Automating Algebraic Methods in Isabelle

Walter Guttmann[1], Georg Struth[2], and Tjark Weber[3]

[1] Institut für Programmiermethodik und Compilerbau, Universität Ulm
walter.guttmann@uni-ulm.de
[2] Department of Computer Science, The University of Sheffield
g.struth@dcs.shef.ac.uk
[3] Computer Laboratory, University of Cambridge
tw333@cam.ac.uk

Abstract. We implement a large Isabelle/HOL repository of algebras for application in modelling computing systems. They subsume computational logics such as dynamic and Hoare logics and form a basis for various software development methods. Isabelle has recently been extended by automated theorem provers and SMT solvers. We use these integrated tools for automatically proving several rather intricate refinement and termination theorems. We also automate a modal correspondence result and soundness and relative completeness proofs of propositional Hoare logic. These results show, for the first time, that Isabelle's tool integration makes automated algebraic reasoning particularly simple. This is a step towards increasing the automation of formal methods.

1 Introduction

Many popular formalisms for developing and verifying programs and software systems, and many system semantics are based on algebra. Many computational logics, for instance temporal, dynamic or Hoare logics, have algebraic siblings. Algebraic approaches offer simple abstract modelling languages, system analysis via equational reasoning, and a well developed meta-theory, namely universal algebra. In the area of formal methods, algebraic semantics form an important part of, for example, Alloy, B and Z.

Among the above algebraic methods, variants of idempotent semirings and Kleene algebras play a fundamental role. They provide the operations for nondeterministic choice, sequential composition and (in)finite iteration of a system; important semantics—binary relations, computation traces, computation trees—are among their models. They have already been applied widely from compiler optimisation and feature-oriented software development to program transformation and refinement. They are particularly suitable for automation [18,20]; automated theorem proving (ATP) systems were, in fact, instrumental for developing some recent variants [10].

Stand-alone ATP systems, however, do not suffice for coherently implementing and applying algebraic methods. Mechanisms for designing modular theory hierarchies, inheriting and instantiating theorems across hierarchies and models, exploiting dualities, filtering relevant hypotheses, supporting (co)inductive

reasoning or decision procedures for data types, and integrating domain specific solvers are indispensable for these tasks. Yet all these mechanisms are available through the recent integration of ATP systems and Satisfiability Modulo Theories (SMT) solvers into Isabelle/HOL [28,4]. Our paper shows that this offers new perspectives for algebraic methods in formal software development: we have implemented a large Isabelle/HOL repository for algebraic methods, which contains more than 1000 facts.[4] They have all been obtained by ATP and SMT, using Isabelle's Sledgehammer tool which calls the external provers E, SPASS and Vampire and internally reconstructs their output with the theorem prover Metis or the SMT solver Z3. While some basic features of the repository have been presented in a tutorial paper [13], this paper describes the more advanced implementation of modal algebras and computational logics and discusses several intricate modelling examples. Our main results are as follows:

* In the context of Kleene algebras [6,22] we show how inductive proofs involving finite iteration can be automated and give a new simple automated calculational proof of a well-known termination theorem [2].
* In the context of modal Kleene algebras [8,26] we automatically derive the axioms of propositional dynamic logic, notably Segerberg's formula, show how dualities give theorems for free, discuss how to obtain an algebraic wlp-calculus, and automate a rather complex computational modelling task.
* Based on modal Kleene algebras, we automatically relate three algebraic notions of termination, which implies a modal correspondence result for Löb's formula. We apply these notions in a simple automated proof of a generalisation of the above termination theorem [12].
* From a minimalist set of algebraic axioms we develop the calculus and semantics of propositional Hoare logic and provide simple abstract automated soundness and relative completeness proofs. We instantiate this theory to modal Kleene algebras and further to the relational wlp-semantics.
* We instantiate all abstract concepts and results to binary relations, thus making them available for relational program semantics and development.

In combination, these results yield the main contribution of this paper: Isabelle's Sledgehammer tool enabled us to design and implement a large modular repository for algebraic methods with relative ease by ATP and SMT, to extend it to implementations of various computational logics and program semantics, and to automate some rather complex modelling tasks. Such methods can complement existing Isabelle verification technology [29] by additional support for developing programs that are correct by construction.

This paper can only present some main theorems and proof sketches. A complete documentation of all formal proofs is available in our repository. This paper has been generated by Isabelle's document preparation system. The theory sources and its technical content are formally verified by Isabelle2011.

[4] The repository is available at http://www.dcs.shef.ac.uk/~georg/isa/.

2 Preliminaries

This work builds on our large repository for Kleene and relation algebras that contains most of the standard calculational theorems in these areas. We use a relational semantics, but do not need full relation algebras.

An Isabelle theory for dioids is the basis of our formalisation. It covers variants of semirings needed for process algebras, the analysis of probabilistic algorithms, program refinement, formal language theory or relational program semantics. Isabelle's class mechanism [14] is used for implementing this theory hierarchy. For the sake of simplicity we only discuss semirings and dioids. Many theorems hold already in weaker variants.

Formally, a *semiring* is a structure $(S, +, \cdot, 0, 1)$ such that $(S, +, 0)$ is a commutative monoid, $(S, \cdot, 1)$ is a monoid, the distributivity laws $x \cdot (y+z) = x \cdot y + x \cdot z$ and $(x+y) \cdot z = x \cdot z + y \cdot z$ hold and 0 is an annihilator, that is, $0 \cdot x = 0 = x \cdot 0$. A *dioid* is a semiring in which addition is idempotent, that is, $x + x = x$. By idempotency, the additive monoid forms a semilattice and the dioid is ordered by the semilattice order $x \leq y \leftrightarrow x + y = y$. The semiring operations are isotone with respect to $\leq$ and 0 is its least element. The order is instrumental for proving theorems in dioids and Kleene algebras, splitting identities into inequalities: 'Use inequalities wherever possible' [6, page 120].

Each semiring comes with a dual semiring—its *opposite*—in which the order of multiplication is swapped. This is captured in Isabelle by defining $x \odot y = y \cdot x$ and proving the following fact.

Lemma (**in** *semiring-one-zero*) *dual-semiring-one-zero*:
class.semiring-one-zero (*op* +) (*op* ⊙) *1 0*

We have shown in Isabelle that the binary relations on a set S under union, relational composition, the empty set and the identity relation form a dioid. In this model, $(a, c) \in x \cdot y$ if and only if $(a, b) \in x$ and $(b, c) \in y$ for some $b \in S$, and $1 = \{(a, a) : a \in S\}$. More abstractly, an element x represents an action of a system, $+$ a non-deterministic choice between actions and $\cdot$ the sequential composition of actions; 0 and 1 are the aborting and the ineffective actions.

