# A Repository for Tarski-Kleene Algebras

Walter Guttmann
Universität Ulm, Germany
walter.guttmann@uni-ulm.de

Georg Struth
University of Sheffield, UK
g.struth@dcs.shef.ac.uk

Tjark Weber
University of Cambridge, UK
tw333@cl.cam.ac.uk

**Abstract**

We have implemented a repository of algebraic structures and theorems in the theorem proving environment Isabelle/HOL. It covers variants of Kleene algebras and relation algebras with many of their models. Most theorems have been obtained by automated theorem proving within Isabelle. Main purposes of the repository are the engineering of algebraic theories for computing systems and their application in formal program development. This paper describes the present state of the repository, illustrates its potential by a theory engineering and a program verification example, and discusses the most important directions for future work.

## 1 Introduction

Algebra has long been used for modelling and reasoning about computing systems. Examples are idempotent semirings in combinatorial optimisation, algorithm design and concurrency theory, lattices in domain theory, categories and allegories in functional programming, relations in program semantics, and fixpoint calculi in model checking.

Algebra supports abstraction by focusing on some crucial system features while disregarding others. It offers uniformity since diverse system models and semantics can often be captured by one single structure or minor variations. Its metatheory—universal algebra—allows structuring or combining algebras, decomposing systems and investigating their computational complexity. Last, but not least, algebraic reasoning is usually equational, hence ideally suited for automation.

Algebra has a long history in formal methods, too, in particular in software development, where programs or protocols are formally constructed from specifications. Back and von Wright's refinement calculus, for instance, is to a large extent algebraic. Jackson's Alloy method uses a relational modelling language. Abrial's B-Book contains long catalogues of algebraic laws for reasoning about programs. Hardly any formal method, however, relies on algebra alone. While 'point-free' algebraic techniques can, for instance, be very suitable for modelling a system's control flow, they need to be complemented by 'point-wise' logical techniques for the data flow. Similarly, abstract algebraic reasoning about a system often needs to be complemented by concrete properties of a particular model.

In program verification, data-flow reasoning often seems to dominate: with Hoare logic for while-programs, for instance, dispensing with the control structure is essentially automatic (one inference rule per program construct). Relating pre- and postconditions for the program store with respect to assignments is usually more involved. In program development or construction, the situation is different. Here, algebra can help to reduce non-determinism or preserve safety and termination conditions.

To support such applications in formal methods we have implemented a large repository for algebraic theories in Isabelle/HOL.[1] It currently contains more than 1000 lemmas and theorems. The algebras considered so far are variants of Kleene algebras [8, 17], as they arise in applications to processes, probabilistic systems, program refinement, relational program semantics and automata theory, their modal extensions [9], and reducts and expansions of Tarski's

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

relation algebras [26]. Tony Hoare has proposed the name *Tarski-Kleene algebras* for this family of structures. Isabelle/HOL [23] is a theorem proving environment based on higher-order logic. It has recently been complemented by tactics for invoking external automated theorem proving systems (ATP systems) and satisfiability modulo theories solvers (SMT solvers) [6], and by counterexample generators [5].

Hierarchies of algebras can be developed in a modular way in Isabelle by using its axiomatic classes and locales; theorems can be inherited across hierarchies. Abstract algebraic structures can be linked formally with concrete models, for instance relation algebras with binary relations. This yields a seamless transition between point-free algebraic and point-wise logical reasoning. Calculational algebraic proofs can to a large extent be automated by ATP and SMT. The user can control the level of granularity of proofs and use Isabelle's proof scripting language Isar to present statements and proofs in a readable, publishable style. Proof search in Isabelle is greatly enhanced by a relevance filter, which selects hypotheses from a large set of internal libraries.

For a tutorial overview of the repository, including some simple theory formalisation and proof examples, see [10]. Some advanced modelling examples in (modal) Kleene algebras and an abstract formalisation of Hoare logic can be found in [12]. Complementing these two articles, this paper provides a more detailed description of the current structure and content of the repository. It also shows two new examples of automated theory engineering and program verification with Isabelle. Further, it discusses the role of the repository as the backbone of prospective proof environments which can be linked with existing formal methods, and various directions for future work. The repository is open to contributions of the formal methods community.

