Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 214 (2008) 81–101

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Modal Tools for Separation and Refinement

Georg Struth [1]

Department of Computer Science
University of Sheffield
Sheffield, United Kingdom

Abstract

Kleene modules and modal Kleene algebras are applied to automatically verify refinement laws for infinite loops and separation of termination. The key concept in this analysis is *divergence* which, in some models, abstractly characterises that part of a state space from which infinite dynamics is possible. In other models it expresses infinite iteration. Equational axioms for divergence are introduced, and the concept is refined for different contexts. In particular it is related to Löb's formula, which describes termination in modal logics.

Keywords: Modal Kleene algebras; Kleene modules; automated deduction; termination and divergence; separation and refinement.

1 Introduction

In an influential paper on *A Calculational Approach to Mathematical Induction*, Doornbos, Backhouse and van der Woude present a complex rearrangement condition for relations that allows the separation of termination of the union of two relations can into termination of the individual relations [11]. This very general rearrangement condition subsumes those previously proposed by Geser [14], and Bachmair and Dershowitz [3].

The DBW-theorem provides a powerful refinement law for relational systems. Relational structures form standard models for discrete system dynamics on state spaces. They include Kripke structures for reactive or multi-agent

1 Email: g.struth@dcs.shef.ac.uk

doi:10.1016/j.entcs.2008.06.005

systems, and input/output behaviours resulting from the actions of imperative programs. The rearrangement condition is a local criterion that abstractly expresses the preference of certain relational steps over others. And by separation of termination, the termination analysis for a complex loop of a relational system, in which relational steps can be executed nondeterministically, can be reduced to the termination analyses of simpler, more deterministic loops.

Doornbos, Backhouse and van der Woude state that the specification and verification of their law has been "a very difficult problem". In fact, their calculational proof in a variant of relation algebra called *regular algebra* covers several pages. But, in contrast to the semi-formal arguments for previous partial results which were based on the explicit analysis of infinite rewrite sequences through diagrams, it is entirely formal.

Motivated by applications in the control and refinement of concurrent systems, Ernie Cohen posed Bachmair and Dershowitz's variant of the termination theorem as a proof challenge for Kleene algebras [6], conjecturing that it *cannot* be proved in this setting. One reason for this negative conjecture might be that Kleene algebras, which are axiomatised within first-order equational logic and which strongly correspond to regular languages [17], are much less expressive than regular algebras, which are essentially second-order relation algebras. But a positive answer to this challenge has recently been published [19]. But the proof and automation of the much more difficult DBW-theorem in a first-order setting remains a challenge.

This paper presents the first automated verification of the DBW-theorem. It is based on modal Kleene algebras [9,10], which have previously been used for a variety of verification and refinement tasks. The key calculational tool comes from the notion of *divergence* [8], which abstractly models that part of a state space at which infinite dynamics may start. Termination can then be characterised as the absence of divergence. Another main contribution is a deeper investigation and a new equational axiomatisation of this concept in the novel and more general setting of divergence Kleene modules, which is inspired by the classical notion of *module* from algebra. In particular, different variants of divergence and termination for different applications are derived. Based on that, a particularly simple modal correspondence result for Löb's formula, which characterises termination in modal logics (cf. [4]), can be obtained. But, most importantly, our modal approach yields powerful tools that not only allow us to give very short and concise proofs of refinement laws like the DBW-theorem, but also to automate them with off-the-shelf automated theorem proving systems (ATP systems).

Divergence Kleene modules have a particularly rich model class that includes relations, traces, paths and languages. They also support simple and combined reasoning with finite and infinite behaviours. Therefore, for the first time, the DBW-theorem becomes available in all these models and hence for various applications. We also provide a very simple automated proof of Bachmair and Dershowitz's termination theorem in these modules relative to a new refinement theorem for infinite loops has been discovered through automated deduction experiments.

Modal logics belong to the most popular tools in program verification. But in the context of program refinement and termination analysis they have so far rather been neglected. This paper demonstrates their relevance to refinement beyond model checking. The integration of modal algebras into off-the-shelf ATP systems yields powerful formal methods with which some complex properties and behaviours can easily be verified.

2 Semirings and Kleene Algebras

This paper considers separation and refinement laws from the point of view of Kleene algebras. This has two main benefits. First, beyond relations, theorems become available also for trace- path- and language-based applications. Second, Kleene algebras are first-order equational structures, and this opens the door for automated deduction.