For modelling iterative behaviour, dioids can be expanded to Kleene algebras. Formally, a *Kleene algebra* is a dioid augmented with a star operation that satisfies the unfold and induction axioms

Assumes *star-unfoldl*: $1 + x \cdot x^* \leq x^*$
Assumes *star-inductl*: $z + x \cdot y \leq y \rightarrow x^* \cdot z \leq y$

and their opposites (with arguments to $\cdot$ swapped). In relation Kleene algebras, x^* is the reflexive transitive closure of the relation x. More abstractly, x^* is the least (pre)fixpoint of the mapping $f(y) = 1 + x \cdot y$ and its opposite. It models finite iteration of x in the sense that, by the unfold law, either 1 is executed (which has no effect) or an x-action is performed before the iteration continues. The next section shows how infinite iteration can be axiomatised similarly.

Kleene algebras were initially conceived as algebras of regular expressions. A classical result states that Kleene algebras are complete for the equational theory

of regular expressions [22]. Hence all *regular identities* from formal languages, for instance, $1 + x \cdot x^* = x^* = x^* \cdot x^* = x^{**}$ and $(x + y)^* = x^* \cdot (y \cdot x^*)^*$ and $x \cdot (y \cdot x)^* = (x \cdot y)^* \cdot x$, hold in this setting. The theory of Kleene algebras in our repository contains more than 100 facts for different variants and, by instantiation, for models based on binary relations, languages and traces. Almost all proofs could be fully automated by invoking Sledgehammer. All identities hold in weak variants where the Isabelle decision procedure for regular expression equivalence [24] is not applicable.

3 Warm-Up: Three Proofs in Kleene Algebra

We now set the scene for later results. First, beyond purely equational reasoning and the capabilities of ATP systems, we automate two inductive proofs in Kleene algebras. Second, reasoning about infinite behaviours, we automate a coinductive proof of a well-known termination theorem.

The Kleene star is often defined as a sum of powers: $x^* = \sum_{i \geq 0} x^i$. A benefit of Kleene algebra, which is slightly weaker, is that it replaces higher-order inductive reasoning about powers and unbounded suprema by equational reasoning in first-order logic. Yet many theorems combine the star and finite sums and require both kinds of reasoning. To implement this combination we use a primitive recursive function *power* for x^i with Isabelle's built-in theory of sums and a small set of simple lemmas (see below). Our first induction example is an unfold law that frequently occurs in automata theory.

Lemma *powerstar-unfoldl*: $(\sum i{=}0..n\ .\ x^i) + x^{n+1} \cdot x^* = x^*$
Proof (*induct n*)
  case *0* **show** *?case* **by** *simp* (*smt mult-oner star-unfoldl-eq*)
  case *Suc* **thus** *?case* **by** (*simp add*: *setsum-cl-ivl-Suc*) (*smt add-assoc*
    power.simps(*2*) *distl mult-oner mult-assoc star-unfoldl-eq power-commutes*)
qed

Isabelle's Isar proof language allows users to obtain human-readable proofs. In our examples, however, the main emphasis is on proof automation beyond the granularity of textbook proofs. Hence the above proof only displays the splitting into inductive cases, which is beyond first-order reasoning. The proofs of the base case and the induction step are fully automatic, using previously verified lemmas that have been selected by Sledgehammer's relevance filter.

In the base case, Isabelle's simplifier strips off the sums before Z3 uses some basic regular identities. In the induction step, we simplify again before Z3 uses the inductive definition of powers $x^{Suc\ n} = x \cdot x^n$, the lemma $x^m \cdot x^n = x^n \cdot x^m$ and further regular identities.

One advantage of the approach is that users can, to a large extent, control the granularity of Isar proofs. In the rest of this paper, we will usually only display Isar proof skeletons, in which the list of lemmas used is omitted. In these cases, we merely indicate whether Metis or the SMT solver has been used.

Our second example is (the dual of) Conway's *powerstar axiom* for Kleene algebras [6]. Its proof requires some inductive facts about sums, but does not need induction itself.

Lemma *conway-powerstar-var*: $x^* = (\sum i{=}0..n\ .\ x^i){\cdot}(x^{n+1})^*$
Proof −
have $x^* \leq (\sum i{=}0..n\ .\ x^i){\cdot}(x^{n+1})^*$ **by** (*smt star-inductl-eq sum-power-3 distr add-assoc add-comm star-unfoldl-eq mult-onel sum-power-2 mult-assoc mult-oner*)
thus *?thesis*
by (*smt power-le-star prod-star-closure star-invol star-iso sum-power-le-star eq-iff*)
qed

The identities $x \cdot \sum_{i=0}^{n} x^i = (\sum_{i=1}^{n} x^i) + x^{n+1}$ and $\sum_{i=0}^{n} x^i = 1 + \sum_{i=1}^{n} x^i$ are used in the first step together with star induction. The second step uses the approximation law $\sum_{i=0}^{n} x^i \leq x^*$. All other properties are again regular identities. We will further need the following instance of the powerstar axiom.

Lemma *conway-powerstar-2*: $x^* = (x^2)^*{+}x{\cdot}(x^2)^*$ — by smt

The above examples show, for the first time, how combined fixpoint-based and inductive reasoning can be fully automated in Isabelle.

To reason about infinite iteration we augment Kleene algebras by an omega operation which is axiomatised as a greatest (post)fixpoint.

Assumes *omega-unfold*: $x^\omega \leq x{\cdot}x^\omega$
Assumes *omega-coinduct*: $y \leq z{+}x{\cdot}y \rightarrow y \leq x^\omega{+}x^*{\cdot}z$

Kleene algebras expanded by this operation are called *omega algebras* [5]. We have shown within Isabelle/HOL that binary relations form omega algebras and developed the basic calculus of omega algebra. The following, for instance, is a separation theorem for infinite loops.