## 2 Automated Algebraic Theory Engineering in Isabelle/HOL

Isabelle/HOL [23] is one of the most widely used interactive theorem proving systems. Since its origins as a metalogical framework, there has been a strong focus on proof automation and applications in program analysis and verification. A recent breakthrough has been the integration of external ATP systems and SMT solvers via the Sledgehammer tactic [6]. State-of-the-art provers such as Vampire, E, SPASS and Z3 are called with a number of hypotheses selected by an internal relevance filter. In contrast to alternative tools such as PVS, these provers are not used as oracles. The internally verified ATP system Metis [16] or alternative methods are used to formally reconstruct proofs within Isabelle. This is desirable, since ATP systems and SMT solvers are complex software systems that depend on sophisticated heuristics. Compared to these, Isabelle's proof engine is very simple, easy to verify, and has stood the test of time. In addition, several counterexample generators have been added to Isabelle.

With this new integration at hand, users can benefit from the best of two worlds: the expressivity and versatility of interactive theorem provers, and the computational power of ATP systems, SMT solvers and counterexample generators.

It was already known that Tarski-Kleene algebras lend themselves very well to automated theorem proving, see [15] and references therein. But an implementation of our repository within Isabelle yields additional benefits:

**Theory hierarchies:** Isabelle's axiomatic classes and locales allow us to design and implement theory hierarchies for Tarski-Kleene algebras in modular ways, building on existing libraries for orders, semilattices and Boolean algebras. For instance, we have formally captured in Isabelle that relation algebras are expansions of Boolean algebras. Models of axiomatic structures can be obtained by instantiation. For example, we have proved that binary relations and formal languages are models of Kleene algebras.

**Cross-theory reasoning:** Theorems are automatically inherited across the hierarchy. All theorems about orders, for instance, are available automatically for semilattices and Boolean algebras; all theorems about relation algebras hold in the model of binary relations; all theorems of Kleene algebras hold in relation algebras expanded by an operation of reflexive-transitive closure. One particular algebraic axiomatisation can have a variety of computationally interesting models. Theorems proved at the abstract algebraic level are automatically available in all models: for instance, our theorems about Kleene algebras hold for binary relations and formal languages. In each particular model they can be augmented by domain-specific facts that will usually be proved by means of logic and set theory. In the relational semantics of imperative programs, for instance, abstract point-free algebraic facts can be combined with concrete point-wise reasoning about the store of a program and its updates.

**Proof management:** Isabelle ensures that only verified facts can be used as hypotheses in proofs. Moreover, with the Isar scripting language, the user owns the means of production: proofs can be either fully automated or refined into steps of arbitrary granularity. The proof of an equation $s = t$ in Boolean algebra, for instance, can be broken down into proofs of $s \leq t$ and $t \leq s$. State-of-the-art ATP systems and SMT solvers are powerful enough to prove calculational algebraic statements at textbook-level granularity. In calculational applications, Isabelle becomes almost entirely a proof manager for the external ATP systems.

**Hypothesis learning:** Isabelle provides a relevance filter that searches its internal libraries and selects hypotheses for individual proof goals. For small theory scopes this is surprisingly effective. In our case studies, proofs of moderate complexity could usually be fully automated by calling Sledgehammer. Structures such as modal Kleene algebras, where large numbers of lemmas are in the scope, seem to bring the relevance filter to its limits.

**Theorems for free:** Isabelle's higher-order features support more sophisticated forms of proof management which are based on symmetries, dualities and similar properties. In semigroups with forward and backward modalities, for instance, there is a symmetry between these two kinds of operations that is expressed by swapping the order of multiplication. In Boolean algebra, there is a duality between the underlying join and meet semilattices. In relation algebra, theorems such as the modular laws can be obtained by instantiating more general theorems about Boolean algebras with operators. Operations that are shown to be adjoints of a Galois connection satisfy certain additivity, isotonicity or cancellation properties. All of these properties can be expressed and exploited in Isabelle, and are heavily used in engineering our repository.

Due to these features, the combination of algebra with automated proof search lends itself to the development of lightweight algebraic formal methods with heavyweight automation. Whereas previously a variety of different Isabelle tactics had to be mastered by users in order to make proofs succeed, the ATP/SMT integration largely simplifies this task to proof planning plus push-button proof search. The complementation of automated proof search with counterexample generators such as Nitpick and Quickcheck [5] allows a style of proof and refutation that is particularly beneficial for engineering new theories and debugging specifications.

