (*metis antidom-test-comp dom-def-aux inf-top-left mult.left-neutral ra-1*)

lemma *dom-antidom* [*simp*]: $d\ (a\ x) = a\ x$
by (*metis antidom-dom dom-def*)

lemma *dom-verystrict*: $d\ x = 0 \longleftrightarrow x = 0$
using *dom-def-aux-var local.schroeder-2* **by** *force*

lemma *a-1* [*simp*]: $a\ x\ ;\ x = 0$
by (*metis antidom-test-comp galois-aux2 maddux-20 mult.left-neutral one-compl ra-1 tc-def*)

lemma *a-2*: $a\ (x\ ;\ y) = a\ (x\ ;\ d\ y)$
by (*metis antidom-test-comp dom-def-aux inf-top-left mult.assoc mult.left-neutral ra-1*)

lemma *a-3* [*simp*]: $a\ x + d\ x = 1'$
by (*metis antidom-def aux4 dom-def-aux double-compl*)

lemma *test-domain*: $x = d\ x \longleftrightarrow x \leq 1'$
apply *standard*
apply (*metis dom-def-aux inf-le1*)
apply (*metis dom-def-aux inf.commute mult.right-neutral test-1 is-test-def*)
done

At this point we have all the necessary ingredients to prove that relation algebras form Boolean domain semirings. However, we omit a formal proof since we haven't formalized the latter.

lemma *dom-one*: $x\ ;\ 1 = d\ x\ ;\ 1$
by (*metis dom-def-aux inf-top-left mult.left-neutral ra-1*)

lemma *test-dom*: *is-test* $(d\ x)$
by (*metis dom-def-aux inf-le1 is-test-def*)

lemma *p-fun-dom*: *is-p-fun* $(d\ x)$
by (*metis test-dom test-is-inj-fun*)

lemma *inj-dom*: *is-inj* $(d\ x)$
by (*metis test-dom test-is-inj-fun*)

lemma *total-alt-def*: *is-total* $x \longleftrightarrow (d\ x) = 1'$
by (*metis dom-def-aux-var le-iff-inf is-total-def*)

end

end

7 Direct Products

theory *Relation-Algebra-Direct-Products*
  imports *Relation-Algebra-Functions*
begin

This section uses the definition of direct products from Schmidt and Ströhlein's book to prove the well known universal property.

context *relation-algebra*
begin

definition *is-direct-product* :: $'a \Rightarrow 'a \Rightarrow bool$
  where *is-direct-product x y* $\equiv x^\smile ; x = 1' \wedge y^\smile ; y = 1' \wedge x ; x^\smile \cdot y ; y^\smile = 1' \wedge x^\smile ; y = 1$

We collect some basic properties.

lemma *dp-p-fun1*: *is-direct-product x y* $\Longrightarrow$ *is-p-fun x*
by (*metis is-direct-product-def eq-refl is-p-fun-def*)

lemma *dp-sur1*: *is-direct-product x y* $\Longrightarrow$ *is-sur x*
by (*metis is-direct-product-def eq-refl is-sur-def*)

lemma *dp-total1*: *is-direct-product x y* $\Longrightarrow$ *is-total x*
by (*metis is-direct-product-def inf-le1 is-total-def*)

lemma *dp-map1*: *is-direct-product x y* $\Longrightarrow$ *is-map x*
by (*metis dp-p-fun1 dp-total1 is-map-def*)

lemma *dp-p-fun2*: *is-direct-product x y* $\Longrightarrow$ *is-p-fun y*
by (*metis is-direct-product-def eq-refl is-p-fun-def*)

lemma *dp-sur2*: *is-direct-product x y* $\Longrightarrow$ *is-sur y*
by (*metis is-direct-product-def eq-refl is-sur-def*)

lemma *dp-total2*: *is-direct-product x y* $\Longrightarrow$ *is-total y*
by (*metis is-direct-product-def inf-le2 is-total-def*)

lemma *dp-map2*: *is-direct-product x y* $\Longrightarrow$ *is-map y*
by (*metis dp-p-fun2 dp-total2 is-map-def*)

Next we prove four auxiliary lemmas.

