Programming and Automating Mathematics in the Tarski-Kleene Hierarchy

Alasdair Armstrong[a], Georg Struth[a,*], Tjark Weber[b]

[a]*Department of Computer Science, The University of Sheffield, UK*
[b]*Department of Information Technology, Uppsala University, Sweden*

Abstract

We present examples from a reference implementation of variants of Kleene algebras and Tarski's relation algebras in the theorem proving environment Isabelle/HOL. For Kleene algebras we show how models can be programmed, including sets of traces and paths, languages, binary relations, max-plus and min-plus algebra, matrices, formal power series. For relation algebras we discuss primarily proof automation in a comprehensive library and present an advanced formalisation example.

Keywords: formalised mathematics, interactive theorem proving, Kleene algebra, relation algebra

Dedicated to Professor Gunther Schmidt on the occasion of his 75th birthday.

1. Introduction

The advancement of relational mathematics as a conceptual and methodological tool across the sciences has been the main impetus of Gunther Schmidt's research for many decades. His articles and books elaborate the fundamental role of the calculus of relations for modelling and reasoning about discrete systems. Having experienced as a young mathematician how computers revolutionised numerical and engineering mathematics, he set out to pioneer a similar approach within his own field of interest: long before model checkers became an industry standard and complex mathematical theorems could be verified by machines, he thought about computer software for programming relational mathematics and automating relational proofs.

The *Munich School*, under his influence, developed a wide range of relational tools, including the RelView system for manipulating finite relations [1] and formalisations of relation algebras in theorem proving environments; see Wolfram Kahl's article [2] for an insider's overview.

Today, relational approaches play a fundamental role in computer science—in program semantics, logics of programs, formal program development, databases and beyond. New applications in engineering or the social sciences are emerging.

In the spirit of the Munich School, this paper presents a series of examples in programming and automating mathematics. They are drawn from a reference formalisation that integrates Tarski's relation algebra with variants of Kleene algebras, which are closely related, in the theorem proving environment Isabelle/HOL. The formalisation of variants of Kleene algebras and relation algebras, including all algebras, models and theorems discussed in this paper, is available from the Archive of Formal Proofs [3, 4]. We have implemented these algebras as a structured modular hierarchy using Isabelle's axiomatic type classes. Starting from (join-)semilattices, we have axiomatised variants of dioids with and without certain distributivity and unit axioms and expanded them by axioms for the Kleene star. Since many models of interest have a richer algebraic structure, we have also included action algebras [5], which add operations of residuation. We

*Corresponding author
Email addresses: `a.armstrong@sheffield.ac.uk` (Alasdair Armstrong), `g.struth@sheffield.ac.uk` (Georg Struth), `tjark.weber@it.uu.se` (Tjark Weber)

have expanded Kleene algebras further to omega algebras with an operation of infinite iteration and, finally, to Tarski's relation algebras and relation algebras with a star or reflexive transitive closure operation. From this basis, further axiomatic variants can easily be implemented. The first steps towards this hierarchy have already been documented [6], so our review in this article can be brief.

Instead, for Kleene algebras, we discuss the programming approach for the most important models. These include powersets over a monoid, (regular) languages, binary relations, sets of program traces in which state and action symbols alternate, sets of paths in graphs, max-plus and min-plus algebras. We also show that matrices over dioids form dioids (implementing the Kleene star is tedious), and that formal power series from free monoids into Kleene algebras form Kleene algebras. All of these models have important applications in computer science.

In the context of relation algebras, we explain how Isabelle's recent integration of external automated theorem provers support the rapid semi-automatic development of a comprehensive library based on the standard text books by Schmidt and Ströhlein [7], Schmidt [8] and Maddux [9]. In fact, calculational proofs which eminent mathematicians found worth publishing some decades ago are often fully automatic with this approach. As an advanced formalisation example we explore the Munich axiomatisation of direct relational products [7]. In addition, we have programmed the binary relation and Boolean matrix model of relation algebra.

While automation is desirable for applications and useful for mathematical explorations, it is not the primary goal of our reference implementation. For Kleene algebras, we typically present readable proofs at text book granularity, using Isabelle's proof language Isar [10]. Apart from some idiosyncrasies our formalisation could serve as a comprehensive introduction to reasoning in Kleene algebras. A similar approach is planned for relation algebra.

The integration of relation algebra and Kleene algebras achieves a uniform approach in which all theorems about the Kleene star become automatically available for a relational reflexive transitive closure operation. This is mandatory for programming applications. By linking the algebras in the hierarchy with various models, the development of the latter is significantly streamlined and unified. With algebraic libraries, only a few axioms must be checked to make abstract theorems automatically available. Large parts of Isabelle's library for binary relations, for instance, become redundant relative to a modular algebraic development. In programming applications, the link between algebras and models couples control flow with data flow and supports seamless reasoning, for instance, about changes and updates in program stores or state spaces.

2. Algebraic Hierarchy

Kleene algebras expand dioids (idempotent semirings) by an operation of Kleene star or finite iteration. Dioids, in turn, expand join-semilattices by an operation of multiplication. Tarski's relation algebras are dioids as well as Boolean algebras which share the join operation and the least element. In addition they axiomatise an operation of relational converse. In other words, the relational operations of complementation, meet and conversion are missing in Kleene algebras whereas the Kleene star is missing in relation algebra.

This section briefly overviews this theory hierarchy and serves as a road map for the formal development in the Archive which defines more than 40 related type classes. The main structures in the Kleene algebra hierarchy are shown in Figure 1. Isabelle already contains a vast amount of mathematical structures and libraries, which any formalisation of new structures can and should use. Relation algebras, for instance, can be based on Isabelle's formalisation of Boolean algebras as well as on the variants of semirings formalised within the Kleene algebra hierarchy. These semirings, in turn, can be based on Isabelle's formalisation of abelian semigroups. By these integrations, all facts proved for the basis structures become automatically available in all expansions. In a similar fashion, the binary relations models of Kleene algebra and relation algebra is based on Isabelle's extensive library for binary relations. As a general rule, algebraic structures should therefore be designed in a modular fashion in order to enhance reuse and expandability.

A *join-semilattice* $(L, +)$ consists of a set L with an associative, commutative and idempotent operation of join denoted by $+$. Join semilattices are also posets with order relation defined by

$$x \leq y \longleftrightarrow x + y = y,$$

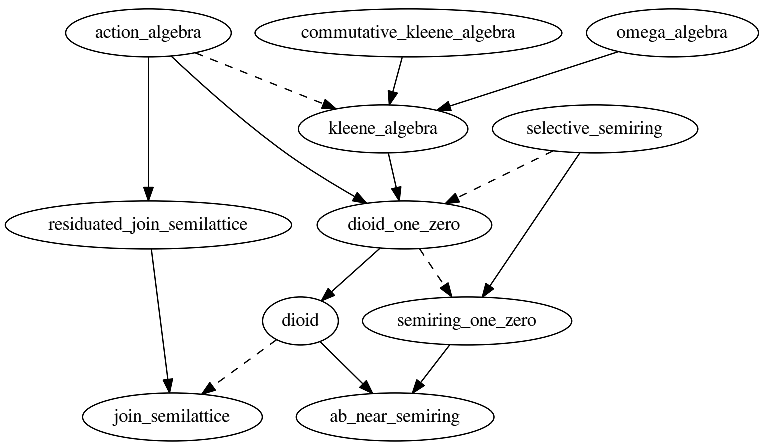

Figure 1: Simplified Kleene algebra hierarchy. Solid arrows denote syntactic class extensions, while dashed arrows indicate subclass relationships established by proof.

such that the supremum $x + y$ exists for all $x, y \in L$. It follows that addition is order-preserving or isotone, that is, $x \le y \longrightarrow z+x \le z+y$. A join-semilattice is *bounded* if it has an additive unit 0 satisfying $x+0 = x$. This unit is the least element with respect to the order. Every bounded join-semilattice is therefore also a commutative monoid.