## 3 Implementing Tarski-Kleene Algebras in Isabelle

'Tarski-Kleene algebras' loosely characterise a family of algebras based around Kleene and relation algebras. Kleene algebras were originally proposed by Kleene as algebras of regular expressions; more recently variants of Kleene algebras have been used for modelling program refinement [27] and probabilistic protocols [20]. Extensions cover infinite systems [7], modal reasoning similar to propositional dynamic logic [9], Hoare logic [21] and true concurrency [14].

Relation algebras have initially been conceived by Tarski as algebras of binary relations [26]. There is a longstanding history of using such structures for program semantics [4, 19] or as a basis for data refinement [24]. In the area of formal methods, relation algebras have been used for developing algorithms for graphs, orders or lattices from logical specifications [2, 3].

Kleene algebras and relation algebras share many properties, but there are also significant differences. Kleene algebras provide precisely the regular operations of addition (or union or join), multiplication and Kleene star, which in the context of programming can be interpreted as non-deterministic choice, sequential composition and finite iteration. Relation algebras lack the star operation, but provide operations for meet, complementation and converse besides addition and multiplication. Relation algebras have been designed with one particular model in mind, whereas Kleene algebras owe their fundamental status to the fact that they capture several important models of computation.

Our hierarchy connects Kleene algebras and relation algebras by expanding the latter with an operation of reflexive-transitive closure, as proposed by Ng and Tarski [22]. Then every expanded relation algebra is a Kleene algebra and the theorems for Kleene algebras are available in this setting. Similarly, we expand relation algebras by operations of domain and range, which project on the first and second coordinate of a binary relation, and link these operations with the modalities of modal Kleene algebras. In this context, every relation algebra thus expanded is a modal Kleene algebra and all theorems are again inherited.

In the context of program development, most of the theory hierarchy should be hidden behind an interface, providing developers with a simple relational specification language à la Alloy and access to Sledgehammer and Nitpick. From that side of the interface, the distinction between reasoning in relation algebra or Kleene algebra would vanish.

We now describe the theory hierarchy in more detail.

**Dioids:** Our hierarchy is based on classes for semilattices and variants of semirings. Near semirings (a generalisation of near rings) consist of an additive and a multiplicative semigroup that interact via a single distribution law; we also require that addition is commutative. Near dioids are obtained by making addition idempotent; this gives a semilattice structure with a canonical order (the refinement order in many models). By distributivity, multiplication is isotone in one argument. Adding isotonicity in the other argument gives predioids; requiring both distribution laws yields dioids (and semirings by omitting idempotency). Further variants are obtained by including a multiplicative or an additive unit. The latter is typically a left annihilator of multiplication, but in several models it is not a right annihilator; we account for this similarly to the omission of one of the distribution laws.

**Kleene Algebras:** All of the dioid variants are expanded by axioms for the Kleene star; they too come in left- and right-sided forms. These weaker Kleene algebras are important in applications involving demonic refinement algebra [27] or probabilistic Kleene algebra [20]. Interestingly, all the known equations of Kleene algebra could already be proved in the variant which omits the right induction axiom.

**Omega Algebras:** Supplementing the Kleene star operation for finite iteration, omega algebra introduces the omega operation for infinite iterations. This is useful, for instance, to model reactive, not necessarily terminating systems. Among the applications of this theory we provide, for example, highly automated proofs of loop refinement laws and termination theorems.

**Domain Semirings:** Semirings and dioids are expanded by a domain operation, which abstractly represents the set of states in which a computation is enabled. In particular, the image of the domain operation corresponds to the state space of a program; depending on the axioms it is a distributive lattice or a Boolean algebra [9]. Domain elements can serve as tests, for example, in preconditions and conditional statements.

**Range Semirings:** The range operation is obtained from domain by swapping the order of multiplication. The entire theory is obtained fully automatically by dualising domain semirings, using Isabelle's locale mechanism.

**Modal Kleene Algebras:** Domain and range give rise to forward and backward diamond and box operators. They abstractly represent states from which a computation may or must reach certain target states; in particular the forward box corresponds to the weakest liberal precondition. With the Kleene star operation we obtain Kleene algebra with tests [18] and, for applications in formal methods, a semantics for simple while-programs, and algebraic variants of propositional Hoare logic and propositional dynamic logic. Axiomatic algebraic approaches to temporal logics such as LTL and CTL can easily be developed from that basis.