lemma *dp-aux1* [*simp*]:
  assumes *is-p-fun z*

and *is-total w*
and $x^\smile$; $z = 1$
shows $(w$; $x^\smile \cdot y$; $z^\smile)$; $z = y$
by (*metis assms inf-top-left mult.assoc ss-422iii total-one inf.commute inf-top-right*)

lemma *dp-aux2* [*simp*]:
assumes *is-p-fun z*
and *is-total w*
and $z^\smile$; $x = 1$
shows $(w$; $x^\smile \cdot y$; $z^\smile)$; $z = y$
by (*metis assms conv-contrav conv-invol conv-one inf-top-left mult.assoc ss-422iii total-one*)

lemma *dp-aux3* [*simp*]:
assumes *is-p-fun z*
and *is-total w*
and $x^\smile$; $z = 1$
shows $(y$; $z^\smile \cdot w$; $x^\smile)$; $z = y$
by (*metis assms dp-aux1 inf.commute*)

lemma *dp-aux4* [*simp*]:
assumes *is-p-fun z*
and *is-total w*
and $z^\smile$; $x = 1$
shows $(\ y$; $z^\smile \cdot w$; $x^\smile)$; $z = y$
by (*metis assms dp-aux2 inf.commute*)

Next we define a function which is an isomorphism on projections.

definition *Phi* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ ($\langle\Phi\rangle$)
where $\Phi \equiv (\lambda w\ x\ y\ z.\ w$; $y^\smile \cdot x$; $z^\smile)$

lemma *Phi-conv*: $(\Phi\ w\ x\ y\ z)^\smile = y$; $w^\smile \cdot z$; $x^\smile$
by (*simp add*: *Phi-def*)

We prove that Φ is an isomorphism with respect to the projections.

lemma *mono-dp-1*:
assumes *is-direct-product w x*
and *is-direct-product y z*
shows $\Phi\ w\ x\ y\ z$; $y = w$
by (*metis assms dp-aux4 is-direct-product-def dp-p-fun1 dp-total2 Phi-def*)

lemma *mono-dp-2*:
assumes *is-direct-product w x*
and *is-direct-product y z*
shows $\Phi\ w\ x\ y\ z$; $z = x$
by (*metis assms dp-aux1 is-direct-product-def dp-p-fun2 dp-total1 Phi-def*)

We now show that Φ is an injective function.

lemma *Phi-map*:

assumes *is-direct-product w x*
and *is-direct-product y z*
shows *is-map* (Φ *w x y z*)
proof −
have $\Phi\ w\ x\ y\ z\ ;\ -(1') = \Phi\ w\ x\ y\ z\ ;\ -(y\ ;\ y^\smile\ \cdot\ z\ ;\ z^\smile)$
by (*metis assms*(*2*) *is-direct-product-def*)
also have ... $= \Phi\ w\ x\ y\ z\ ;\ y\ ;\ -(y^\smile) + \Phi\ w\ x\ y\ z\ ;\ z\ ;\ -(z^\smile)$
by (*metis compl-inf assms*(*2*) *ss43iii dp-map1 dp-map2 mult.assoc distrib-left*)
also have ... $= w\ ;\ -(y^\smile) + x\ ;\ -(z^\smile)$
by (*metis assms mono-dp-1 mono-dp-2*)
also have ... $= -(w\ ;\ y^\smile) + -(x\ ;\ z^\smile)$
by (*metis assms*(*1*) *ss43iii dp-map1 dp-map2*)
also have ... $= -(\Phi\ w\ x\ y\ z)$
by (*metis compl-inf Phi-def*)
finally show *?thesis*
by (*metis is-maprop*)
qed

lemma *Phi-inj*:
assumes *is-direct-product w x*
and *is-direct-product y z*
shows *is-inj* (Φ *w x y z*)
by (*metis Phi-def Phi-conv Phi-map assms inj-p-fun is-map-def*)

Next we show that the converse of Φ is an injective function.

lemma *Phi-conv-map*:
assumes *is-direct-product w x*
and *is-direct-product y z*
shows *is-map* $((\Phi\ w\ x\ y\ z)^\smile)$
by (*metis Phi-conv Phi-def Phi-map assms*)

lemma *Phi-conv-inj*:
assumes *is-direct-product w x*
and *is-direct-product y z*
shows *is-inj* $((\Phi\ w\ x\ y\ z)^\smile)$
by (*metis Phi-inj Phi-conv Phi-def assms*)

lemma *Phi-sur*:
assumes *is-direct-product w x*
and *is-direct-product y z*
shows *is-sur* (Φ *w x y z*)
by (*metis assms Phi-conv Phi-def Phi-map is-map-def sur-total*)

lemma *Phi-conv-sur*:
assumes *is-direct-product w x*
and *is-direct-product y z*
shows *is-sur* $((\Phi\ w\ x\ y\ z)^\smile)$
by (*metis assms Phi-conv Phi-def Phi-sur*)

lemma *Phi-bij*:
assumes *is-direct-product w x*
and *is-direct-product y z*
shows *is-bij* (Φ *w x y z*)
by (*metis assms Phi-inj Phi-map Phi-sur is-bij-def*)

lemma *Phi-conv-bij*:
assumes *is-direct-product w x*
and *is-direct-product y z*
shows *is-bij* ((Φ *w x y z*)$^\smile$)
by (*metis Phi-bij Phi-def Phi-conv assms*)

