Modal Kleene Algebra Applied to Program Correctness

Victor B. F. Gomes[1] and Georg Struth[2]

[1] Computer Laboratory, University of Cambridge
victor.gomes@cl.cam.ac.uk
[2] Department of Computer Science, University of Sheffield
g.struth@sheffield.ac.uk

Abstract. Modal Kleene algebras are relatives of dynamic logics that support program construction and verification by equational reasoning. We describe their application in implementing versatile program correctness components in interactive theorem provers such as Isabelle/HOL. Starting from a weakest precondition based component with a simple relational store model, we show how variants for Hoare logic, strongest postconditions and program refinement can be built in a principled way. Modularity of the approach is demonstrated by variants that capture program termination and recursion, memory models for programs with pointers, and program trace semantics.

1 Introduction

Modal Kleene algebras (MKA) [9] are algebraic relatives of propositional dynamic logic (PDL) [14] in the tradition of the dynamic algebras proposed by Németi and Pratt [22,26] and Hollenberg's algebra of dynamic negation [15]. A particularity of MKA is that reasoning is equational, symmetries between modalities are captured by algebraic and order-theoretic dualities, and soundness and completeness results arise from properties of morphisms between algebras.

MKA has highly compact axioms. It expands Kleene algebra by two dual operations and three simple equational axioms for each of them. While the Kleene algebra operations capture the sequential composition, nondeterministic choice and finite iteration of programs, the additional ones model those states from which a program can be executed and in which it can terminate. Despite its simplicity, MKA has the expressive power of PDL: modal box and diamond operators, i.e. predicate transformers, can be defined; propositions and assertions can be modelled. This makes MKA an interesting tool for program correctness.

Over the last decade, the mathematics of MKA has been well investigated, models relevant to computing have been constructed, extensions and variations introduced, applications from game theory to termination analysis considered, and mathematical components for interactive theorem provers implemented. Nevertheless, the obvious potential of MKA for building construction and verification tools for imperative programs remains to be explored. The main goal of this article is to bridge this gap.

Our main contribution therefore consists in MKA-based program correctness components for Isabelle/HOL [24] in which imperative programs can be verified or constructed by stepwise refinement. Yet our design method is generic applies beyond Isabelle. In a nutshell, it consists in deriving verification conditions for the control structure of programs by equational reasoning within MKA, in linking the algebra formally with denotational semantics of the store and data domain, and in adding data-level verification conditions, notably assignment laws, to the concrete semantics. Detailed contributions are as follows.

* We derive laws for calculating weakest (liberal) preconditions in MKA, most of them equational (Section 3), show how assignment statements and a weakest precondition rule for assignments can be obtained in the relational model of MKA (Section 4), and sketch how this development can be implemented as a simple verification (and dynamic logic) component in Isabelle (Section 5).
* We show how the opposition duality present in MKA yields verification components for strongest postconditions and a Floyd-style assignment rule in the style of [13] with minimal implementation effort (Section 6).
* We formally prove that MKA subsumes Kleene algebras with tests: all theorems of the latter setting hold in the former. This brings previous verification and refinement components based on Hoare logic [2] into scope (Section 7).
* We propose a new meta-equational while rule for weakest preconditions in the context of divergence Kleene algebras and formalise the relational model of these algebras in Isabelle, yielding a total correctness component (Section 8).
* We extend the MKA components for while programs to quantale-based ones for recursive procedures and provide new explicit definitions of modalities in this setting (Section 9).
* We further evidence the versatility of the approach by outlining a more abstract treatment of predicate transformer algebras in Isabelle and by demonstrating how other memory models and denotational program semantics can be integrated in modular ways (Section 10).

Many of the verification rules used in our components are well known, but have been derived in the algebra by simple equational reasoning (and automated theorem proving) in model-independent fashion. Similar store models have been used by the interactive theorem proving community, but our separation of data and control and their modular integration via a shallow embedding is new and yields components that are particularly simple, modular and reusable. Program verification can be performed directly in the concrete denotational semantics by using instances of the abstract algebraic rules and providing a minimal program syntax as a façade. For applications, grammars and semantic maps for suitable fragments of imperative programming languages should be provided, and facilities for code generation should be included. Integrating algebras and models also makes our components correct by construction: our formal soundness proofs make all axiomatic extensions consistent with Isabelle's small trustworthy core.

All results discussed in this article have been programmed in Isabelle and made accessible to readers in the Archive of Formal Proofs [12], including all verification components and a suite of verification examples.

2 Modal Kleene Algebra

Modal Kleene algebras are semirings expanded by two operations.

A *semiring* is a structure $(S, +, \cdot, 0, 1)$ such that $(S, +, 0)$ is a commutative monoid and $(S, \cdot, 1)$ a monoid. These interact via the distributivity laws $w(x + y)z = wxz + wyz$ and the annihilation laws $0x = 0 = x0$. Here and henceforth we drop the multiplication symbol. As in PDL, elements $x, y \in S$ represent programs; xy models sequential composition and $x + y$ nondeterministic choice. The programs 0 and 1 model abort and skip.

An *antidomain semiring* [9] is a semiring S expanded by an antidomain operation $a : S \to S$ that satisfies

$$(a\,x)x = 0, \qquad a\,x + a\,(a\,x) = 1, \qquad a\,(xy) + a\,(x(a\,(a\,y))) = a\,(x(a\,(a\,y))).$$

Intuitively, the antidomain $a\,x$ of program x represents those states from which x cannot be executed, whereas the domain $d\,x = (a \circ a)\,x$ of x models those states from which it can be. The three antidomain axioms give rise to a rich structure with symmetries and dualities.