Kleene algebras provide the essential operations for modelling actions of discrete systems: angelic nondeterministic choice is represented by addition, sequential composition is expressed by multiplication, finite iteration is captured through fixed points. Special constants model abortive and ineffective actions.

A (commutative) *semiring* is a structure $(S, +, \cdot, 0, 1)$ on a set S such that

- $(S, +, 0)$ is a commutative monoid,
- $(S, \cdot, 1)$ is a monoid,
- the distributivity laws $x(y + z) = xy + xz$ and $(x + y)z = xz + yz$ hold, and
- $x0 = 0 = 0x$.

Here and henceforth we omit the multiplication symbol. A semiring is *idempotent* if addition is idempotent, that is $x + x = x$. Every retract $(S, +)$ of an idempotent semiring S is a semilattice, hence idempotent semirings are posets with respect to $x \leq y \Leftrightarrow x + y = y$.

A *Kleene algebra* [17] is an idempotent semiring K extended by the *star* operation * that satisfies, for all $x, y, z \in K$, the *star unfold* and the *star* *induction axioms*

$$1 + xx^* \leq x^*, \qquad z + xy \leq y \Rightarrow x^*z \leq y,$$
$$1 + x^*x \leq x^*, \qquad z + yx \leq y \Rightarrow zx^* \leq y.$$

This axiomatises finite iterations from left to right and from right to left within first-order logic as least prefixed points, which are also least fixed points. It can be shown that all operations of Kleene algebras are isotone with respect to $\leq$. The full axioms of Kleene algebras, as input for the ATP system Prover9 [1], can be found in Appendix B.

It is well known that Kleene algebras have many computationally interesting models. We are mainly interested in relation and trace models.

Example 2.1 Let 2^{A^2} denote the set of all binary relations over some set A. The structure $(2^{A^2}, \cup, ;, {}^*, \emptyset, 1_A)$, where ; denotes the relative product, * the reflexive transitive closure operation, and 1_A the unit relation over A, is a Kleene algebra. It is called the *full relation Kleene algebra* over A. All its subalgebras are again Kleene algebras; so-called *relation Kleene algebras.*

Example 2.2 Let A be an alphabet of action symbols and P an alphabet of state or proposition symbols. A (finite) *trace* over P and A is either the empty trace ϵ or a word $\sigma \in (P \cup A)^*$ in which $\mathsf{first}(\sigma), \mathsf{last}(\sigma) \in P$ and in which letters from P and A alternate. Let $\sigma_1.\sigma_2$ denote the concatenation of words σ_1 and σ_2. The product of traces τ_0 and τ_1 is the trace

$$\tau_0 \cdot \tau_1 = \begin{cases} \sigma_0.p.\sigma_1 & \text{if } \tau_0 = \sigma_0.p \text{ and } \tau_1 = p.\sigma_1, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

So $\tau_0 \cdot \tau_1$ glues two traces together if the last state symbol of τ_0 and the first state symbol of τ_1 are equal. The set of all traces over P and A is denoted by $(P, A)^*$. The power-set algebra $2^{(P,A)^*}$ with addition defined by set union, multiplication by

$$x \cdot y = \{\tau_0 \cdot \tau_1 : \tau_0 \in x, \tau_1 \in y \text{ and } \tau_0 \cdot \tau_1 \text{ defined}\},$$

the star by $x^* = \bigcup_{i \geq 0} x^i$, with powers x^i recursively defined, and with $\emptyset$ and P as neutral elements is a Kleene algebra. It is called the *full trace Kleene algebra* over P and A. Its subalgebras are again Kleene algebras; the so-called *trace Kleene algebras.*

Example 2.3 *Path Kleene algebras* are obtained from trace Kleene algebras by forgetting actions. More precisely, they are isomorphic to trace Kleene algebras over $(P, \{a\})^*$.

Example 2.4 *Language Kleene algebras* are obtained from trace Kleene algebras by forgetting propositions. More precisely, they are isomorphic to trace Kleene algebras over $(\{p\}, A)^*$.

Further models can be obtained by relaxing the Kleene algebra axioms. Positively conjunctive predicate transformers, for instance, are captured by Kleene algebras in which the right annihilation axiom $a0 = 0$ is dropped—the so-called *demonic refinement algebras* [20]. However, infinite behaviours and termination cannot be expressed in Kleene algebras and, for instance, reactive systems cannot properly be analysed.