Theorem *omega-sum-refine*: $y{\cdot}x \leq x{\cdot}(x{+}y)^* \rightarrow (x{+}y)^\omega = x^\omega{+}x^*{\cdot}y^\omega$
Proof
assume $y{\cdot}x \leq x{\cdot}(x{+}y)^*$
hence $(x{+}y)^\omega \leq x{\cdot}(x{+}y)^*{\cdot}(x{+}y)^\omega{+}y^\omega$ — by smt
thus $(x{+}y)^\omega = x^\omega{+}x^*{\cdot}y^\omega$ — by metis
qed

It states that if x *quasicommutes* over y, that is, $y \cdot x \leq x \cdot (x + y)^*$, the infinite loop $(x{+}y)^\omega$ in which x and y are executed non-deterministically can be refined to the more deterministic loops in $x^\omega{+}x^*{\cdot}y^\omega$. The initial step uses the unfold law $(x{+}y)^\omega = y^\omega{+}y^*{\cdot}x{\cdot}(x{+}y)^\omega$ and then applies the consequence $y^*{\cdot}x \leq x{\cdot}(x{+}y)^*$ of quasicommutation.

In omega algebra, termination can be expressed as the absence of infinite iteration: (iteration of) x *terminates* if and only if $x^\omega = 0$. Our previous refinement theorem then implies the following well-known separation of termination:

Corollary *bd*: $y{\cdot}x \leq x{\cdot}(x{+}y)^* \rightarrow ((x{+}y)^\omega = 0 \leftrightarrow x^\omega{+}y^\omega = 0)$
by (*smt add-comm annil no-trivial-inverse omega-sum-refine omega-sum-unfold*)

It states that termination of $x + y$ can be separated into individual termination of x and y whenever x quasicommutes over y. An informal proof chasing infinite

relational diagrams is due to Bachmair and Dershowitz [2]. In omega algebra, it arises as a simple consequence of the loop refinement law. The proof in this section is significantly simpler and more automatic than earlier proofs [18]. A generalisation of this result in modal Kleene algebra is proved in Section 6.

4 Modal Semirings and Kleene Algebras

Reasoning about computing systems often requires modelling state spaces in addition to actions. One way to achieve this is to define modal operators over dioids or Kleene algebras. For an action x of a system and a set of states p, the forward diamond operator $|x\rangle p$ models the set of all states from which executing x *may* lead into p, whereas the forward box operator $|x]p$ models the set of all states from which executing x *must* lead into that set. Backward boxes and diamonds can be defined as well: $\langle x|p$ describes the set of states one may reach from p by executing x, and $[x|p$ describes the set of all states that can only be reached from p. In relational models (Kripke frames), $|x\rangle p$ is the preimage of the set p under the relation x, that is, the *domain* of x restricted in its range to p: $|x\rangle p = d(x \cdot p)$. Similarly, $\langle x|p = r(p \cdot x)$, where r denotes the range operation.

Modal operators can therefore be obtained in dioids or Kleene algebras by axiomatising domain and range. In fact, an *antidomain* operation a can be introduced in dioids by three simple axioms [10]:

Assumes *a1*: $a(x)\cdot x = 0$
Assumes *a2*: $a(x\cdot y)+a(x\cdot a(a(y))) = a(x\cdot a(a(y)))$
Assumes *a3*: $a(a(x))+a(x) = 1$

It is the Boolean complement of the domain operation: $a(x)$ describes the set of states *not* in the domain of an action x, that is, the part of the state space where x is not enabled. In the relational model, $a(x) = \{(s,s) : \neg\exists t\,.\,(s,t) \in x\}$. Domain can then be defined as $d(x) = a(a(x))$. It models that part of the state space where the action x is enabled. Thus $d(x) = \{(s,s) : \exists t\,.\,(s,t) \in x\}$ in the relational model. The domain operation induces an appropriate state space: if S is a dioid, then the set of domain elements $d(S) = \{d(x) : x \in S\}$ forms a Boolean algebra with join $+$, meet $\cdot$ and complement a. Moreover, $p \in d(S)$ if and only if $p = d(p)$, whence domain elements can be typed by applying d or a. We use the letters p, q, ... to highlight domain elements.

A range operation is axiomatised as domain in the opposite semiring. We have formalised this duality in Isabelle. All theorems about domain semirings have been automatically dualised to range semirings.

Forward diamonds and boxes are then defined in an Isabelle class for forward modal semirings.

Assumes *fdiamond-def*: $|x\rangle y = d(x\cdot y)$
Assumes *fbox-def*: $|x]y = a(x\cdot a(y))$

We have axiomatised backward modal operators dually by using range. Kleene algebras extended by all these operations are called *modal Kleene algebras* [26].

Again, duality has been formally established by Isabelle's locale mechanism and all statements about backward modalities have been obtained directly by duality.

Boxes and diamonds are duals, too: $|x]p = a(|x\rangle a(p))$ and $[x|p = a(\langle x|a(p))$. This De Morgan duality acts on the Boolean subalgebra $d(S)$ of S. Capturing it formally within Isabelle requires axiomatisations based on carrier sets, which have a detrimental effect on proof automation. To dualise diamond statements into box statements, we use a simple trick instead: we provide the dual theorem together with a set of about 10 lemmas, including the De Morgan laws for a and d and similar 'conversion theorems'. We show an example in Section 5.

The interaction between the star, the modalities and (anti)domain elements is particularly interesting. The relational semantics of while-programs can be encoded in this setting, for example, if p then x else y as $d(p) \cdot x + a(p) \cdot y$, and while p do x as $(d(p) \cdot x)^* \cdot a(p)$. This is used in Section 7. Furthermore, modal star induction laws can be derived:

Lemma *dia-star-induct*: $d(p)+(|x\rangle d(q)) \leq d(q) \rightarrow |x^*\rangle d(p) \leq d(q)$
Lemma *box-star-induct*: $d(q) \leq d(p)\cdot|x]d(q) \rightarrow d(q) \leq |x^*]d(p)$

These link modal Kleene algebras with computational logics such as propositional dynamic logic or Hoare logic. In particular, the forward box operator abstractly represents the wlp-operator and the calculus of modal Kleene algebra encompasses the laws of partial correctness for while-programs. Finally, we have instantiated the relational model of modal Kleene algebras in Isabelle.

5 Dynamic Algebras and Segerberg's Formula

We now relate modal Kleene algebras with dynamic algebras, which are algebraic siblings of propositional dynamic logic (PDL). Our repository contains automated proofs of all PDL axioms—in algebraic form—as theorems of modal Kleene algebra. As an example we present the derivation of Segerberg's formula, the only non-trivial proof task, from the modal star induction law. To simplify presentation and proof we introduce an auxiliary function.