Adding a multiplicative semigroup structure yields variants of near semirings (or seminearrings). In general, a near semiring consists of an additive and a multiplicative semigroup which interact via one single distributivity law (left or right). We are only interested in near semirings in which addition is commutative and the right distributivity law holds. Formally, an *abelian near semiring* is a structure $(S, +, \cdot)$ such that $(S, +)$ is a commutative (abelian) semigroup, $(S, \cdot)$ a semigroup and the right distributivity axiom

$$(x + y) \cdot z = x \cdot z + y \cdot z$$

holds. A *semiring* is an abelian near semiring which also satisfies the left distributivity law

$$x \cdot (y + z) = x \cdot y + x \cdot z.$$

A *near dioid* is an abelian near semiring whose additive reduct is a join-semilattice. Hence every near dioid can be ordered, and multiplication is right isotone: $x \le y \longrightarrow x \cdot z \le y \cdot z$. A *pre-dioid* is a near dioid in which multiplication is left isotone:

$$x \le y \longrightarrow z \cdot x \le z \cdot y.$$

A *dioid* is a near dioid in which also the left distributivity law holds. Hence dioids are additively idempotent semirings. All variants of near semirings can be expanded by additive units 0 and multiplicative units 1. We have separately formalised 0 as a left annihilator $(0 \cdot x = 0)$ and an annihilator (also $x \cdot 0 = 0$), since right annihilation fails in interesting models. An element x, for instance, could represent an infinite computation, which should rather satisfy $x \cdot 0 = x$ than $x \cdot 0 = 0$. Finally, an abelian near semiring S (with or without units) is *selective* if either $x + y = x$ or $x + y = y$ holds for all $x, y \in S$. All selective abelian near semirings are near dioids with linear order $\le$. The definition extends to semirings and dioids. Max-plus and min-plus algebras [11] are examples of selective semirings.

Next we have formalised the Kleene star over the family of near-dioids with 1. A *left near Kleene algebra* is a structure $(K, +, \cdot, ^*, 1)$ such that $(K, +, \cdot, 1)$ is a near dioid with 1 and the star satisfies the *left unfold axiom* and the *left induction axiom*

$$1 + x \cdot x^* \le x^*, \qquad z + x \cdot y \le y \longrightarrow x^* \cdot z \le y.$$

This implies that x^* is the least (pre)fixpoint of the function $\lambda y.\, 1 + x \cdot y$. The same axioms are used to expand left pre-dioids and dioids with 1 to *left pre-Kleene algebras* and *left Kleene algebras* respectively. We have further expanded these structures by additive units 0. *Near Kleene algebras*, *pre-Kleene algebras* and *Kleene algebras* can be obtained by adding a *right induction axiom*

$$z + y \cdot x \leq y \longrightarrow z \cdot x^* \leq y$$

to the respective variant of left Kleene algebra. Finally, variants of *commutative Kleene algebras* are obtained by adding the axiom $x \cdot y = y \cdot x$.

Near semirings form the basis of process algebras; pre-Kleene algebras have been used for modelling games and probabilistic protocols; the right annihilation axiom has been omitted in the context of total program correctness.

(Left) Kleene algebras use Horn formulas to axiomatise the equational theory of regular expressions. Action algebras provide a finite equational axiomatisation in a larger signature, which is supported by many important models. A *residuated join-semilattice* is a structure $(L, +, \cdot, \leftarrow, \rightarrow)$ such that $(L, +)$ is a join-semilattice, $(S, \cdot)$ is a semigroup and *left* and *right residuation* are axiomatised by the Galois connections

$$x \cdot y \leq z \longleftrightarrow x \leq z \leftarrow y, \qquad x \cdot y \leq z \longleftrightarrow y \leq x \rightarrow z.$$

There exists an equivalent, though less elegant, equational axiomatisation. An *action algebra* is a residuated join-semilattice that is also a dioid, which is expanded by a star operation satisfying the two star axioms

$$1 + x + x^* \cdot x^* \leq x^*, \qquad 1 + x + y \cdot y \leq y \longrightarrow x^* \leq y.$$

These are inspired by the reflexive transitive closure operation, but those of Kleene algebra can equally be used. Thus, in particular, every action algebra is a Kleene algebra. In action algebras, the star can as well be axiomatised by the equations

$$1 + x + x^* \cdot x^* \leq x^*, \qquad x^* \leq (x + y)^*, \qquad (x \rightarrow x)^* \leq x \rightarrow x.$$

This yields the finite equational axiomatisation mentioned.

Finally, in the Kleene algebra hierarchy, an operation of infinite iteration can be added to Kleene algebras. A *left omega algebra* is a left Kleene algebra (with zero) expanded by an omega operation which satisfies the *unfold axiom* and the *coinduction axiom*

$$x^\omega \leq x \cdot x^\omega, \qquad y \leq z + x \cdot y \longrightarrow y \leq x^\omega + x^* \cdot z.$$

Omega algebras can be obtained by expanding Kleene algebras accordingly. We currently do not know any interesting properties that would hold in all omega algebras, but not in all left omega algebras.

Relation algebras form a different dioid expansion. A *relation algebra* is a structure $(R, +, \cdot, -, ;, ^\smile, 0, 1, 1')$ such that $(R, +, \cdot, -, 0, 1)$ is a Boolean algebra, $(R, +, ;, 0, 1')$ is a dioid. Moreover the operation of conversion is involutive, $x^{\smile\smile} = x^\smile$, additive, $(x + y)^\smile = x^\smile + y^\smile$, contravariant, $(x; y)^\smile = y^\smile; x^\smile$ and satisfies the residuation property $x^\smile; -(x; y) \leq -y$.

While axiomatic reasoning in dioids is trivial and in Boolean algebra is rather simple, the interaction of the star or omega with the dioid operations and that of conversion with the Boolean and diod operations adds a significant level of complexity and requires a certain level of experience.

Due to our focus on models of Kleene algebras and on relation algebras, most of the variants of semirings and Kleene algebras mentioned in this section are not essential to the remainder of this article, but nevertheless they play an important role in the modular design of the Tarski-Kleene hierarchy with a view on applications. Most of this article is based on dioids with additive and multiplicative units, Boolean algebras, Kleene algebras, omega algebras and relation algebras.

3. Formalising Algebraic Hierarchies in Isabelle/HOL

Isabelle's axiomatic type class infrastructure [12] is the standard tool for formalising algebraic hierarchies like that depicted in Section 2. Details can be found in the archive proof documents [3, 4]. Additional variants could be implemented easily from this basis. As an example, we have implemented abelian near semirings as

```
class ab_near_semiring = ab_semigroup_add + semigroup_mult +
  assumes right_distrib: "(x + y) · z = x · z + y · z"
```

expanding Isabelle's existing type classes for additive abelian semigroups and multiplicative semigroups, and adding the right distributivity axiom for the interaction between addition and multiplication. They can be expanded to abelian near semirings with a multiplicative unit 1, provided by class *one*, as follows:

```
class ab_near_semiring_one = ab_near_semiring + one +
  assumes mult_onel: "1 · x = x"
  and mult_oner: "x · 1 = x"
```

Simple relationships between algebras can be captured by subclass statements. The fact that selective near semirings are linear orders, for instance, is captured by the following statement and proof.

```
subclass (in selective_near_semiring) linorder
  ⟨proof⟩
```

The statement in braces denotes that the linear order axioms need to be verified in the context of selective near semirings. The proof obligations are dictated by Isabelle. They are not shown in this article, which is indicated by writing ⟨*proof*⟩. This subclass statement makes all Isabelle theorems for linear orders available in selective near semirings. We have carefully used expansions and subclassing to design our algebraic hierarchy and propagate theorems.

Our second example shows the implementation of relation algebra as an expansion of Isabelle's built-in axiomatic type class for Boolean algebra.

```
class relation_algebra = boolean_algebra +
  assumes comp_assoc: "(x ; y) ; z = x ; (y ; z)"
  and comp_unitr: "x ; 1' = x"
  and comp_distr: "(x + y ) ; z = x ; z + y  ; z "
  and conv_invol: "(x˘ )˘  = x"
  and conv_add: "(x + y)˘  = x˘  + y˘ "
  and conv_contrav: "(x ; y)˘  = y˘  ; x˘ "
  and comp_res: "x ˘  ; -(x ; y) ≤ -y"
```

The proof that every relation algebra is a dioid now requires matching the signature of relation algebra with that of dioids, which in turn uses that of near-semirings. Concretely, the relation algebraic operation of + matches the dioid + and the relation algebraic ; matches the operation · of the dioid. This can be captured by a sublocale statement in Isabelle.

```
sublocale relation_algebra ⊆ dioid_one_zero "(op +)" "(op ;)" "1'" "0" "(op ≤)" "(op <)"
  ⟨proof⟩
```

When writing a sublocale statement, Isabelle displays a list of operations to be matched. In this particular case these are the operations of addition, multiplication, one, zero, of the dioid and the associated operations of partial and strict order. The user then has to type the corresponding list of matching relation-algebraic operations. In the above list, all operations except relational composition have been overloaded. As with a subclass statement, Isabelle then dictates that the axioms of dioids with one and zero be proved in the context of relation algebra while matching the operations as indicated.