**Demonic Refinement Algebra:** The axioms in our hierarchy cover related theories, such as von Wright's demonic refinement algebra [27] and the imperative fragment of the Unifying Theories of Programming [13]. So far we only have basic theorems for these; particular models and advanced results need to be added.

**Concurrent Kleene Algebra:** The development is discussed in more detail in Section 4.

**Propositional Hoare Logic:** A more basic setting (than modal Kleene algebra) suffices to encode this logic. Based on a Boolean algebra representing the state space, we directly axiomatise preconditions and while-programs; soundness and completeness of the Hoare rules are then derived automatically. This makes the calculus available for a wider range of models.

**Boolean Algebra:** Based on Huntington's minimalist axioms we have implemented Boolean algebra. This is useful because only few axioms have to be checked for instantiation, but it also yields an interesting test bed for ATP performance due to the difficulty of deriving the usual laws. We use the higher-order features of Isabelle to provide Boolean algebras with operators.

**Relation Algebra:** Expanding Boolean algebras with operations for composition and converse yields relation algebras. In particular, they are dioids, whence we automatically inherit the dioid theorems. We have added most of the ingredients for relational program development: subidentities and vectors for modelling sets, points, a calculus of functions, and domain and range operations. We have formally shown that relation algebras are domain and range semirings. Finally, we have expanded relation algebra by an operation of reflexive-transitive closure and shown that the resulting structure is a Kleene algebra.

We have formalised the most important models of these structures, for instance, binary relations, languages and program traces for Kleene algebras, omega algebras and Kleene algebras with domain. The formal relationship between the abstract algebras and the concrete models is established by using Isabelle's locale mechanism. An example is discussed in more detail in the next section.

## 4 Engineering Concurrent Semirings

This section illustrates theory engineering in the context of concurrent semirings and their models. Concurrent semirings have been proposed—under a different name—two decades ago as algebraic axiomatisations of series-parallel posets [11]. They have recently been studied as models for true concurrency based on a simple aggregation and independence model that is inspired by concurrent separation logic [14]. Here, we sketch the implementation of the abstract theory hierarchy from semigroups to concurrent semirings, and of their set-theoretic models based on notions of aggregation and independence. Due to lack of space we cannot show the Isabelle development; the complete code can be found in our repository.

We have first implemented the following algebraic hierarchy using Isabelle's axiomatic classes. An *ordered semigroup* is a structure $(S, \cdot, \leq)$ such that $(S, \cdot)$ is a semigroup, $(S, \leq)$ is a poset and $\cdot$ is isotone with respect to the order: $x \leq y$ implies $z \cdot x \leq z \cdot y$ and $x \cdot z \leq y \cdot z$. An *ordered monoid* $(S, \cdot, 1, \leq)$ is an ordered semigroup expansion such that $(S, \cdot, 1)$ is a monoid. In our setting, $\cdot$ can be interpreted as a form of sequential or serial composition of actions in $S$.

To model true concurrency we introduce a second operation $\otimes$ of concurrent or parallel composition. In contrast to process algebras such as CCS it is not necessarily related to interleaving. An *ordered bisemigroup* is a structure $(S, \cdot, \otimes, \leq)$ such that $(S, \cdot, \leq)$ is an ordered semigroup and $(S, \otimes, \leq)$ is an ordered commutative semigroup. In particular, $\otimes$ is also isotone. An *ordered bimonoid* $(S, \cdot, \otimes, 1, e, \leq)$ is the obvious expansion, where 1 is the unit of $\cdot$ and $e$ that of $\otimes$.

Next we relate sequential and parallel composition. A *concurrent semigroup* is an ordered bisemigroup that satisfies the *multiplication inclusion law* $x \cdot y \leq x \otimes y$, the *small exchange laws* $(x \otimes y) \cdot z \leq x \otimes (y \cdot z)$ and $x \cdot (y \otimes z) \leq (x \cdot y) \otimes z$, and the *exchange law* $(w \otimes x) \cdot (y \otimes z) \leq (w \cdot y) \otimes (x \cdot z)$. A *concurrent monoid* is an ordered bimonoid that satisfies $1 = e$ and the exchange law. It can be shown that regular languages with shuffle and series-parallel posets satisfy the above laws (for example, $(\{a\} \otimes \{a\}) \cdot (\{b\} \otimes \{b\})$ contains less words than $(\{a\} \cdot \{b\}) \otimes (\{a\} \cdot \{b\})$), but our main justification comes from the model below. We have proved by ATP that the multiplication inclusion law and the small exchange laws are derivable in concurrent monoids, and, using Isabelle's counterexample generators, that none of the specific concurrent semigroup axioms hold already in ordered bisemigroups.