Definition $A\ x\ p \equiv d(x\cdot p)\cdot a(p)$

It models those states outside of p from which executing x may lead into that set. Since the domain and antidomain operations are used for complementation and for typing domain elements, type conversions that humans would leave implicit tend to pollute proofs and inhibit automation. We therefore provide helper lemmas, for instance, $|x\rangle p = d(|x\rangle p)$, $|x\rangle p = |x\rangle d(p)$, $A\ x\ p = d(A\ x\ p)$, $A\ x\ p = (|x\rangle p) \cdot a(p)$ and $a(A\ x\ a(p)) = a(p) + |x]d(p)$, to derive Segerberg's formula after splitting into inequalities.

Lemma *fsegerberg*: $|x^*\rangle d(p) = d(p)+|x^*\rangle(A\ x\ p)$
Proof –
  have $|x^*\rangle d(p) \leq d(p)+|x^*\rangle(A\ x\ p)$ — by smt, using diamond star induction
  thus *?thesis* — by smt
qed

Segerberg's formula is perhaps better known and explained in box form. We prove it by duality using the trick described above.

Lemma *fbox-segerberg*: $|x^*]d(p) = d(p)\cdot|x^*](a(p)+|x]d(p))$
by (*smt a-A a-closure a-de-morgan-var-2 antidomain-semiring-domain-def fbox-simp-2 fdia-fbox fsegerberg*)

The list of lemmas contains the diamond variant of Segerberg's formula plus some helpers, including a De Morgan law for domain and antidomain elements. This list has been obtained from a larger one by minimising with Sledgehammer.

In box form, Segerberg's formula expresses induction: its right-hand side states that the system is originally in p and it is always the case (after executing x any number of times) that p will be preserved when executing x once more. The left-hand side states that the system is always in p (after repeatedly executing x). The term $a(p) + |x]d(p)$ corresponds to the Boolean implication $d(p) \rightarrow |x]d(p)$. We introduce special notation in Isabelle,

Definition $p\xrightarrow{x}q \equiv a(p)+|x]d(q)$

expressing that if p holds in the current state, q must hold after executing the action x. We can then rewrite Segerberg's formula as $|x^*]d(p) = d(p)\cdot|x^*](p\xrightarrow{x}p)$.

Modal Kleene algebra supports automated computational modelling by equational reasoning. As an example we prove a rather intricate formula expressing a separation property for alternating transitions between sets of states p and q. Again, we first introduce some helper lemmas: (i) $d(p)\cdot p\xrightarrow{x}q = d(p)\cdot|x]q$, (ii) $p\xrightarrow{x}q\cdot|x]q\xrightarrow{y}s \leq p\xrightarrow{x\cdot y}s$, and (iii) $|x^*](d(p)\cdot|x]d(p)) = |x^*]d(p)$. Equation (i) expresses a (dynamic) form of modus ponens, (ii) a property of sequential composition, as in the wlp-calculus, and (iii) an unfold property for boxes.

Theorem *alternation*:
$d(p)\cdot|x^*]((p\xrightarrow{x}q)\cdot(q\xrightarrow{x}p)) = |(x\cdot x)^*](d(p)\cdot(q\xrightarrow{x}p))\cdot|x\cdot(x\cdot x)^*](d(q)\cdot(p\xrightarrow{x}q))$
Proof −
have $d(p)\cdot|x^*]((p\xrightarrow{x}q)\cdot(q\xrightarrow{x}p)) = d(p)\cdot|(x\cdot x)^*]((p\xrightarrow{x}q)\cdot|x](q\xrightarrow{x}p)\cdot(q\xrightarrow{x}p)\cdot|x](p\xrightarrow{x}q))$
— essentially by powerstar, distributing boxes and regular identities
also have $\ldots = d(p)\cdot|(x\cdot x)^*](p\xrightarrow{x\cdot x}p)\cdot|(x\cdot x)^*]((p\xrightarrow{x}q)\cdot|x](q\xrightarrow{x}p)\cdot(q\xrightarrow{x}p)\cdot|x](p\xrightarrow{x}q))$
— essentially by the above property (ii)
also have $\ldots = |(x\cdot x)^*]d(p)\cdot|(x\cdot x)^*]((p\xrightarrow{x}q)\cdot|x](q\xrightarrow{x}p)\cdot(q\xrightarrow{x}p)\cdot|x](p\xrightarrow{x}q))$
— by Segerberg's formula
also have $\ldots = |(x\cdot x)^*](d(p)\cdot|x\cdot x]d(p))\cdot|(x\cdot x)^*]((q\xrightarrow{x}p)\cdot|x](d(q)\cdot(p\xrightarrow{x}q)))$
— by distributing boxes, property (i) and rearranging terms
finally show *?thesis* — by property (iii) and distributing boxes
qed

By instantiating q with $a(p)$ or p, our separation theorem specialises to an exercise from Harel, Kozen and Tiuryn's book on dynamic logic [15, Exercise 5.6], namely the identity $p\cdot|x^*](p\xrightarrow{x}a(p)\cdot a(p)\xrightarrow{x}p) = |(x\cdot x)^*]p\cdot|x\cdot(x\cdot x)^*]a(p)$, and to Segerberg's formula. Both instances have again been proved automatically.

6 Termination and Löb's Formula

To generalise our termination example from Section 3 we now implement notions of termination in modal Kleene algebra. In particular, we prove a modal correspondence result, namely that Löb's formula expresses wellfoundedness on transitive Kripke frames. We express both the frame property and Löb's formula in modal Kleene algebra and then establish their equivalence.

Definition $\Omega\ x\ p \equiv d(p)\cdot a(x\cdot p)$

If p is a set, then $\Omega\ x\ p$ describes those elements in p from which no further x-transitions inside of p are possible, hence x-maximal elements. We have first proved helper lemmas such as $\Omega\ x\ p = d(p)\cdot a(|x\rangle p) = d(p)\cdot |x]a(p)$. We have also proved that the non-maximal states in p are those from which there is an x-transition into p: $a(\Omega\ x\ p) = a(p) + |x\rangle p$ and $d(p)\cdot a(\Omega\ x\ p) = d(p)\cdot |x\rangle p$. Finally, we have shown that $\Omega\ x\ p = 0$ if and only if $d(p) \leq |x\rangle p$.