Our hierarchy also contains an axiomatic type class for relation algebras with a star operation. It is obtained by adding the star axioms of Kleene algebra to those of relation algebra. In the context of relation algebras, the star implements a reflexive transitive closure operation. Based on the formalised subclass relationships between relation algebras and dioids, the sublocale proof that every relation algebra with star is also a Kleene algebra is trivial: only the star axioms need to be verified and these are identical in both classes.

```
class relation_algebra_rtc = relation_algebra + star_op +
  assumes rtc_unfoldl: "1' +  x ; x⋆ ≤ x⋆"
  and rtc_inductl: "z + x ; y ≤ y ⟶ x⋆ ; z ≤ y"
  and rtc_inductr: "z + y ; x ≤ y ⟶ z ; x⋆ ≤ y"

sublocale relation_algebra_rtc ⊆ kleene_algebra ⟨signature list⟩
  ⟨proof⟩
```

By our design of the Tarski-Kleene hierarchy with type classes and locales, theorems are propagated upwards in the hierarchy. All theorems from Kleene algebra, for instance, become automatically available in the context of relation algebras with star; all theorems of Boolean algebra become automatically available in the context of relation algebra.

Schmidt and Ströhlein's book, for instance, contains many examples of reflexive transitive closure theorems in relation algebra which are now inherited from Kleene algebra. One of their examples is the Church-Rosser theorem: they prove by induction on powers of x that the Church-Rosser property $(x + x^{\smile})^* = x^*;x^{\smile *}$ follows from confluence $x^{\smile *};x^* \leq x^*;x^{\smile *}$ in a relation algebra that is based on a complete Boolean algebra. Isabelle recognises this theorem as an instance of the fact that

$$y^* \cdot x^* \leq x^* \cdot y^* \longrightarrow (x + y)^* = x^* \cdot y^*$$

holds already in Kleene algebra—hence without an explicit induction.

First steps towards a similar implementation have already been undertaken by Wolfram Kahl [2] about a decade ago. The main differences are as follows.

First, Kahl has implemented a hierarchy of typed or heterogeneous algebras. While this is preferable for some modelling purposes, the proofs of many basic relational theorems can in fact be separated into one phase performing some trivial existence checks on objects and another one which performs equational reasoning in the untyped setting. In Isabelle, in addition, axiomatic type classes can no longer be used in the presence of typed relations; they must be replaced by locales. Definitions based on partial functions cause additional problems in Isabelle. Proofs become more tedious and much less automatic.

Second, our implementation relies heavily on a recent integration of automated theorem provers and counterexample generators in Isabelle, which were not available ten years ago. In the untyped setting, in particular, this typically allows proofs at textbook granularity or even fully automated proofs. Our reference implementation of Kleene algebras aims at readable textbook-style proofs instead of proof automation. The proof

```
lemma star_trans_eq: "x⋆ · x⋆ = x⋆"
proof (rule antisym) — this splits an equation into two inequalities
  have "x⋆ + x · x⋆ ≤ x⋆"
    by (metis add_lub eq_refl star_1l)
  thus "x⋆ · x⋆ ≤ x⋆"
    by (metis star_inductl)
  next show "x⋆ ≤ x⋆ · x⋆"
    by (metis mult_isor mult_onel star_ref)
qed
```

which uses Isabelle's proof scripting language Isar, provides a simple example. Algebraists can easily follow the high-level proof structure, while individual proofs steps have been verified by Isabelle's automated provers. This turns our reference implementation almost into a primer on algebraic reasoning in variants of Kleene algebras. A similar approach for relation algebras is planned for the future.

As a side effect, our proofs are robust to changes and easy to maintain. We have also used Isabelle's counterexample generators Nitpick and Quickcheck [13] for separating algebras. For instance, Nitpick refutes the star slide rule $x^* \cdot (y \cdot x^*)^* = (x^* \cdot y)^* \cdot x^*$ by a six-element counterexample in left pre-Kleene algebras (it holds in left Kleene algebras). There are some interesting challenge problems for counterexample generators, for instance, to show that not all left Kleene algebras are Kleene algebras. Kozen's counterexample [14] involves transfinite induction.

4. Monoidal and Language Models of Kleene Algebra

The following sections present our formalisation of the most important models of dioids and Kleene algebras. They are well known mathematically, but the functional programming approach to mathematics typical to proof assistants leads to some interesting insights. Most of these models, with the notable excecption of binary relations and Boolean matrices, are not relevant to relation algebra.

This section discusses variants of language models. (Regular) languages are already available in Isabelle (cf. [15]), but the algebraic approach makes our implementation simpler and more generic. While theorems should generally be proved for the weakest axiom systems, models should be built for the strongest ones. With Isabelle's type classes, theorems are propagated down the sub-algebra hierarchy; models are propagated upwards.

First we have shown that the powerset lifting of a monoid with suitably defined operations yields a Kleene algebra. Suppose that $(M, \cdot, 1)$ is a monoid (with elements of type α). Define the following operations on the powerset of M (type α *set* in Isabelle): the multiplicative unit is $1 = \{1\}$ (using overloading); the complex product is, using Isabelle's slightly idiosyncratic set builder notation,

$$X \cdot Y = \{x \cdot y \mid xy.\ x \in X \wedge y \in Y\}$$

for $X, Y \subseteq M$. A simple instantiation proof in Isabelle then establishes the following fact.

Theorem 1. *If $(M, \cdot, 1)$ is a monoid, then so is $(2^M, \cdot, 1)$.*

Instantiations link models with Isabelle's axiomatic type classes. With additive unit $0 = \{\}$ and addition $X + Y$ defined as $X \cup Y$ on powersets it takes little effort to show that $(2^M, +, \cdot, 0, 1)$ forms a dioid. The Kleene star on powersets can be defined as

$$X^* = \bigcup_{n \geq 0} X^n$$

using Isabelle's recursive definition of powers X^n for monoids. This leads to the following instantiation result.

Theorem 2. *If $(M, \cdot, 1)$ is a monoid, then $(2^M, +, \cdot, ^*, 0, 1)$ is a Kleene algebra.*

The proof requires only a few auxiliary lemmas, notably the induction laws for powers

$$z + x \cdot y \leq y \longrightarrow x^n \cdot z \leq y, \qquad z + y \cdot x \leq y \longrightarrow z \cdot x^n \leq y$$

and the star continuity laws

$$X \cdot Y^* = \bigcup_{n \geq 0} X \cdot Y^n, \qquad X^* \cdot Y = \bigcup_{n \geq 0} X^n \cdot Y.$$

Only the two induction laws need user-guided induction; for the other laws induction is hidden in lemmas deep down in the Isabelle libraries.

Next we have shown that languages under the regular operations form Kleene algebras. Obviously, languages are sets of words, and words under concatenation and the empty word form monoids. Hence languages form Kleene algebras by Theorem 2. This simple argument by extension of mathematical structure is seamlessly supported by Isabelle. Here, words are usually programmed as lists—sets of which are, in fact, isomorphic to the free dioids—so it suffices to verify the simple algebraic fact that lists under concatenation form monoids.

```
instantiation list :: (type) monoid_mult
  ⟨proof⟩

interpretation lan_kleene_algebra:
  kleene_algebra "op +" "op ·" "1::α lan" "0" "op ⊆" "op ⊂" star
  ⟨proof⟩
```

The instantiation statement, which captures this fact, formally links into Theorem 1 and makes the interpretation statement—languages under the regular operations form Kleene algebras—a trivial corollary of the more abstract Theorem 2 in Isabelle. The interpretation statement allows matching the Kleene algebra signature with the corresponding operations on languages. As in the case of sublocale or subclass statements, Isabelle dictates the proof obligations. In this case, it must be checked that languages satisfy the star axioms of Kleene algebras; all other axioms being subsumed by Theorem 2.