Kleene algebras have been extended by an operation for modelling infinite iterations via greatest fixed points. An *omega algebra* [5] is a Kleene algebra K extended by the omega operation ${}^\omega : K \to K$ that satisfies the *omega unfold* and the *omega coinduction* axioms

$$xx^\omega = x^\omega \qquad \text{and} \qquad y \le z + xy \Rightarrow y \le x^\omega + x^* z.$$

In this setting, termination of an infinite loop x^ω can be expressed as $x^\omega = 0$. It is has been shown that this operation captures infinite and terminating behaviour properly on relation Kleene algebras. However, it behaves anomalously on more general relational structures and on trace, path and language Kleene algebras [8,15]. This is our main motivation for considering terminating and infinite, diverging dynamics in a different, modal setting.

3 Domain Semirings and Kleene Algebras with Domain

Modal box and diamond operators can be defined on semirings and Kleene algebras via a *domain* operation. In relation Kleene algebras, obviously, the domain $d(x)$ of a binary relation x is the set of all elements a such that $(a, b) \in x$. Also, $d(x)$ is the *least left preserver* of x, that is, it is the least set that satisfies $x \le d(x)x$ or even $x = d(x)x$. Similarly, in trace Kleene algebras, when x represents a set of traces, $d(x) = \{\mathsf{first}(\tau) : \tau \in x\}$ yields the starting states of traces in x. Operationally, a domain element $d(x)$ yields precisely those states at which action x is enabled.

In the more abstract setting of idempotent semirings, a domain operation is an endofunction that induces an appropriate set or propositional structure as its image.

A *domain semiring* [10] is a semiring $(S, +, \cdot, 0, 1)$ extended by a function

$d : S \to S$ that, for all $x, y \in S$, satisfies

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

It has been shown that every domain semiring is automatically idempotent. Let $d(S)$ denote the set of all domain elements of S. Then

$$d(S) = \{x \in S : d(x) = x\},$$

whence domain elements are precisely the fixpoints of the domain operation, and this can be used to show that the *domain algebra* $(d(S), +, \cdot, 0, 1)$ is a bounded distributive lattice that abstractly represents the state space on which the system modelled by S acts. Also, the domain operation is isotone with respect to $\leq$.

We will consistently use $x, y, \dots$ for arbitrary actions and $p, q, \dots$ for domain elements.

The domain algebra of domain semirings can be turned into a Boolean algebra—perhaps the most natural model of a state space—by adding an *antidomain operation* $a : S \to S$ that satisfies $d(x) + a(x) = 1$ and $d(x)a(x) = 0$. The resulting structures are called *Boolean domain semirings.*

It can be shown that $d(a(x)) = a(x)$ such that all antidomain elements belong to the domain algebra. It follows that domain and antidomain elements are Boolean complements , that is, $a(x) = \overline{d(x)}$. We will also use the Boolean difference operation $p - q = p \cdot \overline{q}$ and the corresponding Galois connection

$$p - q \leq r \Leftrightarrow p \leq q + r \tag{1}$$

on the domain algebra. Since $a^2(x) = d(x)$ holds on Boolean domain semirings, domain can be eliminated in all axioms. It turns out that a semiring is a Boolean domain semiring if and only if it satisfies the axioms

$$a(x)x = 0, \qquad a(xy) = a(xa^2(y)), \qquad a^2(x) + a(x) = 1.$$

This axiomatisation is extremely compact and therefore well-suited for ATP systems. The domain and antidomain axioms also capture the main intuition behind these concepts. Domain elements are indeed least left preservers, that is, all $x \in S$ and $p \in d(S)$ satisfy

$$x \leq px \Leftrightarrow d(x) \leq p. \tag{2}$$

Dually, antidomain elements are *greatest left annihilators*, whence all $x \in S$

and $p \in d(S)$ satisfy

$$px = 0 \Leftrightarrow p \leq a(x). \tag{3}$$

Many further natural properties of domain and antidomain that hold on trace and relational models can readily be verified. Finally, the interaction of star and domain is trivial and does not require further axioms. So a *Kleene algebra with (Boolean) domain* is simply a (Boolean) domain semiring that is also a Kleene algebra.

4 Modal Kleene Algebras and Kleene Modules

It is well known that modal operators can be defined via domain and codomain operations [9]. Intuitively, the link is provided by the fact that Kripke frames are relational structures and forward and backward diamond operators correspond to relational preimage and image operations on these models.