Firstly, antidomain semirings are ordered. Addition is idempotent, $x + x = x$ holds for all $x \in S$, which is essential for interpreting it as choice. Thus S is a *dioid* and $(S, +)$ a semilattice with order relation $x \leq y \leftrightarrow x + y = y$. Addition and multiplication are order preserving or isotone; 0 is the lest element with respect to $\leq$. The converse of $\leq$ is the refinement order on programs.

Secondly, $(a(S), +, \cdot, a, 0, 1)$, with $a(S)$ denoting the image of the set S under a, is a boolean subalgebra of S with $+$ as join, $\cdot$ as meet, a as complementation and 0 and 1 as least and greatest elements. The set $d(S) = a(S)$ is closed under the operations because $d \circ d = d$, which implies that x is in $a(S)$ iff $d\,x = x$. (Anti)domain elements in $a(S)$, for which we henceforth write p, q, r, can therefore be used as assertions or propositions. We also write $\bar{p}$ instead of $a\,p$ and encode conditionals à la PDL as **if** p **then** x **else** $y = px + \bar{p}y$.

Thirdly, the *opposite* of a dioid is a dioid. Opposition swaps the order of multiplication and runs programs backwards. Yet the class of antidomain semirings is not closed under this duality. In fact, the (anti)domain of a reverse program is the (anti)range of the original one: it represents the set of those states in which it can(not) end. The opposite of an antidomain semiring is thus an *antirange semiring* with dual axioms $x(ar\,x) = 0$, $ar\ x + r\ x = 1$ and $ar\,(x \cdot y) + ar\,(x(r\,y)) = ar\,(xr(\,y))$, where $r = ar \circ ar$.

Fourthly, forward modal operators can be defined in antidomain semirings,

$$|x\rangle p = d\ (xp), \qquad \text{and} \qquad |x]p = a\,(x(a\,p)),$$

whereas backward modalities are definable by opposition in antirange semirings as $[x|y = ar\,(x(ar\,y))$ and $\langle x|y = r\,(xy)$. This is justified by the PDL semantics of $|x\rangle p$, which yields those states (in a Kripke frame) from which executing x may lead into states where p holds, whereas $[x]p$ describes those states from which x must lead to states satisfying p. Note that, in antidomain semirings, px and xp model the domain and range restriction of x to states satisfying p.

A *modal semiring* [9] is an antidomain semiring S that is also an antirange semiring in which $d\,(r\,x) = r\,x$ and $r\,(d\,x) = d\,x$ hold for all $x \in S$, and consequently $a(S) = ar(S)$.

In this setting, boxes and diamonds satisfy a number of dualities: the De Morgan laws $|x]p = \overline{|x\rangle\bar{p}}$, $|x\rangle yp = \overline{|x]\bar{p}}$, $[x|p = \overline{\langle x|\bar{p}}$, and $\langle x|p = \overline{[x|\bar{p}}$, the conjugations $(|x\rangle p)q = 0 \leftrightarrow p(\langle x|q) = 0$ and $(|x]p)q = 0 \leftrightarrow p([x|q) = 0$, and, most importantly for our purposes, the Galois connections

$$\langle x|p \leq q \leftrightarrow p \leq |x]q \qquad \text{and} \qquad |x\rangle p \leq q \leftrightarrow p \leq [x|q.$$

Modal semirings are therefore boolean algebras with operators $\langle\_|$, $[\_|$, $|\_\rangle$ and $|\_]$ of type $S \to (a(S) \to a(S))$ in the sense of Jónsson and Tarski [16]. These operators are otherwise known as predicate transformers.

PDL allows encoding while programs without assignments. A notion of finite iteration of programs must be added to a dioid for that purpose. A *Kleene algebra* is a dioid K expanded by an operation $^* : K \to K$ that satisfies the star unfold and induction axioms

$$1 + xx^* \leq x^* \qquad \text{and} \qquad z + xy \leq y \to x^*z \leq y$$

as well as their opposites $1 + x^*x \leq x^*$ and $z + yx \leq y \to zx^* \leq y$. An *antidomain Kleene algebra* [9] is an antidomain semiring that is also a Kleene algebra and likewise for antirange Kleene algebras. A *modal Kleene algebra* is a modal semiring that is also a Kleene algebra. As in PDL one can now define **while** p **do** $x = (px)^*\bar{p}$.

Henceforth we write AKA for the class of antidomain Kleene algebras and MKA for the class of modal Kleene algebras.

3 Laws for Weakest Preconditions

Conjugations and Galois connections give theorems for free, but many additional properties hold in AKA and MKA. A comprehensive list can be found in the Isabelle formalisation in the Archive of Formal Proofs [11].

In addition to these, the following laws are helpful for program verification.

Lemma 1. *Let $S \in$ AKA. For all $p, q \in a(S)$ and $x, y \in S$,*

1. $p \leq q \to |x]p \leq |x]q$ *and* $x \leq y \to |y]p \leq |x]p$,
2. $\bar{p} + |x]q = |px]q$,
3. $|x\bar{p}]q = |x](p + q)$,
4. $|x]p \leq |x\bar{q}](p\bar{q})$,
5. $p|$**if** p **then** x **else** $y]q = p|x]q$ *and* $\bar{p}|$**if** p **then** x **else** $y]q = \bar{p}|y]q$,
6. $|$**while** p **do** $x]q = (p + q)(\bar{p} + |x]|$**while** p **do** $x]q)$,
7. $p|$**while** p **do** $x]q = p|x]|$**while** p **do** $x]q$ *and* $\bar{p}|$**while** p **do** $x]q \leq q$.

These facts can be used for deriving verification conditions for the control structure of programs. To this end we define a while loop annotated with a loop invariant: **while** p **inv** i **do** $x =$ **while** p **do** x.