Finally, we have shown that *regular* languages under the regular operations form Kleene algebras. Based on an existing data type for regular expressions [15] we have formalised the standard interpretation map from regular expressions to languages and introduced regular languages as a subtype of languages, as induced by the interpretation map. Isabelle's quotient package [16] supports lifting the regular operations from languages to regular languages, and transfers the facts needed to show that regular languages form Kleene algebras. Details can be found in the Archive proof document.

The two language models have been obtained by and large automatically from the abstract algebraic monoid model. Automation was supported by Isabelle's type class and locale mechanisms, by Isabelle's libraries, e.g., for numbers, monoids, powers, sets, and, last but not least, by the link with algebra. Instantiations and interpretations make all abstract theorems of Kleene algebra available in the language models. Many inductive proofs about the star in languages have thus been reduced to simple equational reasoning.

A decision procedure for regular expressions is available in Isabelle/HOL [15], but a formalised completeness proof is required to use it safely for equational reasoning in Kleene algebra. Currently we use this procedure only as an oracle. The completeness proof is based on rather tedious matrix encodings of automata constructions [17]; it has so far only been formalised in Coq [18].

5. Binary Relation Model

Due to Isabelle's excellent libraries for binary relations, it is fully automatic to show that binary relations of a given type form a dioid under the operations of union and relational composition, with the empty relation as additive identity and the diagonal relation as multiplicative identity.

```
interpretation rel_dioid: dioid_one_zero "op ∪" "op O" Id "{}" "op ⊆" "op ⊂"
  by (unfold_locales, auto)
```

In the interpretation proof, the unfold statement generates the proof obligations whereas Isabelle's built-in auto tactic discharges all of them at once. For an algebraic setting this is rather exceptional since tactics such as simp, auto or blast are usually not able to compete with state-of-the-art automated theorem provers which are called by Isabelle's sledgehammer tactic.

As previously, the relational Kleene star is defined as a sum of powers: $R^* = \bigcup_{n\geq 0} R^n$. The interpretation proof that relations form Kleene algebras

```
interpretation rel_kleene_algebra: kleene_algebra "op ∪" "op O" Id "{}" "op ⊆" "op ⊂" rtrancl
  ⟨proof⟩
```

then follows precisely those for monoids and languages. It uses the same kind of continuity laws; only showing that R^* is equal to Isabelle's reflexive transitive closure operation is required in addition (Isabelle uses a specific relational power operation which is not linked properly with its monoidal one). In addition, the proof is once more modular relative to the dioid results in that only the star axioms need to be verified.

Despite the striking similarity to the language proofs, no powerset lifting is involved. One could, however, define a partial composition operation directly on ordered pairs, identify appropriate units and lift these operations to the powerset level of relations. This mathematically rather unconventional approach might allow us to further unify the programming of models in Isabelle. We discuss this in more detail below.

We have extended the relational model to omega algebras. The omega of a relation relates all elements in the relation's domain from which an infinite chain starts with all other elements [19]. It is most naturally defined coinductively.

```
coinductive_set omega :: "(α × α) set → (α × α) set" for R where
  "⟦(x,y) ∈ R; (y,z) ∈ omega R⟧ ⟹ (x,z) ∈ omega R"
```

Isabelle automatically derives a coinduction rule for the omega operation. We have proved a slightly simpler variant of this rule.

```
lemma omega_weak_coinduct:
  "P x z ⟹ (⋀x z. P x z ⟹ ∃y. (x,y) ∈ R ∧ P y z) ⟹ (x,z) ∈ omega R"
  by (metis omega.coinduct)
```

The unfold and coinduction axiom of omega algebra then follow by coinduction.

Finally we have verified the obvious: binary relations form relation algebras.

```
interpretation rel_relation_algebra: relation_algebra ⟨signature list⟩
  ⟨proof⟩
```

Once more, due to Isabelle's excellent libraries for binary relations, the proof is fully automatic. By this result a large amount of concrete reasoning with binary relations is captured algebraically. This nicely demonstrates the unifying power of algebra. Much of explicit inductive reasoning about (reflexive) transitive closures of binary relations and the star in formal language theory has been subsumed by abstract equational reasoning, using the same algebraic laws for both models. A rationalisation of labour has thereby been achieved.

6. Trace and Path Models of Kleene Algebra

Sets of traces form another interesting model of Kleene algebras. A previous implementation has briefly been discussed in [6], hence we only sketch the development. Intuitively, a trace represents an execution sequence of a labelled transition system or automaton in which state and action labels alternate, beginning and ending with a state label. For example, $p_1a_1p_2a_2 \dots p_{n-1}a_{n-1}p_n$ is a trace if the p_i are state labels and the a_i are action labels. *Trace fusion* merges two traces xp and qy into the trace xpy if $p = q$ and is undefined otherwise. This partial operation is lifted to a complex product on sets of traces as in the case of monoids. The Kleene star X^* of a set of traces can be defined again as a union of powers $\bigcup_{n\geq 0} X^n$. It can then be shown that sets of all traces built from a set of action and state labels under these operations together with set union, the empty set of traces as the additive unit and the set of all traces of length one (e.g. single state labels) as the multiplicative unit form Kleene algebras.

Traces are implemented as pairs of type $\pi \times (\alpha \times \pi)$ *list*, where π denotes the type of state labels and α the type of action labels. The fusion and complex product are formalised as follows.

```
definition "t_fusion x y ≡
  if last x = fst y then (fst x, snd x · snd y) else undefined"
```

```
definition "X · Y ≡ {t_fusion x y| x y. x ∈ X ∧ y ∈ Y ∧ last x = fst y}"
```

The operator · in the definition of the fusion product denotes list concatenation. We have shown by case analysis with respect to definedness that traces under fusion form a partial monoid. Due to Isabelle/HOL's limitations with partiality, we fully capture undefinedness only when lifting trace fusion to powersets; the last condition in the set comprehension guarantees that only well-defined trace products contribute. The proof of the interpretation statement

```
interpretation trace_kleene_algebra: kleene_algebra "op ∪" "op ·" t_one "{}" "op ⊆" "op ⊂" t_star
  ⟨proof⟩
```

follows those for languages and relations, but is more tedious and not fully subsumed by Theorem 1 due to partiality.

Path models are very similar to trace models, but only state labels are considered; action labels have been forgotten. These capture, for instance, sets of paths in a (di)graph or transition system. Our development follows essentially [19]. We have separately implemented models with and without the empty path.

Paths including the empty path have been implemented as lists. *Path fusion* is a partial operation similarly to trace fusion. In functional programming style this is written as follows.

```
fun p_fusion where
  "p_fusion [] _ = []"
| "p_fusion _ [] = []"
| "p_fusion ps (q # qs) = ps · qs"
```

For convenience, our implementation of this product is a total recursive function, and we can again establish a monoidal structure on paths. Undefined products are once more filtered out in the definition of complex product, but the filtering condition is more involved than for traces.

```
definition "p_filter ps qs ≡ (ps = [] ∧ qs = []) ∨ (ps ≠ [] ∧ qs ≠ [] ∧ last ps = hd qs)"
```

```
definition "X · Y ≡ {p_fusion ps qs| ps qs. ps ∈ X ∧ qs ∈ Y ∧ p_filter ps qs}"
```

The complexity of filtering shows up in the proofs of the multiplicative monoid laws at the powerset level. They require a case analysis over the structure of paths. From these facts it is then easy to prove that sets of paths form dioids. The Kleene star is again defined as a union of powers. The proof that sets of paths form Kleene algebras is similar to that for traces.

```
interpretation path_kleene_algebra: kleene_algebra "op ∪" "op ·" p_one "{}" "op ⊆" "op ⊂" p_star
  ⟨proof⟩
```

Alternatively, based on non-empty lists, we have build a path model without the empty path. This yields simpler definitions and proofs, but requires implementing a basic library for non-empty lists. The fusion product now becomes a primitive recursive function over paths. The filtering condition in the complex product is simply that the last element of the first path and the first element of the second one must be equal. Verifying the dioid axioms is now almost automatic. The interpretation proofs follows the approach for the other models.