Here, we consider preimages more generally in the setting of semiring elements and domain elements of a domain semiring S. We define (multi)modal diamond operators of type $S \times d(S) \to d(S)$ over domain algebras that are either distributive lattices or Boolean algebras by

$$\langle x \rangle p = d(xp),$$

for each $x \in S$ and $p \in d(S)$. To justify that these diamonds are indeed modalities in the sense of Jónsson and Tarski's Boolean algebras with operators [16], we must show that $\lambda p.\langle x \rangle p$ is strict and additive. We do this in a more general context.

A *Kleene module* [18] is a structure $(K, L, :)$, such that K is a Kleene algebra, L is a semilattice (with least upper bound operation $+$ and least element 0) and the scalar product $:$ of type $S \times L \to L$ satisfies, for all $x, y \in K$ and $p, q \in L$, the axioms

$$(x + y)p = xp + yp, \tag{M1}$$
$$x(p + q) = xp + xq, \tag{M2}$$
$$(xy)p = x(yp), \tag{M3}$$
$$1p = p, \tag{M4}$$
$$x0 = 0, \tag{M5}$$
$$p + xq \leq q \Rightarrow x^*p \leq q. \tag{M6}$$

Here and henceforth we omit the scalar product symbol. Kleene modules are of course strongly inspired by the modules used in algebra, and all axioms except (M6) are inherited from this setting.

Proposition 4.1 ([10]) *Let K be a Kleene algebra with domain. The structure $(K, d(K), (\lambda x, p.\langle x\rangle p))$ is a Kleene module.*

The module axioms (M2) and (M5) are additivity and strictness, hence the diamonds defined via preimages are indeed modal operators. Accordingly, domain semirings and Kleene algebras with domain have also been called *modal semirings* and *modal Kleene algebras.*

It has been shown that module axiom (M5) can be replaced by an equation.

Lemma 4.2 ([13]) *In every Kleene module, the induction axiom (M6) is equivalent to* Segerberg's formula

$$x^*p - p \le x^*(xp - p). \tag{4}$$

Segerberg's original formula, of which (4) is an algebraic variant, appears, for instance, in dynamic and temporal logics. An entirely equational axiomatisation of Kleene modules can be obtained by considering the absolutely free algebra with the signature of Kleene algebras and by adding the axiom $x^*p = (1 + xx^*)p$ [13].

Semiring modules and Kleene modules are weaker than modal semirings and modal Kleene algebras. The following fact does not hold in the module-based setting. An endofunction f on a semilattice L is *completely additive* if it commutes with all existing suprema.

Proposition 4.3 *Domain operators on domain semirings are completely additive.*

Proof Let S be a domain semiring and let $y = \sup(x : x \in A) = \sup(A)$ for some $A \subseteq S$. We show that $d(y)$ is the least upper bound of $d(A)$. Since domain is isotone, $d(y) \ge d(x)$ for all $x \in A$, hence $d(y)$ is an upper bound of $d(A)$.

Let now $p \in d(S)$ be another upper bound of $d(A)$, that is, let $p \ge x$ hold for all $x \in A$. By (2) this is equivalent to $x \le px$, and $x \le y$ then implies $x \le py$. So py is also an upper bound of A. But then $y \le py$, which is equivalent to $d(y) \le p$ by (2), and $d(y)$ is a least upper bound of $d(A)$. We can now conclude that $d(\sup(A)) = d(y) = \sup(d(A))$, and therefore d is completely additive. $\square$

5 Divergence and Termination

Divergence is a powerful operation on Kleene modules and modal Kleene algebras that abstractly characterises infinite and terminating behaviours in discrete dynamic systems.

A *divergence Kleene module* (or a ∇-Kleene module) is a Kleene module $(K, L, :)$ extended by the *divergence operation* ${}^\triangledown : K \to L$ that, for all $x \in K$ and $p, q \in L$, satisfies the *divergence unfold* (∇-unfold) and the *divergence coinduction* (∇-coinduction) axioms

$$x^\triangledown \leq xx^\triangledown \qquad \text{and} \qquad p \leq xp + q \Rightarrow p \leq x^\triangledown + x^* q.$$

These axioms are precisely those of Cohen's omega algebra relativised to a two-sorted setting. ∇-Kleene modules are very versatile and have a rich model class.