Lemma 2. *Let $S \in \mathsf{AKA}$. For all $p, q, i, t \in a(S)$, $x, y \in S$,*

1. $|xy]q = |x]|y]q$,
2. $|\textbf{if } p \textbf{ then } x \textbf{ else } y]q = (\bar{p} + |x]q)(p + |y]q) = \textbf{if } p \textbf{ then } |x]q \textbf{ else } |y]q$,
3. $pq \leq |x]p \rightarrow p \leq |\textbf{while } q \textbf{ do } x](p\bar{q})$,
4. $p \leq i \wedge i\bar{t} \leq q \wedge it \leq |x]i \rightarrow p \leq |\textbf{while } t \textbf{ inv } i \textbf{ do } x]q$.

In PDL, $|x]q$ models the weakest (liberal) precondition of program x and postcondition q. The specification statement for partial correctness—if precondition p holds before executing program x and if x terminates, then postcondition q holds upon termination—is captured by $p \leq |x]q$. The formulas in Lemma 2 thus calculate weakest preconditions recursively from the structure of while programs. Equation (1) yields a rule for sequential composition, (2) yields rules for conditionals, (3) is a quasi-equation for loops, and (4) a quasi-equation for loops with invariants. All rules except those for loops are purely equational and therefore superior to those of Hoare logic in applications.

4 Relational Program Semantics

The standard PDL semantics uses a Kripke frame (S, h). A program x is interpreted as a binary relation $h\,x$ between states in S and a proposition p as a subset $h\,p$ of states in S. Diamond and box formulas are interpreted as $h\,|x\rangle p = \{s \mid \exists s'.\ (s, s') \in h\,x \wedge s' \in h\,p\}$ and $h\,|x]p = \{s \mid \forall s'.\ (s, s') \in h\,x \rightarrow s' \in h\,p\}$.

With MKA it is more convenient to interpret programs and assertions uniformly as binary relations over S by embedding subsets A of S into *subidentity relations* $\{(a, a) \mid a \in A\}$.

Proposition 1 ([9,11]). *Let $(2^{S\times S}, \cup, ;, a, \emptyset, Id, ^*)$ be the set of all binary relations over the set S with the following operations: set union $\cup$, relational composition $R;S = \{(s, s') \mid \exists s''.\ (s, s'') \in R \wedge (s'', s') \in S\}$, relational antidomain $a\,R = \{(s, s) \mid \neg\exists s'.\ (s, s') \in R\}$, the identity relation $Id = \{(s, s) \mid s \in S\}$, the empty relation $\emptyset$, and the reflexive-transitive closure $R^* = \bigcup_{i\in\mathbb{N}} R^i$ of R.*

1. *This structure forms an* AKA*, the* full relation AKA *over S.*
2. *Each of its subalgebras forms a* relation AKA*.*

This soundness result justifies our previous programming intuitions. The algebraic structure of AKA is reflected at the level of relations; the algebra of subidentities $(a(2^{S\times S}), \cup, ;, a, \emptyset, Id)$ is again a boolean subalgebra. The antidomain operation can be written as $a\,R = Id \cap -(R\top)$, where $-R$ denotes the complement of R in $2^{S\times S}$, whereas a is complementation on $a(2^{S\times S})$. The universal relation $\top$ is defined as $\{(s, s') \mid s, s' \in S\}$. The relational domain operation is defined accordingly as $d\,R = \{(s, s) \mid \exists s'.\ (s, s') \in R\}$. The boolean algebra of subidentities in $a(2^{S\times S})$, the boolean algebra of subsets of S and the boolean algebra of predicates as boolean-valued functions of type $S \rightarrow \mathbb{B}$ are of course isomorphic. More precisely, the coercion functions $\lceil P \rceil = \{(s, s) \mid Ps\}$ from predicates to relations and $\lfloor R \rfloor = \{a \mid \exists b.\ (a, b) \in R\}$ from subidentity relations to predicates

form a bijective pair that preserves joins/unions, meets/intersections and negations/complements. The Kripke semantics of relational boxes and diamonds is consistent with the algebraic one: $|R\rangle P = d\,(R; P)$ and $|R]P = a\,(R; a\,P)$.

The dual of Proposition 1 links, accordingly, relations with antirange operation $ar\,R = \{(s', s') \mid \neg\exists s.(s, s') \in R\}$ and antirange Kleene algebras. In combination, these results show that MKA has relational models.

The standard relational semantics of while programs—and that of first-order dynamic logic—considers the store as a function from variables in V to values in a set E, that is, $s : V \to E$ and S is the function space E^V.

Let the update $f[b/a]\,x$ of function $f : A \to B$ in argument $a \in A$ by value $b \in B$ be defined as b whenever $x = a$ and as $f\,a$ otherwise. The relational semantics of an assignment statement is then given by

$$(v := e) \;=\; \{(s, s[(e\,s)/v]) \mid s \in E^V\},$$

where $e\,s$ denotes the value of expression e in store s. This definition allows calculating weakest preconditions of assignments.

Lemma 3. *In every relation* AKA, $|v := e]\lceil Q\rceil = \lceil \lambda s.\ Q(s[(e\,s)/v])\rceil$.

On the one hand, the formulas in Lemma 2 and 3 give us all we need for verifying while programs. On the other hand, they yield a hybrid encoding of first-order dynamic logic, where the propositional part is captured algebraically and the first-order part modelled within the relational semantics. The following section transforms this approach into a verification component.

5 Verification Component using Weakest Preconditions

The results from Section 3 and 4 suffice for implementing a simple component for dynamic logic, and primarily a verification component, quickly and easily in an interactive theorem prover; see [12] for details. Our Isabelle/HOL components use a shallow embedding; verification is performed on the concrete relational store semantics from Section 4. Relational instances of the algebraic weakest precondition laws (Section 3) are brought into scope by formalising Proposition 1. Data types for expressions, statements and while-programs and a semantic map into the relation AKA could be added easily. Instead we merely supply some syntactic sugar for relational programs. We now sketch this implementation.