The similarity between the path and trace models is particularly apparent. They are all based on a powerset lifting of partial monoids. Moreover, languages, relations and paths can be obtained from traces by forgetting part of the structure. This can be formalised in terms of homomorphisms (forgetful functors) and Galois connections [19]. Path models are isomorphic to trace models with one single action symbol; language models are isomorphic to path models with one single state symbol (the fusion product is now total); relations are obtained by projecting on the first and the last element of a trace. On the one hand it is therefore certainly possible to derive all these models from one generic algebraic construction in Isabelle. On the other hand, however, it is not clear that the practical gain of this abstraction would justify the mathematical machinery involved and the effort in dealing with partiality.

7. Matrices

Matrices form an important model of Kleene algebras due to their relevance for completeness proofs [17]. They can be used for encoding computing systems via automata or transition systems. We provide two models; one for potentially infinite matrices and one for the more usual $m \times n$-matrices of fixed dimension. We mainly discuss the first model because it is simpler, but similar to the second.

In the infinite case, matrices are functions from two index sets into some suitable algebra, which we assume to be a dioid. The only restriction is that, in the matrix product, summation ranges over a finite index set. A more general more notion of product could easily be obtained, but it would leave the realm of semirings where only finite sums are available. It can then be shown that dioids are closed under matrix formation.

We have defined the type (α, β, γ) *infmatrix* as a synonym for functions of type $\alpha \to \beta \to \gamma$. The unit and zero matrix are formalised as follows.

definition *"ε i j ≡ (if i = j then 1 else 0)"*

definition *"δ i j ≡ 0"*

In the definition of ε, the equality comparison forces the two index types to be the same, which results in "square" matrices. The target type is usually constrained to be a dioid. Matrix addition is defined in the usual point-wise way.

definition *"(f ⊕ g) i j ≡ (f i j) + (g i j)"*

It is then straightforward to show that matrices over a dioid form a bounded join-semilattice with respect to addition. As already mentioned, matrix multiplication imposes a finiteness constraint on the index type over which the sum is taken.

definition *"(f ⊗ g) i j ≡* $\sum_{\mathtt{k::\beta::finite}}$ *{(f i k) · (g k j)}"*

In the dioid setting, addition is idempotent, and the finite sum $\sum_{\mathtt{k}}$ is actually a finite supremum. Its properties and preconditions are subtly different from the usual Isabelle setsum operation, and we have designed a specific library for finite suprema with around 30 facts. The finiteness constraint in the product is needed because infinite sums need not exist in a dioid. We could then show the remaining dioid laws, some of which are rather involved. The most complex one is associativity of matrix multiplication, which relies on auxiliary lemmas for finite suprema and rearranging sums. The Isar proofs of these facts essentially translate manual proofs step by step. Given these individual algebraic laws, showing that dioids are closed under matrix formation is then automatic.

interpretation *matrix_dioid: dioid_one_zero "op ⊕" "op ⊗" ε δ "op ≤" "op <"*
  ⟨*proof*⟩

We have also formalised the more standard case of $n \times m$-matrices over dioids. This is similar to a development in the Isabelle/HOL library for multivariate analysis, but we explicitly define type constructors for (square) matrices and use weaker assumptions on the element types in matrix operations. Formally,

datatype *(α,β,γ) matrix = Matrix "β atMost → γ atMost → α"*

datatype *(α,β) sqmatrix = SqMatrix "β atMost → β atMost → α"*

where α *atMost* is a finite type whose size is determined by α. We have then defined the usual operations of matrix addition and multiplication on finite (square) matrices, and proved that square matrices over a dioid form a dioid. The approach and proof effort is comparable to the infinite case.

Neither implementation so far includes the star. It is well known that matrices over a Kleene algebra form a Kleene algebra, and this result is useful for various reasons. Its absence is currently a main obstacle for safely using the Isabelle decision procedure for regular expressions for solving Kleene algebra identities. The definition of the star of a (square) matrix is recursive by splitting into appropriate sub-matrices (cf. [17]). Implementing this in Isabelle and verifying the laws of Kleene algebra is rather tedious. An elegant existing implementation in Coq [18] shows the power of dependent types for this purpose.

It is well known that finite relation algebras with n elements are isomorphic to algebras of $n \times n$ Boolean matrices where, in addition to the dioid operations, a top matrix, the intersection of two matrices, complementation of a Boolean matrix and transposition as conversion need to be defined. We have implemented these concepts for infinite Boolean matrices (in the order mentioned) and proved that these matrices form a model of relation algebra.

definition *"τ i j ≡ True"*

definition *"(f ⊓ g) i j ≡ (f i j) · (g i j)"*

definition *"f^c ≡ (λi j. - f i j)"*

definition *"f† ≡ (λi j. f j i)"*

interpretation *matrix_ra: relation_algebra* ⟨*signature list*⟩
⟨*proof*⟩

Only the verification of the residuation axiom—the last axiom in the relation algebra type class—required a certain amount of work in the matrix model. A similar construction for $n \times n$ Boolean matrices can be obtained along these lines.

8. Formal Power Series

Formal power series are functions from free monoids into dioids, Kleene algebras or, more generally, semirings. They are widely applied in language and automata theory, cf. Berstel and Reutenauer's book [20]. Our implementation generalises a formalisation of formal power series, as they are known from combinatorics and ring theory, by Chaieb in the Isabelle/HOL library [21].

We currently focus on Kleene algebras and leave the integration of non-idempotent semirings for future work. Although this may seem mathematically straightforward, this is not entirely trivial from a programming point of view. Dioids can use finite suprema whereas semirings require Isabelle's more general setsums for addition, which are defined for arbitrary commutative monoids. As usual, we program elements of free monoids as lists. Thus, the type of formal power series (with codomain β) is isomorphic to the type of functions from α *list* to β. In contrast, Chaieb's formal power series are functions from the natural numbers.

typedef *(α,β) fps = "{f::α list → β. True}"*
morphisms *$ Abs_fps* ⟨*proof*⟩

The function *Abs_fps* and its counterpart $ are coercions between the types of functions and formal power series.

The dioid operations on formal power series are defined as follows. The zero formal power series is the (isomorphic image of the) function $\lambda n.\, 0$ that maps every monoid element to 0 in the dioid. The multiplicative unit of formal power series, denoted 1 by overloading, maps the monoidal unit (the empty list) to 1 in the dioid and every other element of the monoid to 0:

definition *"1 ≡ Abs_fps (λn. if n = [] then 1 else 0)"*

Addition of formal power series is defined point-wise.

definition *"f + g ≡ Abs_fps (λn. f $ n + g $ n)"*

The product of formal power series is the well known convolution or Cauchy product

$$(f \cdot g)\ n = \sum_{x \cdot y = n} (f\ x) \cdot (g\ y).$$

Programming this in Isabelle requires splitting a list representing an element of a free monoid into all possible prefix/suffix pairs and multiplying with respect to these splittings in the dioid.

definition *"f · g ≡ Abs_fps (λn. ∑{f $ x · g $ y| x y. n = x · y})"*

This summation over all splittings of n is captured by the following definition.

definition *"splitset n ≡ {(x, y)| x y. n = x · y}"*

This set can, of course, be defined more constructively by recursion. Both variants are helpful for proving algebraic properties.