As a first interpretation, let K be a modal Kleene algebra. To turn K into a divergence Kleene module, we extend K by a mapping ${}^\triangledown : K \to d(K)$ that satisfies, for all $x \in K$ and $p \in d(K)$,

$$x^\triangledown \leq \langle x \rangle x^\triangledown \qquad \text{and} \qquad p \leq \langle x \rangle p + q \Rightarrow p \leq x^\triangledown + \langle x^* \rangle q.$$

The resulting structures are called *divergence Kleene algebras* (or ∇-Kleene algebras) [8]. It is easy to see that $x^\triangledown$ denotes that element of $d(K)$ from which infinite iterations of x may arise. By the ∇-unfold axiom, which can be strengthened to $x^\triangledown = xx^\triangledown$, the set $x^\triangledown$ is stable, that is, $x^\triangledown$ is a set to which x-actions always may return, whence loop infinitely. By ∇-coinduction, $x^\triangledown$ is the greatest element with that property. This intuition can be confirmed on relation, trace, paths and language models [15].

On ∇-Kleene modules, termination can be characterised as the absence of divergence. We say that an element x *terminates* or is *Noetherian* if

$$x^\triangledown = 0.$$

This condition obviously rules out infinite progress or, in relational models, the presence of infinitely ascending x-chains. Termination on trace, path and language models is captured as well.

Operationally, it is very useful to relate termination with coinduction, since the ∇-coinduction rule acts as an iteration elimination rule.

Lemma 5.1 *An element x of a ∇-Kleene module terminates if and only if*

$$p \leq xp + q \Rightarrow p \leq x^* q. \tag{5}$$

Proof First, if $x^\triangledown = 0$, then ∇-coinduction reduces to (5). Second, by ∇-unfold, $x^\triangledown$ satisfies the antecedent of (5) with $q = 0$. Therefore, by strictness of diamonds, $x^\triangledown \leq 0$. □

Whenever the semilattice of the ∇-Kleene module is a Boolean algebra, the situation is even simpler. Again motivated by relational modules, we define the *final part* of p with respect to x as

$$\Omega_x(p) = p - \langle x \rangle p. \tag{6}$$

This represents that part of p from which no further x-actions are possible. The divergence coinduction axiom can now be simplified.

Lemma 5.2 ([8]) *For every ∇-Kleene module over a Boolean algebra, ∇-coinduction is equivalent to*

$$\Omega_x(p) = 0 \Rightarrow p \leq x^{\nabla}. \tag{7}$$

This holds by fixed point fusion which requires the Galois connection (1) for Boolean difference. Now, also (5) can be simplified.

Corollary 5.3 *An element x of a ∇-Kleene module over a Boolean algebra terminates if and only if*

$$\Omega_x(p) = 0 \Rightarrow p = 0. \tag{8}$$

The proof is analogous to that of Lemma 5.1, and property (8) is well known from regular algebra [11]. Interestingly, in the Boolean case, divergence and termination can be defined without the star, that is, on modal semirings alone. This gives rise to ∇-*semiring modules* and ∇-*semirings.*

All concepts and result so far, that have been motivated through modal Kleene algebras, but defined and proved in the more general context of Kleene modules, allow a different interpretation of a ∇-Kleene module $(K, L, :, ^{\nabla})$. Let now K denote a set of finite and L a set of infinite computations. Let the scalar product map finite computations from K and infinite computations from L to infinite computations. This approach makes it impossible to compose an infinite element at its right-hand side with any other element, but the composition of infinite elements and infinite elements with finite elements at their right-hand sides is impossible. The divergence operation maps finite elements to infinite ones and acts as a revised omega operator. $x^{\nabla} = 0$ models again the absence of infinite iteration, hence termination. If the semilattice L of infinite actions is a Boolean algebra, $\Omega_x(p)$ can again be defined. It now denotes those infinite computations in p which do not possess a finite prefix x. Also the statements of Lemma 5.1, Lemma 5.2 and Corollary 5.3 remain meaningful.