Following [9] we have formalised three algebraic notions of termination in Isabelle. In set theory, a relation x on a set q is *Noetherian* if every non-empty subset of q has an x-maximal element, which means that if a subset p of q has no x-maximal elements, then it must be empty:

Definition $\mathit{Noetherian}(x) \equiv (\forall\, p\ .\ \Omega\ x\ p = 0 \rightarrow d(p) = 0)$

This is equivalent to $\forall p\,.\, d(p) \leq |x\rangle p \rightarrow d(p) = 0$. Our abstract notion of Noetherity has been formally linked with the standard relational definition within Isabelle, that is, in the relational model the two definitions are equivalent. The second way of expressing termination is as follows:

Definition $\mathit{PreLoebian}(x) \equiv (\forall\, p\ .\ d(p) \leq |x^*\rangle(\Omega\ x\ p))$

Third, if x is transitive, which implies $x = x\cdot x^*$, we can apply $|x\rangle$ to both sides of this formula and obtain Löb's formula.

Definition $\mathit{Loebian}(x) \equiv (\forall\, p\ .\ |x\rangle p \leq |x\rangle(\Omega\ x\ p))$

We now relate the three properties, formalising the approach in [9]. Noetherity can be interpreted as a frame property via the relational model, and Löb's formula as a formula of modal logic; hence we establish a modal correspondence result. The main step is to show that an element is pre-Löbian if and only if it is Noetherian.

Theorem $\mathit{Noetherian}(x) \leftrightarrow \mathit{PreLoebian}(x)$
Proof –
  have $\forall\, p\ .\ d(p)\cdot a(|x^*\rangle(\Omega\ x\ p)) \leq |x\rangle(d(p)\cdot a(|x^*\rangle(\Omega\ x\ p)))$ — mainly star unfold
  hence $\mathit{Noetherian}(x) \rightarrow \mathit{PreLoebian}(x)$ — by Noetherity
  thus *?thesis* — straightforward
qed

The remaining proofs are then straightforward.

Lemma $\mathit{Loebian}(x) \rightarrow \mathit{Noetherian}(x)$
Lemma $(\forall\, p\ .\ |x\rangle(|x\rangle p) \leq |x\rangle p) \rightarrow \mathit{PreLoebian}(x) \rightarrow \mathit{Loebian}(x)$
Theorem $(\forall\, p\ .\ |x\rangle(|x\rangle p) \leq |x\rangle p) \rightarrow (\mathit{Noetherian}(x) \leftrightarrow \mathit{Loebian}(x))$

Finally, we translate Löb's formula into its more conventional box version:

Lemma $(\forall\, p\ .\ |x\rangle p \leq |x\rangle(\Omega\ x\ p)) \leftrightarrow (\forall\, p\ .\ |x](a(|x]d(p))+d(p)) \leq |x]d(p))$

As an example for termination analysis with modal Kleene algebra, we now prove a generalisation of Bachmair and Dershowitz's theorem which, in a higher-order relational setting, is due to Doornbos, Backhouse and van der Woude [12]. We expand modal Kleene algebras by an operation ∇ of *divergence* [9], mapping each action x to the set of those states from which infinite x-transition sequences may start. Divergence is modelled as a greatest (post)fixpoint; it is the greatest set of states that is invariant with respect to 'stepping back' with x.

Assumes *nabla-closure*: $d(\nabla x) = \nabla x$
Assumes *nabla-unfold*: $\nabla x \leq |x\rangle\nabla x$
Assumes *nabla-coinduction*: $d(y) \leq |x\rangle d(y)+d(z) \rightarrow d(y) \leq \nabla x+|x^*\rangle d(z)$

We have developed a simple ∇-calculus in Isabelle which is very similar to that of the omega operation. We use the instance $d(y) \leq |x\rangle d(y) \rightarrow d(y) \leq \nabla x$ of nabla coinduction, the fact that nabla is a fixpoint $\nabla x = |x\rangle\nabla x$, subdistributivity $\nabla x \leq \nabla(x+y)$, isotonicity $x \leq y \rightarrow \nabla x \leq \nabla y$, star absorption $|x^*\rangle\nabla x = \nabla x$, and $\Omega\ x\ d(y) = 0 \rightarrow d(y) \leq \nabla x$. We have also formally linked divergence Kleene algebras with the relational model.

In divergence Kleene algebras an action x terminates if and only if $\nabla x = 0$. We have shown that this property implies that x is Noetherian (hence pre-Löbian).

Doornbos, Backhouse and van der Woude's theorem generalises quasicommutation to *lazy commutation*: $y \cdot x \leq x \cdot (x+y)^* + y$. We use that lazy commutation implies $y \cdot x^* \leq x \cdot (x+y)^* + y$.

Theorem *dbw*: $y{\cdot}x \leq x{\cdot}(x+y)^*+y \rightarrow (\nabla x+\nabla y = 0 \leftrightarrow \nabla(x+y) = 0)$
Proof (*rule*+)
  assume *lazycomm*: $y{\cdot}x \leq x{\cdot}(x+y)^*+y$ **and** *xy-wf*: $\nabla x+\nabla y = 0$
  hence $\nabla(x+y) \leq |x\rangle\nabla(x+y)+|y\rangle|x^*\rangle(\Omega\ x\ (\nabla(x+y)))$
    — by nabla unfold and because x is pre-Löbian
  hence $\nabla(x+y) \leq |x\rangle\nabla(x+y)+|x\rangle|(x+y)^*\rangle(\Omega\ x\ (\nabla(x+y)))+|y\rangle(\Omega\ x\ (\nabla(x+y)))$
    using *lazycomm* — and distributing diamonds
  hence $\nabla(x+y) \leq |x\rangle\nabla(x+y)+|y\rangle(\Omega\ x\ (\nabla(x+y)))$ — by star absorption
  with *xy-wf* **show** $\nabla(x+y) = 0$ — by Noetherity
next assume $\nabla(x+y) = 0$ **thus** $\nabla x+\nabla y = 0$ — by subdistributivity of nabla
qed

In particular, this theorem holds in the relational model, where $\nabla x = d(x^\omega)$.

7 Hoare Logic

In this section we consider propositional Hoare logic (PHL), a fragment of Hoare logic that abstracts from assignments and focuses on the control structure of while-programs. PHL is also a fragment of PDL [15] and it is subsumed by Kleene algebra with tests [23]. We give an abstract algebraic formalisation and

automatically derive soundness and relative completeness of PHL. To link Hoare-style reasoning about programs with modal Kleene algebras, we show that the latter satisfy the abstract axioms.