Firstly, Isabelle provides axiomatic type classes and locales for formalising modular algebraic hierarchies and their models. Our verification component is based on comprehensive mathematical components for Kleene algebras [4] and AKA [11]. AKA, for instance, could have been formalised as follows.

```
class antidomain-kleene-algebra = kleene-algebra +
  fixes ad :: 'a ⇒ 'a (ad)
  assumes as1 [simp]: ad x · x = 0
  and as2 [simp]: ad (x · y) + ad (x · ad (ad y)) = ad (x · ad (ad y))
  and as3 [simp]: ad (ad x) + ad x = 1
```

This definition expands the type class of Kleene algebras to the class of AKA by adding the antidomain operation and the three antidomain axioms. The type of the antidomain operation indicates that AKAs are polymorphic and can be instantiated—for instance to the type of polymorphic binary relations or that of binary relations over a polymorphic store. Notions such as domain, boxes or diamonds can be defined within this class. Isabelle's simplifiers and integrated theorem provers can be used for proving facts about AKA, and, by the expansion, all facts proved about Kleene algebras are in scope as well.

Secondly, our Isabelle components provide soundness proofs for various models. That of relation AKA, for instance, can be formalised as follows.

```
interpretation rel-aka: antidomain-kleene-algebra
  Id {} op ∪  op ; op ⊆ op ⊂ rtrancl rel-ad
```

The interpretation statement says that any relational structure specified as in Proposition 1 forms an AKA. Isabelle dictates its proof obligations. After discharging them, instances of all abstract properties proved about AKA are available in the concrete relational semantics; in particular those from Lemma 2. In addition, the soundness proof makes the axiomatic extension of AKA consistent with Isabelle's small trustworthy core. Our verification components thus become correct by construction relative to it.

The relations introduced in the soundness proof are again polymorphic. They can be instantiated further to relations over a polymorphic store, which is defined as **type-synonym** *'a store = string ⇒ 'a*. It can model data of arbitrary and heterogenous type. The assignment command and the corresponding weakest precondition rule can then be implemented as follows.

```
definition gets :: string ⇒ ('a store ⇒ 'a) ⇒ 'a store rel (- ::= - [70, 65] 61) where
  v ::= e = {(s,s (v := e s)) |s. True}

lemma wp-assign [simp]: wp (v ::= e) ⌈Q⌉ = ⌈λs. Q (s (v := e s))⌉
```

Here, ::= is syntax for assignments, whereas := denotes Isabelle's built-in function update. After programming additional syntactic sugar for program specifications and syntax, one can start verifying while programs.

```
lemma euclid:
 PRE (λs::nat store. s ''x'' = x ∧ s ''y'' = y)
  (WHILE (λs. s ''y'' ≠ 0) INV (λs. gcd (s ''x'') (s ''y'') = gcd x y)
   DO
    (''z'' ::= (λs. s ''y''));
    (''y'' ::= (λs. s ''x'' mod s ''y''));
    (''x'' ::= (λs. s ''z''))
   OD)
  POST (λs. s ''x'' = gcd x y)
  by (rule rel-antidomain-kleene-algebra.fbox-while)  (auto simp: gcd-non-0-nat)
```

In this simple example proof, a relational instance of the while rule from Lemma 2(4) is applied first. All proof obligations generated are then simplified. As all rules except the one for loops are pure equations, they can be added to Isabelle's simplifier. Here, in particular, the sequential composition rule from Lemma 2(1) and the assignment rule from Lemma 3 eliminate the entire control structure. Automated theorem proving can then finish off the remaining data-level proof. For straight-line programs, verification proofs are purely equational and the entire control structure of programs can usually be simplified away.

Several variables of heterogenous type are handled by instantiating the type $'a$ of the store by a sum type. Verifying a typical sorting algorithm, for instance, requires type $nat + 'a + 'a\ list$ with the natural number measuring the length of the input list and $'a$ being a linearly ordered type. Assignments of variables of a summand type can then be expressed by using projections and injections. Our Isabelle theories contain an example verification of insertion sort [12].

Verification condition generation can be automated further with tactics that apply the while rule recursively. These can be programmed elegantly in Isabelle's Eisbach proof method language [18]. Verifying simple programs thus reduces to calling an Eisbach method to eliminate the control structure and then using Isabelle's provers and simplifiers for the data level; see our online examples [12].

The Isabelle formalisation provides a template for developing external verification tools. This makes it desirable to generate proof obligations as far as possible within first-order logic, so that they can be tackled by automated theorem provers and SMT solvers. It is easy to tune verification condition generation accordingly. The weakest precondition operator can, for instance, be presented as $\lfloor wp\, X\, \lceil Q \rceil \rfloor = \lambda s.\ \forall s'.\ (s, s') \in X \rightarrow Q\, s'$, the loop rule allows deriving $\forall s.\ P\, s \rightarrow \lfloor wp\, (\mathsf{WHILE}\ T\ \mathsf{INV}\ I\ \mathsf{DO}\ X\ \mathsf{OD})\, \lceil Q \rceil \rfloor\, s$ from the assumptions $\forall s. P\, s \rightarrow I\, s$, $I\, s \wedge \neg\, T\, s \rightarrow Q\, s$ and $I\, s \wedge T\, s \rightarrow \lfloor wp\, X\, \lceil I \rceil \rfloor\, s$), and the assignment rule can be written as $\lfloor wp\, (v ::= e)\, \lceil Q \rceil \rfloor = \lambda s.\ Q\, (s[(e\, s)/v])$.