Under this interpretation, the main relevant models are not relations, but combinations of finite and infinite traces, paths and languages, as they are known from the theory of Büchi automata and ω-regular languages. We have already seen in Example 2.2 that finite traces form Kleene algebras. So the first component of our ∇-Kleene module is well defined. For the second component, we define an *infinite trace* over P and A as an (infinite) alternating sequence of letters from P and letters from A, starting with a letter from P. We will use π and ρ for infinite traces. For a finite trace τ and an infinite trace π we define their product as the infinite trace

$$\tau \cdot \pi = \begin{cases} \sigma.p.\rho & \text{if } \tau = \sigma.p \text{ and } \pi = p.\rho, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The set of all infinite traces over P and A is denoted by $(P,A)^\omega$. The union of sets of infinite traces is defined in the obvious way. For a set x of finite traces and a set p of infinite traces we define the scalar product

$$x : p = \{\tau \cdot \pi : \tau \in x, \pi \in p \text{ and } \tau.\pi \text{ defined}\}.$$

We also define the divergence as

$$x^\nabla = \{\pi \in (P,A)^\omega : \pi = \tau_0 \cdot \tau_1 \cdot \ldots \text{ with } \tau_i \in (P,A)^* \text{ for } i \geq 0\}.$$

Routine calculations then show the following fact.

Proposition 5.4 $(2^{(P,A)^*}, 2^{(P,A)^\omega}, :, ^\nabla)$ *is a* ∇*-Kleene module.*

We call it the *full trace* ∇*-Kleene module* over P and A. Again, all subalgebras are ∇-Kleene modules which we call *trace* ∇*-Kleene modules*. As for finite traces, we obtain path ∇-Kleene modules or language ∇-Kleene modules by forgetting actions or propositions. In particular, the ω-regular languages form ∇-Kleene modules; the question whether they form the free divergence Kleene modules seems very interesting.

6 Equational Axioms for Divergence

Section 4 showed that the induction axiom of Kleene modules, which is a quasi-identity, is equivalent to Segerberg's formula (4), which is an identity. Motivated by this result, we now develop equational axioms for divergence. As before, this depends on the presence of complementation, so we restrict our attention to ∇-Kleene modules over Boolean algebras and the corresponding modal Kleene algebras.

Theorem 6.1 *For all ∇-Kleene modules over a Boolean algebra, (7) is equivalent to the identity*

$$p \leq x^{\nabla} + x^{*}\Omega_x(p). \tag{9}$$

Proof First, assume that (7) holds. For (9), it suffices by ∇-coinduction, which is equivalent to (7), to show that $p \leq xp + \Omega_x(p) = xp + (p - xp)$. But this holds in every Boolean algebra.

Let now (9) hold and we prove (7), that is, we assume $\Omega_x(p) = 0$ and must show $p \leq x^{\nabla}$. By (9) in the first step, the assumption in the second step and the module axiom $x0 = 0$ in the third step, $p \leq x^{\nabla} + x^{*}\Omega_x(p) = x^{\nabla} + x^{*}0 = x^{\nabla} + 0 = x^{\nabla}$. □

This immediately yields yet another equational characterisation of termination.

Theorem 6.2 *An element x of some ∇-Kleene module over a Boolean algebra terminates if and only if*

$$p \leq x^{*}\Omega_x(p). \tag{10}$$

Proof First, let x terminate. Then (10) follows immediately from setting $x^{\nabla} = 0$ in identity (9).

Now assume (10). We further assume $\Omega_x(p) = 0$ and show that $p = 0$ since, by Corollary 5.3 this is equivalent to $x^{\nabla} = 0$. But $p \leq x^{*}\Omega_x(p) = x^{*}0 = 0$ holds by the assumptions and the module axiom $x0 = 0$. □

Obviously, if x satisfies (10), then every set p is contained in that set of states from which the terminal elements of p can be reached by a finite x-iteration.

Identity (10) has already been shown to be equivalent—in a more pedestrian way—to termination in the slightly less general setting of ∇-Kleene algebras [10], whereas the equational characterisation of divergence is novel. Identity (10) is strongly related to *Löb's formula* from modal logic which, translated into Kleene modules, is

$$xp \leq x\Omega_x(p). \tag{11}$$

In fact, (10) replaces Löb's formula in situations where the star is available. It has automatically been verified that every element of a ∇-Kleene algebra that satisfies Löb's formula is Noetherian and that each element x that satisfies the diamond transitivity law $\langle x\rangle\langle x\rangle p \leq \langle x\rangle p$ and (10) also satisfies Löb's formula [10]. All these proofs can easily be replayed in the more general setting of ∇-Kleene modules.