The proof that formal power series form bounded join-semilattices requires little more than unfolding definitions. Establishing the multiplicative monoid properties, in contrast, is tedious, and needs a number of auxiliary lemmas and, once more, facts about finite suprema. The multiplicative left unit law is best proved by case analysis over the structure of lists using the recursive definition of splitset. For the right unit law, however, dual lists would be needed. We have therefore proved it in a more algebraic fashion from properties of finite suprema. Both proofs formalise a simple mathematical insight. Consider the left unit law

$$(1 \cdot f)\ xs = \sum_{ys \cdot zs = xs} (1\ ys) \cdot (f\ zs)$$

for a list xs. The power series 1 maps the empty list to 1 in the dioid, which yields $f\ xs$ in the product. All other lists ys are mapped to 0, which annihilates the product. Hence only $f\ xs$ survives the Cauchy product. The right unit law is dual. The proof of the associativity law relies on a generic lemma for rearranging the sums that occur, similar to the case of matrices. We have combined the proofs of the individual algebraic laws into an instantiation proof that formal power series into a dioid form a dioid.

instantiation *fps :: (type,dioid_one_zero) dioid_one_zero*
  ⟨*proof*⟩

Two approaches of formalising the star of formal power series can be found in the literature, but detailed proofs are usually omitted. First, it can be shown that for *proper* power series, which satisfy $f\ 1 = 0$, the star can be defined as the finite sum $f^*\ x = \sum_{0 \leq i \leq |x|} f^i\ x$. We have formalised the more interesting case of power series into Kleene algebras, where the star can be defined recursively as

$$f^*\ 1 = (f\ 1)^*, \qquad f^*\ x = f^*\ 1 \cdot \sum_{y \cdot z = x, y \neq 1} f\ y \cdot f^*\ z,$$

where $x \neq 1$. We have first defined the star on functions, where we can use Isabelle's package for recursive functions, and then lifted it to the type of formal power series.

```
fun star_fps_rep where
  "star_fps_rep f [] = (f [])*"
| "star_fps_rep f n = (f [])* ·
    ∑{f y · star_fps_rep f z |y z. n = y @ z ∧ y ≠ []}"
```

lift_definition *star_fps :: "('a, 'b) fps ⇒ ('a, 'b) fps"*

```
  is star_fps_rep ..
```

The verification of the star unfold and star induction axioms from these definitions is tedious. It uses case analyses on the structure of the free monoid and a few auxiliary lemmas. This yields the main result of this section.

```
instantiation fps :: (type,kleene_algebra) kleene_algebra
  ⟨proof⟩
```

9. Max-Plus and Min-Plus Algebras

Max-plus and min-plus algebras are important mathematical structures in their own right. A standard reference is Gondran and Minoux's book [11]. Applications include combinatorial optimisation, control theory, algorithm design or internet routing. We have developed the basics of this approach; as far as we know, for the first time in Isabelle. All instances require extensions of number fields by elements $\pm\infty$, either in combination or separately. Separate extensions are currently not available in Isabelle, though they can easily be obtained.

We first discuss max-plus and min-plus algebras over $\mathbb{R}$. The carrier set of the max-plus algebra is the set $\mathbb{R} \cup \{-\infty\}$. The operation of addition is maximum, that of multiplication is addition, the additive unit is minus infinity and the multiplicative unit is zero.

We have defined a datatype for $\mathbb{R} \cup \{-\infty\}$ and extended the (recursive) functions of maximum and addition accordingly. After fixing the units, the instance proof that max-plus algebras over the reals form selective semirings is a straightforward case analysis over the data type; the proof that they also form dioids is then automatic from our abstract algebraic results.

The proofs for min-plus algebras are very similar. Now, the real numbers are extended by $+\infty$, multiplication is minimum, addition is multiplication, the additive unit is $+\infty$ and the multiplicative unit is 1.

None of these algebras can be expanded to Kleene algebras. In the max-plus algebra, for instance, the star should be an upper bound of $\{x^n \mid n \in \mathbb{N}\}$, but this set obviously has no upper bound in $\mathbb{R}\cup\{-\infty\}$. Such expansions work, however, for restricted carrier sets. As an example we have formalised the min-plus algebra over $\mathbb{N} \cup \{+\infty\}$. In this case, $x^* = 1$ for all x. Verifying the laws of (commutative) Kleene algebra from this definition is trivial, based on Isabelle's library for numbers, monoids and semilattices. The combination of max/min-plus algebras with matrices is particularly interesting for algorithmic developments or modelling, for instance, discrete event systems or Petri nets.

10. Other Models of Kleene Algebra

We have implemented further models of dioids and Kleene algebras. It can, for instance, easily be shown that the Booleans under conjunction and disjunction and, more generally, bounded distributive lattices—distributive lattices with a least and a greatest element—form Kleene algebras. Here, multiplication is meet and the Kleene star maps each element to the greatest element of the lattice.

More interestingly, many of the models considered so far have a richer structure than that captured by Kleene algebra. In particular, residuation can be defined uniformly on all power set models, hence these and other models form action algebras. In the monoid model, residuation can be defined explicitly as

$$X \to Z = \bigcup\{Y \mid X \cdot Y \subseteq Z\}, \qquad Z \leftarrow Y = \bigcup\{X \mid X \cdot Y \subseteq Z\}.$$

This follows from general properties of Galois connections over complete lattices. To establish action algebras, we have first shown in an instance proof that $\to$ and $\leftarrow$ are upper adjoints in the Galois connection axioms of residuated lattices. The two proofs are duals and special cases of more abstract lattice theoretic proofs. From this basis, the instantiation proof for action algebras is purely algebraic; it only links the star axioms of Kleene algebras, for which monoidal models have already been built, with those of action algebras.

Residuation for languages, relations, paths and traces has been formalised in similar ways. The verification of the residuated semilattice axioms is straightforward; proofs of the star axioms of action algebra are similar to those for monoids. Finally, we have formalised residuation for the min-plus algebra over the extended natural numbers. Since multiplication is commutative there is only one residual: the operation of truncated subtraction: $x \to y = y - x$. Showing that this induces a residuated lattice is immediate, based on Isabelle's libraries for natural numbers. The extension to action algebra is equally simple.

11. Proof Automation in Relation Algebra

Calculational reasoning in relation algebras is more difficult for humans than that in dioids or Boolean algebras, and perhaps even Kleene algebras. Proof automation is therefore typically less successful [22]. In this situation, a key to proof automation in applications is high-quality library design and modularisation. Domain specific knowledge is necessary to identify the right set of laws and prover specific knowledge to make these laws interact with built-in tactics and simplifiers.

When reading the following discussion, readers might find it helpful to consult the underlying Archive proof document [4] in parallel.

As a first measure to increase proof automation in relation algebra, we have implemented more than twenty useful laws of Boolean algebra which are missing in Isabelle's standard libraries. These include simplification rules, for instance

$$x \cdot y + x \cdot -y = x, \qquad (x + y) \cdot -x = y \cdot x, \qquad x + -x \cdot y = x + y,$$

isotonicity rules such as $x \leq y \longrightarrow w + x + z \leq w + y + z$, which are notoriously difficult to "find" for automated theorem provers, and Galois connections such as

$$x \cdot y = 0 \longleftrightarrow x \leq -y, \qquad x \cdot -y \leq z \longleftrightarrow x \leq y + z.$$

None of their proofs required any user interaction.

As a second measure, we have formalised Jónsson and Tarski's notion of *Boolean algebra with operators* [23, 24] in Isabelle. Following Maddux [9] we restrict our attention to unary functions of type $B \to B$ where B is a Boolean algebra. Of particular relevance are pairs of *conjugate* functions, that is operators f and g which satisfy

$$(f\ x) \cdot y = 0 \longleftrightarrow x \cdot (g\ y) = 0.$$

We have proved the basic facts about conjugate functions, most of them from Jónsson and Tarski's articles and Maddux's book. We have shown that conjugation is a symmetric relation and that conjugates are uniquely defined. Second, we have proved that conjugates are adjoints in a Galois connection, that is,

$$f\ x \leq y \longleftrightarrow x \leq -g\ (-y).$$

It then follows easily that f and g are additive, isotone and strict, that is,

$$f\ (x + y) = f\ x + f\ y, \qquad x \leq y \longrightarrow f\ x \leq f\ y, \qquad f\ 0 = 0$$

and likewise for g. They also satisfy the cancellation laws

$$f\ (-g\ x) \leq -x, \qquad g\ (-f\ x) \leq -x.$$

All these proofs were again fully automatic. Next, we have proved modular laws for conjugate functions. After proving the auxiliary law $f\ (x \cdot -g\ y) \cdot y = 0$ and its symmetric counterpart we have derived the *modular law*

$$f\ x \cdot y = f(x \cdot g\ y) \cdot y$$

and its symmetric counterpart with a few user interactions. The modular laws of relation algebras are instances of these abstract laws.