6 Verification Component using Strongest Postconditions

In an influential article, Gordon and Collavizza [13] contrast the backwards approach that uses weakest preconditions and Hoare's assignment law with the lesser known forward one with strongest postconditions and Floyd's assignment law. With MKA, the two approaches are related by opposition duality, the Galois connection between forward boxes and backward diamonds. As indicated in Section 2, the specification statement $p \leq |x]q$ is equivalent to $\langle x|p \leq q$, with $\langle x|p$ capturing the strongest postcondition of program x and precondition p.

We have formalised opposition duality between antidomain and antirange Kleene algebras in Isabelle and implemented MKA based on that duality [11]. As a consequence, all facts for forward modalities can be dualised by Isabelle rather effortlessly. Facts from Lemma 1 dualise, for instance, to $p\langle x|q = \langle xp|q$, $\langle x|(pt) = \langle tx|p$, $\langle x|p \leq \langle \bar{q}x|(p\bar{q})$. More importantly, we immediately obtain the following dual statements to those in Lemma 2.

Lemma 4. *Let $S \in \mathsf{MKA}$. For all $p, q, i, t \in a(S)$, $x, y \in S$,*

1. $\langle xy|p = \langle y|\langle x|p$,
2. **if** p **then** x **else** $y|q = \langle x|(pq) + \langle y|(\bar{p}q)$,
3. $\langle x|(pq) \leq p \rightarrow \langle$**while** q **do** $x|p \leq (p\bar{q})$,
4. $p \leq i \wedge i\bar{t} \leq q \wedge \langle x|(t\bar{i}) \rightarrow \langle$**while** t **inv** i **do** $x|p \leq q$.

The lemmas listed in the following proof show how Isabelle picks up duality.

lemma *bdia-seq-var*: $\langle x|\ p \leq p' \Longrightarrow \langle y|\ p' \leq q \Longrightarrow \langle x \cdot y|\ p \leq q$
by (*metis ardual.ds.fd-subdist-1 ardual.ds.fdia-mult dual-order.trans* ...)

A forward assignment law is derivable in the relational store model.

lemma *bdia-assign* [*simp*]:
rel-aka.bdia $(v ::= e)\ \lceil P \rceil = \lceil \lambda s.\ \exists w.\ s\ v = e\ (s(v := w)) \wedge P\ (s(v{:=}w)) \rceil$

Here, *rel-aka.bdia* denotes the backward diamond operator of relation MKA. Once more, the rules in Lemma 4 and our variant of Floyd's assignment axiom suffice for program verification; the algebraic and the relational layer are linked seamlessly by the relational soundness proof for MKA. We found little difference in performance between the backward and the forward approach on simple examples. Beyond that, the forward approach offers potential for symbolic execution and static analysis [13].

7 Components for Hoare Logic and Refinement

We have previously developed program correctness components using Kleene algebras with tests (KAT) [2,1]: a verification component based on Hoare logic and a refinement component based on Morgan's specification statement [21].

Inspired by MKA we have implemented KAT as a Kleene algebra K expanded by an *antitest operation* $n : K \rightarrow K$ that satisfies

$$t\,1 = 1, \quad t\,((t\,x)(t\,y)) = (t\,x)(t\,y), \quad (n\,x)(t\,x) = 0, \quad (n\,x)(n\,y) = n\,(t\,x + t\,y),$$

where $t = n \circ n$ is the *test operation*. Similarly to MKA, $n(K) = t(K)$ forms a boolean subalgebra useful for modelling tests and assertions.

Propositional Hoare logic, that is, Hoare logic without assignment axioms, is subsumed by PDL [14]. Here we obtain the following correspondence.

Proposition 2 ([9]). *Every* AKA *and antirange Kleene algebra is a* KAT.

Formalising this fact in Isabelle brings the verification components for KAT into the scope of MKA. In KAT, Hoare triples are defined as

$$H\,p\,x\,q \leftrightarrow px \leq xq,$$

where x is a program and p, q are tests. In MKA, in turn, $p \leq |x]q \leftrightarrow px \leq xq \leftrightarrow \langle x|p \leq q$. Thus Hoare triples relate to the specification statements for weakest preconditions and strongest postconditions:

$$H\,p\,x\,q \leftrightarrow p \leq |x]q \leftrightarrow \langle x|p \leq q,$$

and this correspondence confirms that $|x]q$ is indeed the weakest precondition and $\langle x|q$ the strongest postcondition satisfying the Hoare triple. However, weakest preconditions or strongest postconditions cannot be expressed in KAT [28].

The standard rules for propositional Hoare logic are thus available in MKA and can once more be combined with Floyd and Hoare's assignment axioms. Hoare's axiom, for instance, is derivable because

$$\lceil \lambda s.\ P\,(s[(e\,s)/v]\rceil; (v := e) \;=\; (v := e); \lceil P \rceil.$$

KAT has been expanded to *refinement* KAT [2] by adding an operation $R : K \times K \to K$ and an axiom

$$H\,p\,x\,q \leftrightarrow x \leq R\,p\,q,$$

stating that Morgan's *specification statement* $R\,p\,q$ is the greatest program that satisfies the Hoare triple $H\,p\,x\,q$. MKA can be expanded to *refinement* MKA in the same way, but Galois connections

$$\langle x|p \leq q \leftrightarrow x \leq R\,p\,q \leftrightarrow p \leq [x|q$$

between the specification statements are now revealed. Once more, every refinement MKA is a refinement KAT, and the simple refinement component developed previously for KAT in Isabelle is automatically available for MKA.