At the final stage of the abstract hierarchy we have implemented concurrent semirings. Formally, a *concurrent semiring* is a structure $(S, +, \cdot, \otimes, 0, 1)$ such that both $(S, +, \cdot, 0, 1)$ and $(S, +, \otimes, 0, 1)$ are idempotent semirings $(x + x = x)$, and $(S, \cdot, \otimes, 1, \leq)$ is a concurrent monoid, where $x \leq y$ if and only if $x + y = y$.

At the concrete set-theoretic level, we have implemented independence algebras over aggregation algebras. An *aggregation semigroup* is a semigroup $(A, \oplus)$ and an *aggregation monoid* a monoid $(A, \oplus, u)$. An *independence relation* is a bilinear binary relation $R$ on an aggregation algebra: $R\ (x \oplus y)\ z \Leftrightarrow R\ x\ z \wedge R\ y\ z$ and $R\ x\ (y \oplus z) \Leftrightarrow R\ x\ y \wedge R\ x\ z$. In the monoidal case, $R$ is also *bistrict*: $R\ u\ x$ and $R\ x\ u$. The idea is that $x \oplus y$ represents a system that consists of two parts $x$ and $y$; $u$ is the empty system. The linearity laws say that a compound system is independent from another system if and only if its parts are. The strictness laws say that the empty system is independent from any system. We use two independence relations $R$ and $S$ for sequential and concurrent composition and require that $S\ x\ y \Leftrightarrow S\ y\ x$ and $R \subseteq S$.

We have verified a number of properties by ATP that are useful for proving instances of the concurrent semirings and monoid axioms. The following law, for example, is used in the proof of the exchange law: $R\ (w \oplus x)\ (y \oplus z) \wedge S\ w\ x \wedge S\ y\ z \Rightarrow R\ w\ y \wedge R\ x\ z \wedge S\ (w \oplus y)\ (x \oplus z)$.

The last step for building models is to define operations on the powerset of the carrier of an aggregation algebra $A$. This is similar to lifting word to language products. We define the *complex product* $\circ_R : 2^A \times 2^A \to 2^A$ as $X \circ_R Y = \{x \oplus y \mid x \in X \wedge x \in Y \wedge R\ x\ y\}$. Since ATP systems are rather erratic on set expressions, we prove the property $z \in X \circ_R Y \Leftrightarrow \exists x, y : z = x \oplus y \wedge x \in X \wedge y \in Y \wedge R\ x\ y$ (and similarly for $\circ_S$). It can be used in combination with Isabelle's built-in laws for set extensionality and set inclusion to simplify to first-order expressions that ATP systems can handle.

Theorem proving at this level usually requires the application of Isabelle's simplifier with the mentioned rules, before calling Sledgehammer. We could then easily verify that the independence algebras under consideration form concurrent semigroups or concurrent monoids. Finally, with $+$ interpreted as set union, we verified that independence algebras form concurrent semirings.

# 5 Verification of a Naive Reachability Algorithm

As a second example, we show the verification of a naive reachability algorithm [2] using the algebraic structures and lemmas available in our repository. This example is peculiar in that relations are not only used at the control flow level, but also, and primarily, as data structures that capture the digraphs or transition systems on which reachability is considered.

The algorithm is implemented in a simple imperative language with assignment, sequential composition and while-loops:

$x := V$; **while** $\neg(x \cdot Y \leq x)$ **do** $x := x + x \cdot Y$ **od**

The algorithm operates on a single variable $x$. First, $x$ is initialised to $V$, a vector that represents initial states. $Y$ is an adjacency matrix. The elements $x$, $V$ and $Y$ can be modelled by binary relations; we represent them more abstractly as elements of a Kleene algebra. Upon termination, $x$ contains the relation $V \cdot Y^*$, that is, those states reachable from $V$ via the reflexive-transitive closure of $Y$. Partial correctness is thus expressed by the following Hoare triple.