Next we construct, for given functions f and g, a function F which makes the standard product diagram commute, and we verify these commutation properties.

definition $F :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$
where $F \equiv (\lambda f\ x\ g\ y.\ f\ ;\ x^\smile \cdot g\ ;\ y^\smile)$

lemma *f-proj*:
assumes *is-direct-product x y*
and *is-map g*
shows $F\ f\ x\ g\ y\ ;\ x = f$
by (*metis assms dp-aux4 F-def is-map-def is-direct-product-def dp-p-fun1*)

lemma *g-proj*:
assumes *is-direct-product x y*
and *is-map f*
shows $F\ f\ x\ g\ y\ ;\ y = g$
by (*metis assms dp-aux1 F-def is-map-def is-direct-product-def dp-p-fun2*)

Finally we show uniqueness of F, hence universality of the construction.

lemma
assumes *is-direct-product x y*
and *is-map f*
and *is-map g*
and *is-map G*
and $f = G\ ;\ x$
and $g = G\ ;\ y$
shows $G = F\ f\ x\ g\ y$
proof −
have $F\ f\ x\ g\ y = G\ ;\ (x\ ;\ x^\smile) \cdot G\ ;\ (y\ ;\ y^\smile)$
by (*metis assms*(*5*) *assms*(*6*) *F-def mult.assoc*)
also have $\ldots = G\ ;\ (x\ ;\ x^\smile \cdot y\ ;\ y^\smile)$
by (*metis assms*(*4*) *map-distl*)
thus *?thesis*
by (*metis assms*(*1*) *is-direct-product-def calculation mult.right-neutral*)
qed

end

end

8 Reflexive Transitive Closure

theory *Relation-Algebra-RTC*
imports *Relation-Algebra*
begin

We impose the Kleene algebra axioms for the star on relation algebras. This gives us a reflexive transitive closure operation.

class *relation-algebra-rtc* = *relation-algebra* + *star-op* +
assumes *rtc-unfoldl*: $1' + x ; x^\star \leq x^\star$
and *rtc-inductl*: $z + x ; y \leq y \longrightarrow x^\star ; z \leq y$
and *rtc-inductr*: $z + y ; x \leq y \longrightarrow z ; x^\star \leq y$

sublocale *relation-algebra-rtc* $\subseteq$ *kleene-algebra* $(+)$ $(;)$ $1'$ 0 $(\leq)$ $(<)$ *star*
apply (*unfold-locales*)
apply (*rule rtc-unfoldl*)
apply (*simp add*: *local.rtc-inductl*)
apply (*simp add*: *rtc-inductr*)
done

context *relation-algebra-rtc*
begin

First, we prove that the obvious interaction between the star and converse is captured by the axioms.

lemma *star-conv*: $(x^\star)^\smile = (x^\smile)^\star$
proof (*rule order.antisym*)
show $(x^\smile)^\star \leq (x^\star)^\smile$
by (*metis local.conv-add local.conv-contrav local.conv-e local.conv-iso local.star-rtc1 local.star-rtc-least*)
show $(x^\star)^\smile \leq (x^\smile)^\star$
by (*metis boffa-var conv-add conv-contrav conv-e conv-invol conv-iso star-denest star-ext star-iso star-plus-one sup.idem*)
qed

Next we provide an example to show how facts from Kleene algebra are picked up in relation algebra.

lemma *rel-church-rosser*: $(x^\smile)^\star ; x^\star \leq x^\star ; (x^\smile)^\star \Longrightarrow (x + x^\smile)^\star = x^\star ; (x^\smile)^\star$
by (*fact church-rosser*)

end

end

9 Models of Relation Algebra

theory *Relation-Algebra-Models*
  imports *Relation-Algebra Kleene-Algebra.Inf-Matrix*
begin

We formalise two models. First we show the obvious: binary relations form a relation algebra. Then we show that infinite matrices (which we formalised originally for Kleene algebras) form models of relation algebra if we restrict their element type to *bool*.