Soundness and relative completeness of different variants of Hoare logic are well known and have already been proved in Isabelle/HOL [27,29]. Our development abstracts from underlying structures such as state spaces and program executions. By assuming a small axiom set, it generalises previous approaches in modal Kleene algebra [26], benefits automated proving and supports models beyond relational ones. Our proofs are highly automatic using Metis and Z3. Basic algebraic properties, meaningful lemmas and whole cases in inductive proofs can be shown by single calls to these tools.

Our presentation focuses on partial correctness, but this is not an inherent limitation. We take the following steps.

1. Axiomatise tests as a subset of elements that form a Boolean algebra. Tests are needed as conditions in while-programs and as preconditions in correctness statements.
2. Axiomatise preconditions. We use a subset of axioms known from the weakest liberal precondition operator.
3. Axiomatise while-programs. We use equational axioms for the conditional, the unfold rule for the while-loop and an axiom capturing soundness of the loop rule in the Hoare calculus.
4. Derive soundness and relative completeness. We can thus axiomatise validity of Hoare triples and obtain the rules of PHL as consequences.
5. Show that modal Kleene algebras form an instance of the above theory.

We now elaborate these steps.

1. Boolean subset: As described in Section 4, the range of the antidomain operation a forms a Boolean algebra. In program semantics, its elements typically represent conditions on the state space. This motivates the following axiomatisation.

We assume a structure $(S, \cdot, a)$ such that $a(S)$, the range of a, is a Boolean algebra with meet operation $\cdot$ and complement a. Technically, this is achieved by taking an axiomatisation for Boolean algebra and replacing each variable x with $a(x)$, denoting an arbitrary element of the range of a. We use Huntington's axioms [25], which are particularly concise, and therefore yield a small set of axioms for a. Additionally, closure of $a(S)$ under $\cdot$ is asserted by the axiom $a(x) \cdot a(y) = a(a(a(x) \cdot a(y)))$; by definition it is closed under a. The constants 0 and 1, the join operation $+$ and the order $\leq$ can then be expressed in terms of $\cdot$ and a. Laws of Boolean algebra, including [25, Theorems 3, 5, 7], are restricted to the range of a and derived automatically.

It is essential to impose the Boolean algebra only on a subset of elements, because some models of programs are not closed under general complements [17].

2. Preconditions: The elements of the Boolean subset serve as tests, firstly in preconditions. Our axioms for preconditions are motivated by the properties of weakest liberal preconditions [11]. The weakest liberal precondition $\mathsf{wlp}(x, q)$ is the set of initial states from which all terminating executions of the program x

end up in a state satisfying q. While the program x may be an arbitrary element, q and $\mathsf{wlp}(x, q)$ must be tests, that is, in the range of a.

This motivates the introduction of the binary operation $x\mathbin{«}q$, our abstract version of $\mathsf{wlp}(x, q)$, with the following axioms in an Isabelle class.

Assumes *pre-closed*: $x\mathbin{«}a(q) = a(a(x\mathbin{«}a(q)))$
Assumes *pre-seq*: $x{\cdot}y\mathbin{«}a(q) = x\mathbin{«}y\mathbin{«}a(q)$
Assumes *pre-test*: $a(p){\cdot}(a(p)\mathbin{«}a(q)) = a(p){\cdot}a(q)$
Assumes *pre-distrib*: $x\mathbin{«}a(p){\cdot}a(q) = (x\mathbin{«}a(p)){\cdot}(x\mathbin{«}a(q))$

Similarly to the axiomatisation of the Boolean subset, we use $a(q)$ to denote an arbitrary element in the range of a. The axiom *pre-closed* states that the result of « is a test, effectively making « an operation which takes an element and a test and yields a test. The axioms *pre-seq* and *pre-test* capture the interaction of preconditions with sequential composition and tests, respectively. The axiom *pre-distrib* separates the conjunction of two postconditions.

3. While-Programs: A second use of tests is as conditions: the statement $\mathsf{if}\ p\ \mathsf{then}\ x\ \mathsf{else}\ y$ is obtained by the ternary operation $x{\lhd}p{\rhd}y$, where p is a test. For our derivation of PHL it suffices to characterise the two branches by the following axioms; see [16,21] for more comprehensive axiomatisations.

Assumes $a(p){\cdot}(x{\lhd}a(p){\rhd}y) = a(p){\cdot}x$
Assumes $a(a(p)){\cdot}(x{\lhd}a(p){\rhd}y) = a(a(p)){\cdot}y$

The following consequence exemplifies the interaction of the conditional with preconditions. It essentially states soundness of the PHL conditional rule.

Lemma $a(p){\cdot}a(q) \leq x\mathbin{«}a(s) \wedge a(a(p)){\cdot}a(q) \leq y\mathbin{«}a(s) \rightarrow a(q) \leq x{\lhd}a(p){\rhd}y\mathbin{«}a(s)$
— by smt

Conditions also occur in while-loops: the statement $\mathsf{while}\ p\ \mathsf{do}\ x$ is obtained by the binary operation $p{*}x$, where p is a test. The unfold property of the while-loop is captured by the following axiom.

Assumes $a(p){*}x = x{\cdot}(a(p){*}x){\lhd}a(p){\rhd}1$

While-programs can be constructed from atomic programs by the operations of sequential composition, conditional and while-loop. In PHL, atomic programs are an unspecified set; in concrete models they contain, for example, assignments.

Inductive-Set *While-program*
where $x \in \mathit{Atomic\text{-}program} \Rightarrow x \in \mathit{While\text{-}program}$
| $x \in \mathit{While\text{-}program} \wedge y \in \mathit{While\text{-}program} \Rightarrow x{\cdot}y \in \mathit{While\text{-}program}$
| $x \in \mathit{While\text{-}program} \wedge y \in \mathit{While\text{-}program} \Rightarrow x{\lhd}a(p){\rhd}y \in \mathit{While\text{-}program}$
| $x \in \mathit{While\text{-}program} \Rightarrow a(p){*}x \in \mathit{While\text{-}program}$

Isabelle expects the meta-logic implication $\Rightarrow$ in such inductive definitions; we also use it instead of $\rightarrow$ for subsequent results proved by induction.

For simplicity, we assume that all tests in the range of a can be used as conditions in while-programs. A more detailed theory prescribes how to construct tests from an unspecified set of atomic tests by Boolean operations. It is then necessary to assume that preconditions are such tests; see [7,15,1] for related questions of expressibility.