8 A Meta-Equational while-Rule

A *divergence Kleene algebra* [8] is an AKA K expanded by a divergence operation $\nabla : K \to K$ that satisfies the unfold and coinduction axioms

$$\nabla\,x \leq |x\rangle \nabla\,x \qquad \text{and} \qquad p \leq |x\rangle p + q \to p \leq \nabla\,x + |x^*\rangle q.$$

Intuitively, ∇x models the set of those states from which program x need not terminate. We have formalised divergence Kleene algebras [11] and their relational models in Isabelle. In this setting, $\nabla\,R = \bigcup\{P.\ P \subseteq |R\rangle P\}$, and we have proved in Isabelle that $\nabla R = 0$ if and only if R is *noetherian* in the sense that there are no infinitely ascending R-chains. This is the case if and only if the converse of R is wellfounded. This condition is interesting for total program correctness because $\nabla\,x = 0$ relates to loop termination.

In [27], algebraic conditions for the existence of solutions in y of equations of the form $y = xy + z$ have been investigated in the context of Kleene algebras. It is well known that, by Arden's rule, a unique solution $y = x^*z$ exists in the

regular language models of Kleene algebra if language x does not contain the empty word. In relational models this empty word property can be replaced by a noethericity assumption. This analogy motivates a new meta-equational while rule for predicate transformers.

Lemma 5. *In every divergence Kleene algebra, if* $\nabla\, x = 0$*, then*

1. $p = |x\rangle p + q \leftrightarrow p = |x^*\rangle q$,
2. $p = q|x]p \leftrightarrow p = |x^*]q$.

The second meta-equation can be derived from the first one. It specialises to while loops and weakest preconditions as follows.

Lemma 6. *In every divergence Kleene algebra, if* $\nabla\,(tx) = 0$*, then*

1. $p = (t + q)|tx]p \leftrightarrow p = |\textbf{while } t \textbf{ do } x]q$,
2. $i = (t + q)|tx]i \leftrightarrow i = |\textbf{while } t \textbf{ inv } i \textbf{ do } x]q$.

In these rules, $\nabla\,(tx) = 0$ prevents that the body x of the while loop can be executed forever from states where test t holds. This of course expresses loop termination. Dual rules for forward reasoning with strongest postconditions follow immediately from the Galois connections. Partial correctness reasoning now no longer hides the explicit assumption of program termination, whereas total correctness requires discharging this assumption. Our relational soundness proof for divergence Kleene algebras in Isabelle links the absence of divergence formally with wellfoundedness and brings Isabelle's tools for termination analysis into scope (e.g. [19]). A deeper investigation of total program correctness in applications remains beyond the scope of this article.

A second benefit of the rules in Lemma 6 is that they can simplify verification condition generation, as equations for calculating the weakest precondition of a loop can be simplified to equivalent equations involving only the body of the loop. In our Isabelle implementation, however, we found it so far difficult to make this rule cooperate with the simplifiers. See our Isabelle theories for examples [12].

9 Domain Quantales and Components for Recursion

A limitation of MKA is that recursion cannot be expressed. This requires the more expressive setting of quantales, which subsume MKA, and in which classical fixpoint theory, and thus recursion, can be developed. Antidomain and modal operators can be axiomatised as before, but we present a class of quantales consistent with the relational semantics, in which an antidomain operation can be defined explicitly. We restrict our attention to single recursive procedures; mutual recursion could be captured as well by using polyvariadic fixpoint combinators.

Formally a *quantale* (or *standard Kleene algebra* [7]) is a structure $(Q, \cdot, 1, \leq)$ such that $(Q, \cdot, 1, \leq)$ is a monoid, $(Q, \leq)$ a complete lattice, and the infinite distributivity laws $x(\bigsqcup_{i\in I} y_i)z = \bigsqcup_{i\in I} xy_iz$ hold, where $\bigsqcup X$ denotes the supremum of a set $X \subseteq Q$. A quantale is *boolean* if the underlying complete lattice is a

complete boolean algebra. Thus the infinite distributivity laws $x \sqcap (\bigsqcup_{i\in I} y_i) = \bigsqcup_{i\in I}(x \sqcap y_i)$ and its lattice dual $x \sqcup (\bigsqcap_{i\in I} y_i) = \bigsqcap_{i\in I}(x \sqcup y_i)$ are required, where we write $\bigsqcap X$ for the infimum of $X \subseteq Q$.

Every quantale is a Kleene algebra with $x^* = \bigsqcup_{i\in\mathbb{N}} x^i$, and binary relations under the usual operations form boolean quantales. In every quantale, $x \sqcup y = \bigsqcup\{x, y\}$; the annihilation laws are special cases of distributivity and $\bigsqcup_{i\in\emptyset} x_i = \bot$.

Boolean quantales are similar to algebras of relations; only the operation of relational converse and the associated axioms are absent. The domain and antidomain of a relation can be defined explicitly in relation algebra as $d\,x = 1 \sqcap x\top$ and $a\,x = 1 \sqcap -(d\,x) = 1 \sqcap -(x\top)$. In boolean quantales this is impossible.

Lemma 7. *In some boolean quantale,* $(1 \sqcap x\top)x \neq x$ *and* $(1 \sqcap -(x\top))x \neq \bot$.

Proof. Consider the four-element boolean quantale with $Q = \{\bot, 1, \alpha, \top\}$ in which the order, infima, suprema and complements are defined by the fact that 1 and α are incomparable and multiplication is given by $\alpha\alpha = \bot$, $\alpha\top = \alpha = \top\alpha$ and $\top\top = \top$. Then $(1 \sqcap \alpha\top)\alpha = \bot \neq \alpha$ whereas $(1 \sqcap -(\alpha\top))\alpha = \alpha \neq \bot$. □

Though that does not rule out other explicit definitions, one can resort to axiomatising (anti)domain in quantales as in the case of MKA. As an alternative, we present a new explicit definition of antidomain for a class of boolean quantales that is consistent with binary relations.