9.1 Binary Relations

Since Isabelle's libraries for binary relations are very well developed, the proof for this model is entirely trivial.

interpretation *rel-relation-algebra*: *relation-algebra* $(-)$ *uminus* $(\cap)$ $(\subseteq)$ $(\subset)$ $(\cup)$ $\{\}$ *UNIV* (O) *Relation.converse Id*
by *unfold-locales auto*

9.2 Infinite Boolean Matrices

Next we consider infinite Boolean matrices. We define the maximal Boolean matrix (all of its entries are *True*), the converse or transpose of a matrix, the intersection of two Boolean matrices and the complement of a Boolean matrix.

definition *mat-top* :: $('a, 'b, bool)$ *matrix* $(\langle\tau\rangle)$
  where $\tau\ i\ j \equiv True$

definition *mat-transpose* :: $('a, 'b, 'c)$ *matrix* $\Rightarrow$ $('b, 'a, 'c)$ *matrix* $(\langle\text{-}^\dagger\rangle\ [101]\ 100)$
  where $f^\dagger \equiv (\lambda i\ j.\ f\ j\ i)$

definition *mat-inter* :: $('a, 'b, bool)$ *matrix* $\Rightarrow$ $('a, 'b, bool)$ *matrix* $\Rightarrow$ $('a, 'b, bool)$ *matrix* (**infixl** $\langle\sqcap\rangle$ *70*)
  where $f \sqcap g \equiv (\lambda i\ j.\ f\ i\ j \cdot g\ i\ j)$

definition *mat-complement* :: $('a, 'b, bool)$ *matrix* $\Rightarrow$ $('a, 'b, bool)$ *matrix* $(\langle\text{-}^c\rangle\ [101]\ 100)$
  where $f^c = (\lambda i\ j.\ -\ f\ i\ j)$

Next we show that the Booleans form a dioid. We state this as an *instantiation* result. The Kleene algebra files contain an *interpretation* proof, which is not sufficient for our purposes.

instantiation *bool* :: *dioid-one-zero*
begin

  definition *zero-bool-def*:
    zero-bool $\equiv$ *False*

definition *one-bool-def*:
one-bool ≡ *True*

definition *times-bool-def*:
times-bool ≡ (∧)

definition *plus-bool-def*:
plus-bool ≡ (∨)

instance
by *standard* (*auto simp*: *plus-bool-def times-bool-def one-bool-def zero-bool-def*)

end

We now show that infinite Boolean matrices form a Boolean algebra.

lemma *le-funI2*: $(\bigwedge i\ j.\ f\ i\ j \leq g\ i\ j) \Longrightarrow f \leq g$
by (*metis le-funI*)

interpretation *matrix-ba*: *boolean-algebra* $\lambda f\ g.\ f \sqcap g^c$ *mat-complement* (⊓) (≤) (<) *mat-add mat-zero mat-top*
by *standard* (*force intro*!: *le-funI simp*: *mat-inter-def plus-bool-def mat-add-def mat-zero-def zero-bool-def mat-top-def mat-complement-def*)+

We continue working towards the main result of this section, that infinite Boolean matrices form a relation algebra.

lemma *mat-mult-var*: $(f \otimes g) = (\lambda i\ j.\ \sum\ \{(f\ i\ k) * (g\ k\ j) \mid k.\ k \in UNIV\})$
by (*rule ext*)+ (*simp add*: *mat-mult-def*)

The following fact is related to proving the last relation algebra axiom in the matrix model. It is more complicated than necessary since finite infima are not well developed in Isabelle. Instead we translate properties of finite infima into properties of finite suprema by using Boolean algebra. For finite suprema we have developed special-purpose theorems in the Kleene algebra files.