**Theorem 1: vars** $x$
$\{\ \mathit{True}\ \}\ x := V$; **while** $\neg(x \cdot Y \leq x)$ **inv** $\{\ V \leq x \wedge x \leq V \cdot Y^*\ \}$ **do** $x := x + x \cdot Y$ **od** $\{\ x = V \cdot Y^*\ \}$

Here, we have additionally annotated the while-loop with its invariant, which captures the idea of the program: to compute intermediate relations $x$ iteratively such that after each iteration, $x$ is a superset of $V$ and a subset of $V \cdot Y^*$. To prove this theorem, we rely on built-in automation in Isabelle/HOL that uses the invariant together with Hoare rules for assignment, sequential composition and while-loops to eliminate the algorithm's control structure [25]:

**proof** (vcg_simp)

After this simplification we are left with three automatically generated verification conditions. The first states that the precondition implies the loop invariant:

**show** $V \leq V \cdot Y^*$

We invoke Sledgehammer with this subgoal. It calls 5 external ATP systems, all of which find a proof within a few seconds. They return the set of lemmas from our Kleene algebra repository used in the proof. For example, the prover E automatically generates the following command:

**by** (metis mult_isol mult_oner star_ref)

This invokes Isabelle's built-in automation for first-order logic, Metis, to reconstruct the proof using three basic lemmas. Metis immediately succeeds and the first subgoal is thus proved. The second condition states that the invariant is preserved under execution of the loop's body:

**next fix** $x$ **show** $V \leq x \wedge x \leq V \cdot Y^* \wedge \neg(x \cdot Y \leq x) \Rightarrow V \leq x + x \cdot Y \wedge x + x \cdot Y \leq V \cdot Y^*$

We invoke Sledgehammer again. This time, only Vampire finds a proof within a few minutes. It uses 9 lemmas and neither Metis nor SMT are able to reconstruct a proof within Isabelle. Instead we proceed by proving one part of the condition from a reduced set of assumptions:

**next fix** $x$ **show** $x \leq V \cdot Y^* \Rightarrow x + x \cdot Y \leq V \cdot Y^*$
**by** (metis add_lub leq_def mult_assoc mult_isol star_1r subdistr)

In a few seconds, Vampire returns with 6 lemmas, and Metis is able to reconstruct a proof. The second condition is completed by invoking Sledgehammer again. All provers and Metis succeed:

**thus** $V \leq x \wedge x \leq V \cdot Y^* \wedge \neg(x \cdot Y \leq x) \Rightarrow V \leq x + x \cdot Y \wedge x + x \cdot Y \leq V \cdot Y^*$
**by** (metis add_ub order_trans)

The final condition states that the loop invariant after termination implies the postcondition. Called from Sledgehammer, a proof is automatically produced by SPASS within a few seconds:

**next fix** $x$ **show** $V \leq x \wedge x \leq V \cdot Y^* \wedge x \cdot Y \leq x \Rightarrow x = V \cdot Y^*$
**by** (metis add_lub le_neq_trans less_le_not_le star_inductr) **qed**

This completes the proof of Theorem 1. It is fully automatic except for the second verification condition, where Isabelle's proof reconstruction does not keep up with Vampire. This issue would vanish if the external prover returned a detailed proof that could be checked in Isabelle.

We formulated the reachability algorithm in terms of Kleene algebra operations. The proof of Theorem 1 only used axioms and lemmas of Kleene algebra. In our repository, we have shown that binary relations are a model of Kleene algebras. Isabelle/HOL, therefore, automatically generates an instance of Theorem 1 where all Kleene algebra operations have been replaced by the corresponding operations in the relational model: $\cdot$ by relational composition, $+$ by set union, $^*$ by the reflexive-transitive closure operation, and $\leq$ by set inclusion.

# 6 Future Directions

Our repository already contains a significant part of the calculus of variants of Kleene algebras and relation algebras. Extensions for domain-specific applications can be obtained with minor effort. In the context of program development, a large number of laws for dealing with the control structure of programs, as needed by Kleene algebra with tests, relational program semantics, Hoare logic, propositional dynamic logic or the w(l)p calculus, are present. Links with the data flow layer, for instance via the assignment rule of Hoare logic (as described in Section 5), are currently under construction. These will help transforming our repository into a program development and verification environment that could be adapted to support various existing formal methods and perhaps introduce a higher level of simplicity and automation.