4. Hoare Calculus: The unfold rule for while-loops is sufficient for proving relative completeness of PHL. Soundness of the partial correctness while-loop rule is essentially captured by the following, additional axiom.

Assumes $a(p)\cdot a(q) \leq x \mathbin{«} a(q) \rightarrow a(q) \leq a(p) * x \mathbin{«} a(a(p))\cdot a(q)$

It can be derived in models where an explicit definition of while-loops is available. The reason for this indirect characterisation is that different models may have different semantics of while-loops.

The calculus makes correctness claims in the form of Hoare triples $p\,\{x\}\,q$. Intuitively, this triple states that all terminating executions of the program x started from a state satisfying p end up in a state satisfying q.

We capture the rules of the Hoare calculus by the following inductive predicate. Hence $p\,(\!|x|\!)\,q$ holds if the triple $p\,\{x\}\,q$ is derivable.

Inductive *derived-hoare-triple* (- (| - |) -)
where $x \in \mathit{Atomic\text{-}program} \Rightarrow x \mathbin{«} a(p) (\!|x|\!) a(p)$
| $a(p) (\!|x|\!) a(q) \wedge a(q) (\!|y|\!) a(s) \Rightarrow a(p) (\!|x\cdot y|\!) a(s)$
| $a(p)\cdot a(q) (\!|x|\!) a(s) \wedge a(a(p))\cdot a(q) (\!|y|\!) a(s) \Rightarrow a(q) (\!|x \lhd a(p) \rhd y|\!) a(s)$
| $a(p)\cdot a(q) (\!|x|\!) a(q) \Rightarrow a(q) (\!|a(p) * x|\!) a(a(p))\cdot a(q)$
| $a(p) \leq a(q) \wedge a(q) (\!|x|\!) a(s) \wedge a(s) \leq a(t) \Rightarrow a(p) (\!|x|\!) a(t)$

The calculus has one axiom for atomic programs, one rule for each program construct and the rule of consequence. It follows by induction that only while-programs appear in derivable Hoare triples.

Lemma $p (\!|x|\!) q \Rightarrow p = a(a(p)) \wedge q = a(a(q)) \wedge x \in \mathit{While\text{-}program}$
by (*induct rule*: *derived-hoare-triple.induct*) — and 5 applications of smt

Validity of the Hoare triple $p\,\{x\}\,q$ is defined by the predicate $p\,\langle x\rangle\,q$, which holds if p and q are tests, x is a while-program and the condition p is sufficient to establish the postcondition q:

Definition $p \langle x \rangle q \equiv (p = a(a(p)) \wedge q = a(a(q)) \wedge x \in \mathit{While\text{-}program} \wedge p \leq x \mathbin{«} q)$

Soundness and relative completeness are proved separately by induction. Reasoning for each case is automated by Metis or Z3.

Theorem *soundness*: $p (\!|x|\!) q \Rightarrow p \langle x \rangle q$
by (*induct rule*: *derived-hoare-triple.induct*) — and 5 applications of smt

Lemma *pre-completeness*: $x \in \mathit{While\text{-}program} \Rightarrow x \mathbin{«} a(q) (\!|x|\!) a(q)$
by (*induct arbitrary*: *q rule*: *While-program.induct*) — and 4 applications of smt

Theorem *completeness*: $p \langle x \rangle q \rightarrow p (\!|x|\!) q$ — by smt

For convenient application of the calculus, we axiomatise validity of Hoare triples without referring to while-programs, using the predicate $p\,\{\!|x|\!\}\,q$.

Assumes $a(p) \{\!|x|\!\} a(q) \leftrightarrow a(p) \leq x \mathbin{«} a(q)$

Based on this, we derive the above Hoare rules and further, auxiliary rules [1].

5. Instance for Modal Kleene Algebra: In the richer structure of modal Kleene algebra, we can explicitly define preconditions, the conditional, the while-loop

and the validity of Hoare triples. To inherit the results derived above, we establish the subclass relationship by verifying the axioms, again using Z3.

— in the context of modal Kleene algebra
Assumes $x \ll p = |x]p$
Assumes $x \lhd p \rhd y = d(p) \cdot x + a(p) \cdot y$
Assumes $p * x = (d(p) \cdot x)^* \cdot a(p)$
Assumes $p \{\!| x |\!\} q \leftrightarrow d(p) \leq |x] q$

This makes the Hoare calculus available for reasoning about programs in modal Kleene algebra. The following simple example treats a while-loop, whose body contains two sub-programs w and y switching between states p and q. They are surrounded by sub-programs v, x and z for which p and q are invariants. The result shows that p is preserved by the while-loop.

Lemma $d(p) \{\!| v |\!\} d(p) \wedge d(p) \{\!| w |\!\} d(q) \wedge d(q) \{\!| x |\!\} d(q) \wedge d(q) \{\!| y |\!\} d(p) \wedge d(p) \{\!| z |\!\} d(p)$
$\rightarrow d(p) \{\!| d(s) * v \cdot w \cdot x \cdot y \cdot z |\!\} a(s) \cdot d(p)$ — by smt

8 Discussion and Conclusion

Our results show that algebraic formal methods can easily be developed by automated reasoning within Isabelle/HOL. A surprising observation is that the SMT solver Z3 often outperformed Metis and sometimes even the external ATP systems invoked by Sledgehammer. Although not especially designed for proofs in algebra, it could frequently automate proof steps at textbook level. Related empirical evidence supporting this observation has been obtained by using a benchmark suite of seven representative Isabelle formalisations that range from fast Fourier transforms to security protocol analysis [3].

Variants of Kleene algebras and their modal extensions are particularly suitable as algebraic methods because, in general, their theories admit a high degree of automation and important logics of programs and program semantics can be developed from that basis. This also includes temporal logics [18], which we did not discuss in this text. In contrast to previous work that was based solely on ATP [19,20], Isabelle's mechanisms for higher-order reasoning, proof management and theory modularisation are an essential aspect of the formalisation. To highlight the concision and simplicity of the algebraic approach, we only presented some proofs at the algebraic level. The complete formalisation of the relational model can be found in our repository.

While most of the basic proofs in Kleene algebras, omega algebras and modal Kleene algebras could be found automatically based on proof search by Sledgehammer (sometimes after splitting identities into inequalities), most of the more complex proofs in this paper have been engineered: Sledgehammer was often only able to automate individual proof steps at the granularity of handwritten proofs. Sometimes, when the scope of hypotheses was large, it was even unable to filter out the relevant lemmas. We have then merged steps for Metis or Z3 until those failed within reasonable time limits.