Proposition 3. *Every boolean quantale S in which $(z \sqcap x\top)y = zy \sqcap x\top$ holds for all $x, y, z \in S$ is an* AKA *with $a\,x = 1 \sqcap -(x\top)$.*

In fact, the AKA axioms are already derivable in boolean monoids, i.e., boolean quantales where infinite infima and suprema need not exist. A dual result holds for antirange Kleene algebras satisfying $x(y \sqcap \top z) = xy \sqcap \top z$ and $ar\,x = 1 \sqcap \top x$. Boolean quantales satisfying both laws and $a(Q) = ar(Q)$ are thus MKAs.

We have already added test axioms, as described in Section 7, to quantales [2] and implemented a basic fixpoint calculus for quantales in Isabelle [3]. Two key ingredients are Knaster-Tarski's and Kleene's fixpoint theorems. A subsumption result similar to Proposition 2 can also be formalised in Isabelle.

Proposition 4. *Every antidomain and antirange quantale is a test quantale.*

Our previous components for test quantales are thus in the scope of domain quantales and can be combined with the rules for weakest preconditions and strongest postconditions from Sections 3-6. The following recursion rule, for example, can be derived for every isotone endofunction f over a domain quantale:

$$(\forall x \in Q.\ p = |x]q \rightarrow p = |f\ x]q) \rightarrow p = |\mu f]q.$$

Examples showing this rule at work have already been published in the setting of test quantales [2] and are not worth repeating.

Finally, many of the concepts used so far can be defined explicitly in the modal quantale setting. In particular,

$$|x]q = \bigsqcup\{p \mid px \leq xq\}, \qquad \langle x|p = \bigsqcap\{q \mid px \leq xq\},$$
$$R\,p\,q = \bigsqcup\{x \mid px \leq xq\}, \qquad \nabla\,x = \bigsqcup\{p \mid p \leq |x\rangle p\},$$

where of course $px \leq xq \leftrightarrow H\,p\,x\,q$ and $|x\rangle p = \bigsqcap\{q \mid xp \leq qx\}$ by opposition duality. Hence every antidomain quantale is a modal refinement Kleene algebra in the sense of Section 7 and a divergence Kleene algebra in the sense of Section 8.

10 Extensions and Variations

Program transformation and optimisation. By contrast to PDL and similarly to KAT, MKA allows considering programs outside of modal formulas as first class citizens. This allows, for instance, the treatment of program transformations and optimisations, for example in the context of a compiler [17,2].

Predicate transformer algebras. The algebra of predicate transformers as functions $S \to (a(S) \to a(S))$ can be studied abstractly in the setting of MKA [20]. Consider transformers $|x] = \lambda p.|x]p$ under multiplication $|x]|y] = |x] \circ |y]$ as function composition, meet as $|x] \sqcap |y] = \lambda p.(|x]p)(|y]p)$, a multiplicative unit $|1] = \lambda p.d\,p$ and an additive unit $|0] = \lambda p.1$. Transformers over (left) antidomain Kleene algebras then form (left) Kleene algebras with meet as addition and $|x]^* = |x^*]$ [20]. Some laws from Lemma 2 can now be represented in point-free style. Equation (1), for instance, becomes $|xy] = |x] \circ |y]$; equation (2) becomes $|\textbf{if } p \textbf{ then } x \textbf{ else } y] = |d\,p] \circ |x] \sqcap |a\,p] \circ |y]$. The Kleene algebra structure simplifies proofs at this level, but unfortunately it seems impossible to implement the lifting result in Isabelle, though all laws needed for it can be derived.

Changing the memory model. The simple store from Section 5 can be replaced modularly, for instance, by one of type *string ⇒ ('a ref + ('a ⇒ 'a ref))*, where *'a ref* is a polymorphic reference type for pointers and heaps provided by Nipkow [23]. Our approach is modular with respect to this new memory model for verifying pointer algorithms in the predicate transformer setting while using Nipkow's lemmas, for example for linked lists (see our Isabelle theories [12]), at the data level. A store in which variables of heterogeneous type are modelled by Isabelle records, which is another standard implementation (e.g. [2]), could also be added with little effort.

Changing the program semantics. One can also replace the relational program semantics modularly by other ones. As an example we have implemented a path semantics that considers non-empty finite paths $(s_1, \ldots, s_n)$ of program stores generated by the actions of a program [5,4]. Paths $(s_1, \ldots, s_m)$ and $(t_1, \ldots, t_n)$ are composed by a fusion product that yields $(s_1, \ldots, s_m, t_2, \ldots t_n)$ if $s_m = t_1$ and is undefined otherwise. Sets of non-empty paths under the product $XY = \{p \mid \exists p' \in X, p'' \in Y.\ p = p'p''\}$, set union, the empty set, the set of all paths of length one and $X^* = \bigcup_{i\in\mathbb{N}} R^i$ form Kleene algebras [4], in fact AKAs with $a\,X = \{s \mid \neg(\exists p \in X.\, s = \mathsf{first}\,p)\}$. Then $(v := e) = \{(s, s[(e\,s)/v))] \mid s \in E^V\}$ models (the paths semantics of) assignments, predicates are lifted to paths by $\lceil P\rceil = \{s \mid P\,s\}$ and a path assignment rule $|(x := e)]\lceil Q\rceil = \lceil \lambda s.\ Q(s[(e\,s)/v])\rceil$ can be derived in the path model. It can be combined with the path instances of our abstract algebraic rules for verifying programs in the path semantics.