An obvious link between Boolean algebras with operators and relation algebras are the conjugation relations between $\lambda y.\, x;y$ and $\lambda y.\, x^{\smile};y$, as well as between $\lambda y.\, y;x$ and $\lambda y.\, y;x^{\smile}$. We have formalised this relationship in Isabelle. In total we have formalised more than fifty basic theorems of relation algebra. Only six proofs required user interaction. Among these are *Peirce's law*

$$x;y \cdot z^{\smile} = 0 \longleftrightarrow y;z \cdot x^{\smile} = 0,$$

one of Schröder's laws, which is similar, and the *Dedekind rule*

$$x;y \cdot z \leq (x \cdot z;y^{\smile});(y \cdot x^{\smile};z).$$

But all rules could be proved at the level of textbook granularity. The following proof is an example.

```
lemma dedekind: "x ; y · z ≤ (x · z ; y˘) ; (y · x˘ ; z)"
proof -
  have "x ; y · z ≤ (x · z ; y˘) ; (y · ((x · z ; y˘)˘ ; z))"
    by (metis modular_2' modular_1_var)
  also have "... ≤ (x · z ; y˘) ; (y · x˘ ; z)"
    by (metis conv_iso inf_le1 inf_mono mult_isol mult_isor order_refl)
  thus ?thesis
    by (metis calculation inf.commute order_trans)
qed
```

A key for increasing the automation of many of these proofs are the conjugation properties mentioned. In particular, they immediately imply the strictness or annihilation laws $x;0 = 0$ and $0;x = 0$ of the relational dioid retract.

From these basic laws we have implemented special concepts, such as vectors, subidentities or tests, various kinds of functions, and a domain operation, following the books of Schmidt and Ströhlein, Schmidt and Maddux. While the basic properties of vectors and domain were proved automatically and without much user interaction, some properties of tests turned out to be surprisingly resilient, in particular the law $-(x;1) \cdot 1' = -x \cdot 1'$, where x is a test. In the case of domain we verified little more than the axioms of domain semirings [25] in order to make all properties of this more abstract setting available in Isabelle.

We put particular emphasis on the development of a comprehensive library for functions. As usual we have implemented a partial function as a relation x satisfying $x^{\smile};x \leq 1'$; whereas a total relation satisfies $1' \leq x;x^{\smile}$, an injective relation satisfies $x;x^{\smile} \leq 1'$, and a surjective relation satisfies $1' \leq x^{\smile};x$. Moreover, a map or function is a total relation which is a partial function while a bijection is an injective and surjective function. We have proved most of the basic properties that can be found in Schmidt and Ströhlein's, Schmidt's and Maddux's books, and most of them automatically. For applications, we found the following properties very helpful. First, $(x \cdot z;y^{\smile}) = x;y \cdot z$, whenever y is a partial function. Second, $-(x;y) = x;-y$ whenever x is a function. Both required a moderate amount of user interaction. The following Isar proof of Theorem 4.2.2 (iii) from Schmidt and Ströhlein's book follows directly the textbook proof.

```
lemma ss_422iii: "p_fun_p y ⟹ (x · z ; y˘) ; y = x ; y · z"
proof (rule antisym)
  assume "p_fun_p y"
  show "x ; y · z ≤ (x · z ; y˘) ; y"
    by (metis dedekind eq_refl  mult_subdistl order_trans)
  have "(x · z ; y˘) ; y ≤  x ; y · (z ; (y˘ ; y))"
    by (metis mult_subdistr_var mult_assoc)
  also have "... ≤ x ; y · z ; 1'"
    by (metis `p_fun_p y` inf_absorb2 inf_le1 le_infI le_infI2 mult_subdistl p_fun_p_def)
  finally show "(x · z ; y˘) ; y ≤ x ; y · z"
    by (metis mult.right_neutral)
qed
```

12. Formalising Direct Relational Products

An important contribution of the Munich School consists in the axiomatisation of the concept of *direct product* by four simple identities in the calculus of relations. As far as we are aware, details have been published for the first time in Schmidt and Ströhlein's book.

$$\begin{tikzcd}
 & Y \arrow[dl, "f_1"'] \arrow[d, dashed, "f"'] \arrow[dr, "f_2"] & \\
X_1 & X_1 \times X_2 \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & X_2
\end{tikzcd}$$

It is obvious from the standard diagram for direct products that this requires a typed setting. The proofs in Schmidt and Ströhlein's book, however, suggest that the proofs demonstrating universality of the construction can be carried out safely without types. Our untyped formalisation in Isabelle directly follows their approach, but should be taken with a grain of salt since it could lead to false positives: theorems that we could prove may still fail in the presence of types; type safety has not been demonstrated formally within Isabelle.

The projections π_1 and π_2, as relations, satisfy the axioms

$$\pi_1^{\smile};\pi_1 = 1', \qquad \pi_2^{\smile};\pi_2 = 1', \qquad \pi_1;\pi_1^{\smile} \cdot \pi_2;\pi_2^{\smile} = 1', \qquad \pi_1^{\smile};\pi_2 = 1.$$

It follows that π_1 and π_2 are surjective functions. In Isabelle we have formalised direct products as follows, writing x and y instead of π_1 and π_2:

```
definition "direct_product_p x y ≡
  (x⌣ ; x = 1' ∧ y⌣ ; y = 1' ∧ x ; x⌣ · y ; y⌣ = 1' ∧ x⌣ ; y = 1)"
```

After some basic properties we have automatically proved two helpful technical lemmas about general functions using Theorem 4.2.2(iii).

```
lemma dp_aux1:
  assumes "p_fun_p z"
  and "total_p w"
  and "x⌣ ; z = 1"
  shows "(w ; x⌣ · y ; z⌣) ; z = y"
  ⟨proof⟩

lemma dp_aux2:
  assumes "p_fun_p z"
  and "total_p w"
  and "z⌣ ; x = 1"
  shows "(w ; x⌣ · y ; z⌣) ; z = y"
  ⟨proof⟩
```

Schmidt and Ströhlein's first main theorem about direct products states that direct products are monomorphic, that is, projections are, for given sets, characterised up to isomorphism. In fact, for two pairs of projections (w, x) and (y, z), this isomorphism is given explicitly by the following relation:

```
definition  "Φ ≡ (λw x y z. w ; y⌣ · x ; z⌣)"
```

We have verified the isomorphism properties described in Schmidt and Ströhlein's proof automatically, using our two technical lemmas.

```
lemma mono_dp_1:
  assumes "direct_product_p w x"
  and "direct_product_p y z"
  shows "Φ w x y z ; y = w"
  ⟨proof⟩

lemma mono_dp_2:
  assumes "direct_product_p w x"
  and "direct_product_p y z"
  shows "Φ w x y z ; z = x"
  ⟨proof⟩
```

We have then formalised Schmidt and Ströhlein's proof that Φ is indeed a function, using the characterisation $\Phi; -1' = -\Phi$ which has been proved equivalent to the usual definition of a function in our function library. We provide a detailed proof, which follows Schmidt and Ströhlein step by step, as an example. It shows that, with support from Isabelle's automated theorem prover integration, well developed libraries and an appropriate level of abstraction, even more advanced mathematical constructions and proofs can often be translated in a one-to-one fashion from paper and pencil proofs.

```
lemma Phi_map:
  assumes "direct_product_p w x"
  and "direct_product_p y z"
  shows "map_p (Φ w x y z)"
proof -
  have "Φ w x y z ; -(1') = Φ w x y z ; -(y ; y˘ · z ; z˘)"
    by (metis assms(2) direct_product_p_def)
  also have "... = Φ w x y z ; y ; -(y˘) + Φ w x y z ; z ; -(z˘)"
   by (metis compl_inf assms(2) ss43iii dp_map1 dp_map2 mult.assoc distrib_left)
 also have "... = w ; -(y˘) + x ; -(z˘)"
   by (metis (full_types) assms mono_dp_1 mono_dp_2)
 also have "... = -(w ; y˘) + -(x ; z˘)"
   by (metis assms(1) ss43iii dp_map1 dp_map2)
 also have "... = -(Φ w x y z)"
   by (metis compl_inf)
 finally show ?thesis
   by (metis map_prop)
qed
```

The proof that Φ is an injection is then fully automatic. Schmidt and Ströhlein write that $\Phi^{\smile}$ is a function as well "for reasons of symmetry". This symmetry is picked up by Isabelle and consequently the proof is fully automatic.