We currently envisage the following main directions for future research and development.

**Hypothesis learning:** While Isabelle's relevance filter works impressively well on smaller theory scopes, learning hypotheses in large theories remains difficult. Our repository is very interesting in that respect since it yields a large benchmark suite of similar algebras, in which a similar kind of reasoning is required. It seems particularly useful to complement syntactic techniques, for instance, whether some term in a lemma matches some term in a proof goal, by domain-specific semantic techniques. For instance, a standard trick in ordered structures such as dioids or Boolean algebras is splitting the unit: $x = x \cdot 1 = x \cdot (x + x') = x \cdot x + x \cdot x' = x \cdot x$ proves idempotency of meet in Boolean algebra. How can such tricks be learned?

**Solvers and decision procedures:** Some fragments of Tarski-Kleene algebras are known to be decidable, for instance the equational theories of Kleene algebras, Kleene algebras with tests, (continuous) probabilistic Kleene algebras and concurrent semirings, but only the decision procedure for Kleene algebra is available in Isabelle. For many other fragments, decidability is not known. Sometimes, Horn formulas with antecedents of a particular shape can be reduced to equations. None of these hypothesis elimination algorithms are available in Isabelle, and for most variants of Tarski-Kleene algebras they have not been investigated. Integrating such algorithms could dramatically increase proof automation. In this context, decision procedures are typically based on automata, trees or graphs. Thus their output cannot be directly verified by Isabelle. Such procedures would have to be used as oracles or would have to be verified within Isabelle. Most of the proofs in the repository would only require small data structures in the decision procedures, hence even naive implementations would make a difference.

**Integration of point-free and point-wise reasoning:** The examples in Sections 4 and 5 suggest that these two styles can effectively be combined in our framework. In simple applications, entire verification tasks could probably be blasted away by SMT solvers. More generally, however, updates on program states must be modelled in a concrete semantics (for example, binary relations or predicate transformers) or the abstract algebraic layer must be augmented by rules for assignments and substitutions, as for instance in the B method. The development of specific lemmas and tactics that link the two layers is crucial for applications.

**Design of simple modelling languages:** The taxonomy of algebraic variants, their axioms and lemmas should, to a large extent, be hidden from the users. Instead simple modelling languages should be developed, for example, relational ones similar to that of Alloy. The underlying provers and counterexample generators could be used by developers to guide their semiformal understanding of a system's properties to be analysed.

**Interfaces to formal methods:** Due to their versatility, the structures and properties implemented in the repository are relevant to many applications. A prime example is relational program development for which a variety of tools with their own languages and idiosyncrasies exist. Many of these methods could be supported by creating interfaces to our repository.

# 7 Conclusion

We presented ongoing work on a repository for Tarski-Kleene algebras in Isabelle/HOL which is intended to provide automated proof support for program development methods. The development of the repository and its applicative potential depend strongly on the recent integration of ATP systems, SMT solvers and counterexample generators into Isabelle. Using this technology, new algebraic theories could be engineered quickly and easily, and a high degree of automation should be achievable in practical applications.

While the previous section contains a detailed discussion of ongoing and future work on this project, we conclude the paper with some remarks on automated theorem proving technology.

First of all, order-based reasoning is as important for program development as equational reasoning, for instance, in the context of refinement or when modelling simulation relations. Also reasoning in Tarski-Kleene algebras is, to a large extent, order based. Shifting between the two styles is possible, in principle, since $x = y$ if and only if $x \leq y$ and $y \leq x$, and $x \leq y$ if and only if $x + y = y$, but whereas splitting an equation into inequalities often simplifies proofs, the replacement of inequalities by equations blows up the size of terms and makes proof search more difficult. Ordered chaining calculi [1] have been developed to complement the superposition calculi used in many ATP systems in order to enhance order-based reasoning. But this technology has not been implemented in state-of-the-art tools.

Second, Isabelle's current integration uses only a handful of ATP systems and SMT solvers. Prover9, which on algebraic proof examples often shows the best overall performance [15], is not among them. Standardisation projects for ATP inputs (TPTP) and, in particular, proof output (TSTP) are important here. Via these interfaces, a large class of ATP systems could be accessed via Sutcliffe's System on TPTP. For SMT solvers, similar standards (SMT-LIB) exist.