11 Conclusion

Our program correctness components [12] are small. Relative to the previous mathematical components for MKA [11], the weakest precondition component outlined in Section 5 required proving about 30 facts, most of them by automated theorem proving within the algebra. Based on this, the strongest postconditions component needed about 10 facts, mainly to make the dual verification rules explicit. The verification and refinement components for KAT were brought into scope (Section 7) by proving three routine facts in the algebra. The development of the meta-equational while rule and the resulting total correctness component required again about 30 facts. Integrating the memory model for pointers and the path model needed once more just a handfull of proofs. Only the proofs linking divergence Kleene algebras with noethericity in the relational model required some more tedious background work.

The components for (modal) Kleene algebras in Isabelle [4,11] include axiomatic variants that can be used, for instance, for comparing programs and processes up to simulation and bisimulation equivalence and that are suitable, for instance, for analysing probabilistic programs. They also contain further models of computational interest. We expect that verification components for many of them can be developed as minor variations to the ones presented. To make such developments easy, we have usually expanded proofs in our Isabelle components to make them readable, easy to compile and robust to change.

In verification proofs we have obtained a high level of automation that compares with similar tools. Domain-specific data level proof support or techniques for inferring invariants seem crucial for enhancing automation further.

A modular extension of MKA to separation logic will be the subject of a successor article (cf. [10]); the integration of more advanced predicate transformer models with angelic and demonic nondeterminism, as described in Back and von Wright's book [6], seems equally possible. Finally, dynamic logic forms the basis of Platzer's approach to verifying hybrid and cyber-physical systems [25]. Developing MKA-based Isabelle components for it would require implementing a substantial amount of continuous mathematics in Isabelle, but might still be modular with respect to an algebraic control flow layer.

Acknowledgements. This work was partly supported by EPSRC Programme Grant *REMS: Rigorous Engineering for Mainstream Systems*, EP/K008528/1.

References

1. A. Armstrong, V. B. F. Gomes, and G. Struth. Kleene algebra with tests and demonic refinement algebras. *Archive of Formal Proofs*, 2014.
2. A. Armstrong, V. B. F. Gomes, and G. Struth. Building program construction and verification tools from algebraic principles. *Formal Aspects of Computing*, 28(2):265–293, 2016.
3. A. Armstrong and G. Struth. Automated reasoning in higher-order regular algebra. In W. Kahl and T. G. Griffin, editors, *RAMiCS 2012*, volume 7560 of *LNCS*, pages 66–81. Springer, 2012.

4. A. Armstrong, G. Struth, and T. Weber. Kleene algebra. *Archive of Formal Proofs*, 2013.
5. A. Armstrong, G. Struth, and T. Weber. Programming and automating mathematics in the Tarski-Kleene hierarchy. *J. Logical and Algebraic Methods in Programming*, 83(2):87–102, 2014.
6. R. Back and J. von Wright. *Refinement Calculus - A Systematic Introduction*. Springer, 1998.
7. J. H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, 1971.
8. J. Desharnais, B. Möller, and G. Struth. Algebraic notions of termination. *Logical Methods in Computer Science*, 7(1), 2011.
9. J. Desharnais and G. Struth. Internal axioms for domain semirings. *Science of Computer Programming*, 76(3):181–203, 2011.
10. V. B. F. Gomes. *Algebraic Principles for Program Correctness Tools in Isabelle/HOL*. PhD thesis, University of Sheffield, 2015.
11. V. B. F. Gomes, W. Guttman, P. Höfner, G. Struth, and T. Weber. Kleene algebra with domain. *Archive of Formal Proofs*, 2016.
12. V. B. F. Gomes and G. Struth. Program construction and verification components based on Kleene algebra. *Archive of Formal Proofs*, 2016.
13. M. Gordon and H. Collavizza. Forward with Hoare. In A. W. Roscoe, C. B. Jones, and K. W. Wood, editors, *Reflections on the Work of C.A.R. Hoare*, pages 101–121. Springer, 2010.
14. D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.
15. M. Hollenberg. An equational axiomatization of dynamic negation and relational composition. *Journal of Logic, Language and Information*, 6(4):381–401, 1997.
16. B. Jónsson and A. Tarski. Boolean algebras with operators, part I. *Americal Journal of Mathematics*, 73(4):207–215, 1951.
17. D. Kozen and M. Patron. Certification of compiler optimizations using Kleene algebra with tests. In J. W. Lloyd and al, editors, *CL 2000*, volume 1861 of *LNCS*, pages 568–582. Springer, 2000.
18. D. Matichuk, T. C. Murray, and M. Wenzel. Eisbach: A proof method language for Isabelle. *J. Automated Reasoning*, 56(3):261–282, 2016.
19. J. Meng, L. C. Paulson, and G. Klein. A termination checker for Isabelle Hoare logic. In *International Verification Workshop*, 2007.
20. B. Möller and G. Struth. Algebras of modal operators and partial correctness. *Theoretical Computer Science*, 351(2):221–239, 2006.
21. C. Morgan. *Programming from specifications, 2nd Edition*. Prentice Hall, 1994.
22. I. Németi. Dynamic algebras of programs. In F. Gécseg, editor, *FCT'81*, volume 117 of *LNCS*, pages 281–290. Springer, 1981.
23. T. Nipkow and G. Klein. *Concrete Semantics—With Isabelle/HOL*. Springer, 2014.
24. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
25. A. Platzer. *Logical Analysis of Hybrid Systems*. Springer, 2010.
26. V. R. Pratt. Dynamic algebras as a well-behaved fragment of relation algebras. In C. Bergman, R. D. Maddux, and D. Pigozzi, editors, *Algebraic Logic and Universal Algebra in Computer Science*, volume 425 of *LNCS*, pages 77–110. Springer, 1990.
27. G. Struth. Left omega algebras and regular equations. *J. Logic and Algebraic Programming*, 81(6):705–717, 2012.
28. G. Struth. On the expressive power of Kleene algebra with domain. *Information Processing Letters*, 116(4):284–288, 2016.