Finally we have constructed, for given functions f and g, a function F which makes the standard product diagram commute, and verified these commutation properties, as usual in category theory. This verification uses again our two technical lemmas and hence justifies their extraction. The proofs are fully automatic; projections are denoted by again by x and y.

```
definition "F ≡ (λ f x g y. f ; x˘ · g ; y˘)"

lemma f_proj:
  assumes "direct_product_p x y"
  and "map_p g"
  shows "F f x g y ; x = f"
  ⟨proof⟩
```

```
lemma g_proj:
  assumes "direct_product_p x y"
  and "map_p f"
  shows "F f x g y ; y = g"
  ⟨proof⟩
```

This completes Schmidt and Ströhlein's proof of universality of direct products. In addition we have verified universality of F more directly in the obvious category-theoretic way, assuming a second function G that makes the product diagram commute and showing, in a handful of interactions, that $G = F$.

In sum, all proofs in this section were automatic except for the verification that Φ is a map, where we have decided to follow Schmidt and Ströhlein's proof step by step. This case study demonstrates the quality of our relation algebra library and Isabelle's power in picking up the right lemmas and employing automated theorem proving technology. The fact that our development is based on untyped relations might raise objections from Munich. But, in fact, our proofs do not hide anything that Schmidt and Ströhlein themselves found worth mentioning. At least our proofs faithfully capture the case of a product $X \times X$, where types are not needed.

13. Discussion and Conclusion

We have presented examples for programming and automating mathematics in the spirit of the Munich School of relation algebra. These are drawn from a reference formalisation for a hierarchy of Kleene algebras and relation algebras in Isabelle/HOL. The theory hierarchy implemented supports a wide range of computing applications and more advanced mathematical investigations. Most of the models presented are new in Isabelle, in particular the monoid, path, matrix, formal power series, min-plus and max-plus model, and the relational model of omega algebra. Interestingly, the omega operation does not exist in the trace, path and language models presented in this paper [19]. While the construction of models usually required significant user interaction due to their inherently higher-order concepts, the degree of proof automation in calculational reasoning with Kleene and relation algebras observed was certainly encouraging.

Variants of Kleene algebras support and enrich each other. This is a main lesson learnt from our work on the Tarski-Kleene hierarchy. Kleene algebras, on the one hand, emphasise diversity by capturing different models of computational interest. By focusing on the star they describe one of the most ubiquitous and powerful operations of computing, which is absent in Tarski's axiomatisation of relation algebra. Relation algebra, on the other hand, specialises on one particular model and yields a more precise description of it. In applications such as program correctness and verification we expect that the two approaches will cooperate and complement each other in Isabelle and the tool will automatically set the scope by selecting the most useful hypotheses for proofs.

Relations are making their way from logics to mathematics and applied sciences; to quote from one of Gunther Schmidt's recent keynote speeches. Our work demonstrates how far state-of-the-art theorem proving environments such as Isabelle support this transition. On the one hand, the proof and programming power of these tools might exceed the expectations that computer scientists or mathematicians such as Gunther Schmidt had at the beginning of their careers. On the other hand, our formalisation also demonstrate some current shortcomings. Additional work is certainly needed to transform theorem provers such as Isabelle into mainstream tools for programming and automating mathematics in relation algebra, Kleene algebras and beyond. However, to anyone working in the field of relational an algebraic methods in computer science, these tools are highly recommended.

Acknowledgements. The authors are grateful to Simon Foster, Victor Gomes and Walter Guttmann for contributions to the formalisation of the Tarski-Kleene hierarchy in Isabelle. Rudolf Berghammer has provided valuable information on the history of relational tool development by the Munich School—an account of which remains to be written. Hitoshi Furusawa and Yasuo Kawahara have helped to clarify the presentation of this paper through several discussions. Last, but not least, the second author is indebted to Gunther Schmidt for his continuous support and critical encouragement.

References

[1] R. Berghammer, F. Neumann, Relview - an obdd-based computer algebra system for relations, in: V. G. Ganzha, E. W. Mayr, E. V. Vorozhtsov (Eds.), Computer Algebra in Scientific Computing, volume 3718 of *LNCS*, Springer, 2005, pp. 40–51.

[2] W. Kahl, Calculational relation-algebraic proofs in Isabelle/Isar, in: R. Berghammer, B. Möller, G. Struth (Eds.), Relational and Algebraic Methods in Computer Science, volume 3051 of *LNCS*, Springer, 2004, pp. 178–190.

[3] A. Armstrong, G. Struth, T. Weber, Kleene algebra, Archive of Formal Proofs (2013). `http://afp.sf.net/entries/Kleene_Algebra.shtml`.

[4] A. Armstrong, S. Foster, G. Struth, T. Weber, Relation algebra, Archive of Formal Proofs (2014). `http://afp.sf.net/entries/Relation_Algebra.shtml`.

[5] V. G. Pratt, Action logic and pure induction, in: J. van Eijck (Ed.), JELIA '90, volume 478 of *LNCS*, Springer, 1991, pp. 97–120.

[6] S. Foster, G. Struth, T. Weber, Automated engineering of relational and algebraic methods in Isabelle/HOL, in: H. C. M. de Swart (Ed.), RAMICS 2011, volume 6663 of *LNCS*, Springer, 2011, pp. 52–67.

[7] G. Schmidt, T. Ströhlein, Relationen und Graphen, Springer, 1987.

[8] G. Schmidt, Relational Mathematics, Cambridge University Press, 2011.

[9] R. D. Maddux, Relation Algebras, Elsevier, 2006.

[10] M. Wenzel, Isabelle/Isar— a versatile environment for human-readable formal proof documents, Ph.D. thesis, Technische Universität München, Germany, 2002.

[11] M. Gondran, M. Minoux, Graphs, Dioids and Semirings: New Models and Algorithms, Springer, 2010.

[12] F. Haftmann, M. Wenzel, Constructive type classes in Isabelle, in: T. Altenkirch, C. McBride (Eds.), TYPES 2006, volume 4502 of *LNCS*, Springer, 2007, pp. 160–174.

[13] J. C. Blanchette, L. Bulwahn, T. Nipkow, Automatic proof and disproof in Isabelle/HOL, in: C. Tinelli, V. Sofronie-Stokkermans (Eds.), FroCoS 2011, volume 6989 of *LNAI*, Springer, 2011, pp. 12–27.

[14] D. Kozen, On Kleene algebras and closed semirings, in: B. Rovan (Ed.), MFCS 1990, volume 452 of *LNCS*, Springer, 1990, pp. 26–47.

[15] A. Krauss, T. Nipkow, Proof pearl: Regular expression equivalence and relation algebra, J. Autom. Reasoning 49 (2012) 95–106.

[16] C. Kaliszyk, C. Urban, Quotients revisited for Isabelle/HOL, in: W. C. Chu, W. E. Wong, M. J. Palakal, C.-C. Hung (Eds.), SAC 2011, ACM, 2011, pp. 1639–1644.

[17] D. Kozen, A completeness theorem for Kleene algebras and the algebra of regular events, Inf. Comput. 110 (1994) 366–390.

[18] T. Braibant, D. Pous, An efficient Coq tactic for deciding Kleene algebras, in: M. Kaufmann, L. Paulson (Eds.), ITP 2010, volume 6172 of *LNCS*, Springer, 2010, pp. 163–178.

[19] P. Höfner, G. Struth, Algebraic notions of nontermination: Omega and divergence in idempotent semirings, J. Logic and Algebraic Programming 79 (2010) 794–811.

[20] J. Berstel, C. Reutenauer, Rational Series and their Languages, Springer, 1988.

[21] A. Chaieb, Formal power series, J. Automated Reasoning 47 (2011) 291–318.

[22] P. Höfner, G. Struth, On automating the calculus of relations, in: A. Armando, P. Baumgartner, G. Dowek (Eds.), IJCAR 2008, volume 5195 of *LNCS*, Springer, 2008, pp. 50–66.

[23] B. Jónsson, A. Tarski, Boolean algebras with operators, part 1, American Journal of Mathematics 73 (1951) 891–939.

[24] B. Jónsson, A. Tarski, Boolean algebras with operators, part 2, American Journal of Mathematics 74 (1952) 127–162.

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