For programming applications, sorts or types are very important. They are currently supported by only a few ATP systems. Although they can be encoded explicitly as constraints or guards to the algebraic specification, this can drastically slow down the proof search.

**Acknowledgements.** Walter Guttmann was supported by 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] L. Bachmair and H. Ganzinger. Ordered chaining calculi for first-order theories of transitive relations. *Journal of the ACM*, 45(6):1007–1049, 1998.

[2] R. Berghammer. Combining relational calculus and the Dijkstra–Gries method for deriving relational programs. *Information Sciences*, 119(3–4):155–171, 1999.

[3] R. Berghammer. Applying relation algebra and Rel View to solve problems on orders and lattices. *Acta Informatica*, 45(3):211–236, 2008.

[4] R. Berghammer and H. Zierer. Relational algebraic semantics of deterministic and nondeterministic programs. *Theoretical Computer Science*, 43:123–147, 1986.

[5] S. Berghofer and T. Nipkow. Random testing in Isabelle/HOL. In J. R. Cuellar and Z. Liu, editors, *SEFM 2004*, pages 230–239. IEEE Computer Society, 2004.

[6] J. C. Blanchette, S. Böhme, and L. C. Paulson. Extending Sledgehammer with SMT solvers. In *Automated Deduction: CADE-23*, 2011. To appear.

[7] E. Cohen. Separation and reduction. In R. Backhouse and J. N. Oliveira, editors, *MPC 2000*, volume 1837 of *LNCS*, pages 45–59. Springer, 2000.

[8] J. H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, 1971.

[9] J. Desharnais and G. Struth. Internal axioms for domain semirings. *Science of Computer Prog.*, 76(3):181–203, 2011.

[10] S. Foster, G. Struth, and T. Weber. Automated engineering of relational and algebraic methods in Isabelle/HOL. In H. de Swart, editor, *RAMiCS*, volume 6663 of *LNCS*, pages 52–67. Springer, 2011.

[11] J. L. Gischer. The equational theory of pomsets. *Theoretical Computer Science*, 61(2–3):199–224, 1988.

[12] W. Guttmann, G. Struth, and T. Weber. Automating algebraic methods in Isabelle. In *Formal Methods and Software Engineering: ICFEM*, 2011. To appear.

[13] C. A. R. Hoare and J. He. *Unifying theories of programming*. Prentice Hall Europe, 1998.

[14] C. A. R. Hoare, B. Möller, G. Struth, and I. Wehrman. Concurrent Kleene Algebra and its foundations. *Journal of Logic and Algebraic Programming*, 80(6):266–296, 2011.

[15] P. Höfner, G. Struth, and G. Sutcliffe. Automated verification of refinement laws. *Annals of Mathematics and Artificial Intelligence*, 55(1–2):35–62, 2009.

[16] J. Hurd. System description: The Metis proof tactic. In C. Benzmüller, J. Harrison, and C. Schürmann, editors, *ESHOL 2005*, pages 103–104, 2005.

[17] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994.

[18] D. Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems*, 19(3):427–443, 1997.

[19] R. D. Maddux. Relation-algebraic semantics. *Theoretical Computer Science*, 160(1–2):1–85, 1996.

[20] A. K. McIver and T. Weber. Towards automated proof support for probabilistic distributed systems. In G. Sutcliffe and A. Voronkov, editors, *LPAR*, volume 3835 of *LNCS*, pages 534–548. Springer, 2005.

[21] B. Möller and G. Struth. Algebras of modal operators and partial correctness. *Theoretical Computer Science*, 351(2):221–239, 2006.

[22] K. C. Ng. *Relation Algebras with Transitive Closure*. PhD thesis, Univ. of California, Berkeley, 1984.

[23] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[24] W.-P. de Roever and K. Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*. Cambridge University Press, 1998.

[25] N. Schirmer. *Verification of Sequential Imperative Programs in Isabelle/HOL*. PhD thesis, TU München, 2006.

[26] A. Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6(3):73–89, 1941.

[27] J. von Wright. Towards a refinement algebra. *Science of Computer Prog.*, 51(1–2):23–45, 2004.