lemma *mat-res-pointwise*:
fixes *i j* :: *'a*::*finite*
and *x* :: (*'a*, *'a*, *bool*) *matrix*
shows $(x^\dagger \otimes (x \otimes y)^c)\ i\ j \leq (y^c)\ i\ j$
proof −
have $\sum\{(x^\dagger)\ i\ k \wedge ((x \otimes y)^c)\ k\ j \mid k.\ k \in UNIV\} \leq (y^c)\ i\ j \longleftrightarrow (\forall k.\ ((x^\dagger)\ i\ k \wedge ((x \otimes y)^c)\ k\ j) \leq (y^c)\ i\ j)$
by (*subst sum-sup*) *auto*
also have ... $\longleftrightarrow (\forall k.\ ((x^\dagger)\ i\ k \wedge -\ (x \otimes y)\ k\ j) \leq (y^c)\ i\ j)$
by (*simp only*: *mat-complement-def*)
also have ... $\longleftrightarrow (\forall k.\ (x^\dagger)\ i\ k \leq ((y^c)\ i\ j \vee (x \otimes y)\ k\ j))$
by *auto*
also have ... $\longleftrightarrow (\forall k.\ (x^\dagger)\ i\ k \leq (-\ y\ i\ j \vee (x \otimes y)\ k\ j))$

by (*simp only*: *mat-complement-def*)
also have $\ldots \longleftrightarrow (\forall k.\ ((x^\dagger)\ i\ k \wedge y\ i\ j) \leq (x \otimes y)\ k\ j)$
by *auto*
also have $\ldots \longleftrightarrow (\forall k.\ (x\ k\ i \wedge y\ i\ j) \leq (x \otimes y)\ k\ j)$
by (*simp add*: *mat-transpose-def*)
also have $\ldots \longleftrightarrow (\forall k.\ (x\ k\ i \wedge y\ i\ j) \leq \sum \{x\ k\ l \wedge y\ l\ j\ |l.\ l \in UNIV\})$
by (*simp add*: *mat-mult-def times-bool-def*)
also have $\ldots \longleftrightarrow (\forall k.\ \sum \{x\ k\ i \wedge y\ i\ j\} \leq \sum \{x\ k\ l \wedge y\ l\ j\ |l.\ l \in UNIV\})$
by *simp*
also have $\ldots \longleftrightarrow$ *True*
by (*intro iffI TrueI allI sum-intro*[*rule-format*]) *auto*
moreover have $(x^\dagger \otimes (x \otimes y)^c)\ i\ j = \sum \{(x^\dagger)\ i\ k \wedge ((x \otimes y)^c)\ k\ j\ |k.\ k \in UNIV\}$
by (*subst mat-mult-def*) (*simp add*: *times-bool-def*)
ultimately show *?thesis*
by *auto*
qed

Finally the main result of this section.

interpretation *matrix-ra*: *relation-algebra* $\lambda f\ g.\ f \sqcap g^c$ *mat-complement* $(\sqcap)$ $(\leq)$ $(<)$ $(\oplus)$ $\lambda i\ j.$ *False* τ $(\otimes)$ *mat-transpose* ε
proof
fix $x\ y\ z :: 'a{::}finite \Rightarrow 'a \Rightarrow bool$
show $(\lambda(i{::}'a)\ j{::}'a.\ False) \leq x$
by (*metis predicate2I*)
show $x \sqcap x^c = (\lambda i\ j.\ False)$
by (*metis matrix-ba.bot.extremum matrix-ba.inf-compl-bot rev-predicate2D*)
show $x \oplus x^c = \tau$
by (*fact matrix-ba.sup-compl-top*)
show $x \sqcap y^c = x \sqcap y^c$
by (*fact refl*)
show $x \otimes y \otimes z = x \otimes (y \otimes z)$
by (*metis mat-mult-assoc*)
show $x \otimes \varepsilon = x$
by (*fact mat-oner*)
show $x \oplus y \otimes z = (x \otimes z) \oplus (y \otimes z)$
by (*fact mat-distr*)
show $(x^\dagger)^\dagger = x$
by (*simp add*: *mat-transpose-def*)
show $(x \oplus y)^\dagger = x^\dagger \oplus y^\dagger$
by (*simp add*: *mat-transpose-def mat-add-def*)
show $(x \otimes y)^\dagger = y^\dagger \otimes x^\dagger$
by (*simp add*: *mat-transpose-def mat-mult-var times-bool-def conj-commute*)
show $x^\dagger \otimes (x \otimes y)^c \leq y^c$
by (*metis le-funI2 mat-res-pointwise*)
qed

end

References

[1] A. Armstrong, G. Struth, and T. Weber. Programming and automating mathematics in the Tarski-Kleene hierarchy. 2014. To appear in Journal of Logic and Algebraic Programming.

[2] R. Maddux. *Relation Algebras*, volume 150 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science, 2006.

[3] G. Schmidt and T. Ströhlein. *Relationen und Graphen*. Springer, 1987.

[4] A. Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6:73–89, 1941.