Our repository contains a large coherent set of algebraic theorems that require both equational and order-based reasoning. Therefore, it lends itself ideally

for empirical investigations, tool optimisation, the design of tactics and the development of proof presentation methods. The repository can be extended for algebraic proof support for existing formal methods. This is promising for applications in formal program development and analysis. It would yield a high degree of automation and the possibility to switch seamlessly between pointwise domain-specific and abstract algebraic reasoning.

Another aspect of tool integration into Isabelle has not been discussed in this paper. Counterexample generators such as Nitpick complement the ATP systems and allow a proof and refutation game which is useful for developing and debugging formal specifications. Examples can be found across the repository.

Our main interest in a repository for algebraic methods and our main motivation for the research in this paper is to devise program development methods that complement and augment existing verification environments and tools. The combination of algebraic methods with proof technology ranging from domain-specific solvers and ATP systems to higher-order reasoning within the Isabelle theorem proving environment could make formal program development significantly simpler and more automatic. Integrating additional statements such as assignments, and data types such as numbers, arrays or lists into our abstract approach, and linking it with state-of-the-art program development methods is therefore the obvious direction for future work.

Acknowledgement. Walter Guttmann was supported by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD). Georg Struth acknowledges funding from EPSRC grant EP/G031711/1. Tjark Weber acknowledges funding from EPSRC grant EP/F067909/1.

References

1. Apt, K.R., de Boer, F.S., Olderog, E.R.: Verification of Sequential and Concurrent Programs. Springer, third edn. (2009)
2. Bachmair, L., Dershowitz, N.: Commutation, transformation, and termination. In: Siekmann, J.H. (ed.) 8th International Conference on Automated Deduction. LNCS, vol. 230, pp. 5–20. Springer (1986)
3. Blanchette, J.C., Böhme, S., Paulson, L.C.: Extending Sledgehammer with SMT solvers. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) Automated Deduction: CADE-23. LNCS, vol. 6803, pp. 116–130. Springer (2011)
4. Böhme, S., Weber, T.: Fast LCF-style proof reconstruction for Z3. In: Kaufmann, M., Paulson, L.C. (eds.) Interactive Theorem Proving. LNCS, vol. 6172, pp. 179–194. Springer (2010)
5. Cohen, E.: Separation and reduction. In: Backhouse, R., Oliveira, J.N. (eds.) Mathematics of Program Construction. LNCS, vol. 1837, pp. 45–59. Springer (2000)
6. Conway, J.H.: Regular Algebra and Finite Machines. Chapman and Hall (1971)
7. Cook, S.A.: Soundness and completeness of an axiom system for program verification. SIAM J. Comput. 7(1), 70–90 (1978)
8. Desharnais, J., Möller, B., Struth, G.: Kleene algebra with domain. ACM Transactions on Computational Logic 7(4), 798–833 (2006)

9. Desharnais, J., Möller, B., Struth, G.: Algebraic notions of termination. Logical Methods in Computer Science 7(1:1), 1–29 (2011)
10. Desharnais, J., Struth, G.: Internal axioms for domain semirings. Sci. Comput. Program. 76(3), 181–203 (2011)
11. Dijkstra, E.W.: A Discipline of Programming. Prentice Hall (1976)
12. Doornbos, H., Backhouse, R., van der Woude, J.: A calculational approach to mathematical induction. Theor. Comput. Sci. 179(1–2), 103–135 (1997)
13. Foster, S., Struth, G., Weber, T.: Automated engineering of relational and algebraic methods in Isabelle/HOL. In: de Swart, H. (ed.) RAMiCS 2011. LNCS, vol. 6663, pp. 52–67. Springer (2011)
14. Haftmann, F., Wenzel, M.: Local theory specifications in Isabelle/Isar. In: Berardi, S., Damiani, F., de'Liguoro, U. (eds.) Types for Proofs and Programs. LNCS, vol. 5497, pp. 153–168. Springer (2009)
15. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press (2000)
16. Hoare, C.A.R., Hayes, I.J., He, J., Morgan, C.C., Roscoe, A.W., Sanders, J.W., Sorensen, I.H., Spivey, J.M., Sufrin, B.A.: Laws of programming. Commun. ACM 30(8), 672–686 (1987)
17. Hoare, C.A.R., He, J.: Unifying theories of programming. Prentice Hall Europe (1998)
18. Höfner, P., Struth, G.: Automated reasoning in Kleene algebra. In: Pfenning, F. (ed.) Automated Deduction: CADE-21. LNCS, vol. 4603, pp. 279–294. Springer (2007)
19. Höfner, P., Struth, G.: On automating the calculus of relations. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS, vol. 5195, pp. 50–66. Springer (2008)
20. Höfner, P., Struth, G., Sutcliffe, G.: Automated verification of refinement laws. Annals of Mathematics and Artificial Intelligence 55(1–2), 35–62 (2009)
21. Jackson, M., Stokes, T.: Semigroups with if-then-else and halting programs. International Journal of Algebra and Computation 19(7), 937–961 (2009)
22. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. Information and Computation 110(2), 366–390 (1994)
23. Kozen, D.: On Hoare logic and Kleene algebra with tests. ACM Transactions on Computational Logic 1(1), 60–76 (2000)
24. Krauss, A., Nipkow, T.: Proof pearl: Regular expression equivalence and relation algebra. Journal of Automated Reasoning (2011), http://dx.doi.org/10.1007/s10817-011-9223-4
25. Maddux, R.D.: Relation-algebraic semantics. Theor. Comput. Sci. 160(1–2), 1–85 (1996)
26. Möller, B., Struth, G.: Algebras of modal operators and partial correctness. Theor. Comput. Sci. 351(2), 221–239 (2006)
27. Nipkow, T.: Hoare logics in Isabelle/HOL. In: Schwichtenberg, H., Steinbrüggen, R. (eds.) Proof and System-Reliability. pp. 341–367. Kluwer Academic Publishers (2002)
28. Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: Sutcliffe, G., Ternovska, E., Schulz, S. (eds.) Proceedings of the 8th International Workshop on the Implementation of Logics. pp. 3–13 (2010)
29. Schirmer, N.: Verification of Sequential Imperative Programs in Isabelle/HOL. Ph.D. thesis, TU München (2006)