Here, however, we are more interested in (10) and related formulas as computational tools. Moreover, the equational characterisations of divergence and termination are interesting because they yield purely equational characterisations of finite and infinite loops. Following our discussion in Section 4, also the star can be equationally characterised in the setting of Kleene modules by adding further axioms. This can be advantageous for automated theorem proving, but also from the abstract algebraic point of view, because equational theories admit particularly powerful constructions.

7 Quasicommutation and Separation of Termination

This section is a warm-up before proving the more difficult DBW-theorem. It also allows us to formulate some questions related to this theorem.

Bachmair and Dershowitz's termination theorem states that termination of the union of two rewrite systems can be separated into termination of the individual rewrite systems if one system quasicommutes over the other. In fact, the statement of the theorem and its proof is purely relational and can be translated into Kleene algebras. In this more abstract setting, the theorem yields a refinement law for separating termination of some complex system into termination of simpler ones.

Let K be a Kleene algebra and let $x, y \in K$. Then x *quasicommutes* over y if

$$yx \leq x(x+y)^*. \tag{12}$$

This conditions abstractly expresses that each sequence in which a y-action precedes an x-action can be rearranged by an x-action followed by arbitrary finite (possibly empty) sequences of x-actions and y-actions. We also say that *termination of x and y can be separated* if

$$(x+y)^\triangledown = 0 \Leftrightarrow x^\triangledown + y^\triangledown = 0.$$

Clearly, this states that $x+y$ terminates if and only if x and y terminate separately.

However, experiments with ATP-systems for proving this theorem showed that the termination theorem arises as a special case of the following refinement theorem for infinite loops.

Theorem 7.1 *Let x and y be elements of some ∇-Kleene module and let x quasicommute over y. Then*

$$(x+y)^\triangledown = x^\triangledown + x^*y^\triangledown. \tag{13}$$

Proof The right-to-left direction holds without assuming quasicommutation by isotonicity of divergence. The equational proof for the left-to-right direction is essentially a one-liner:

$$(x+y)^\nabla = y^\nabla + y^* x(x+y)^\nabla \leq y^\nabla + x(x+y)^*(x+y)^\nabla = y^\nabla + x(x+y)^\nabla \leq x^\nabla + x^* y^\nabla.$$

The last step, in particular, uses ∇-coinduction. All other steps use simple identities that have previously been automatically verified. □

This theorem states that, in the presence of quasicommutation, an infinite loop, in which two actions are nondeterministically executed, can be separated into two deterministic infinite loops. A fully automated proof with the ATP system SPASS [2], which is particularly suited for the two-sorted setting of divergence Kleene modules, has also been given. The left-to right direction that uses quasicommutation took about two minutes on a standard PC. The converse direction took about three minutes.

Bachmair and Dershowitz's termination theorem is an immediate consequence of Theorem 7.1.

Corollary 7.2 *Termination of elements x and y of some ∇-Kleene module can be separated if x quasicommutes over y.*

Proof First, by isotonicity of divergence, $(x+y)^\nabla = 0$ implies $x^\nabla = 0$ and $y^\nabla = 0$. Second, if $x^\nabla = 0$ and $y^\nabla = 0$, then $(x+y)^\nabla = 0$ by Theorem 7.1. □

Automation of the left-to-right direction took about three minutes; its converse, assuming Theorem 7.1 was immediate. An input file for SPASS is presented in Appendix A.

Compared to the original results, that were valid only for binary relations, the more abstract setting of ∇-Kleene modules encompasses also trace, path and language models, and in particular the combination of finite and infinite objects. This makes the loop refinement theorem and the termination theorem of this section available for diverse applications beyond rewriting.

8 Lazy Commutation and Separation of Termination

We now prove the main application of this paper, the termination theorem of Doornbos, Backhouse and van der Woude [11]. Compared to Bachmair and Dershowitz's theorem, only the assumption of quasicommutation needs to be relaxed to

$$yx \leq x(x+y)^* + y. \tag{14}$$

Nachum Dershowitz has suggested to call this rearrangement property *lazy commutation* [7], so we say that x *lazily commutes* over y if the above property holds. Lazy commutation can be extended to iterations.

Lemma 8.1 *In every Kleene algebra, x lazily commutes over y if and only if*

$$yx^* \leq x(x+y)^* + y. \tag{15}$$

Proof Obviously, (15) implies (14) since $x \leq x^*$. For the converse implication it suffices, by star induction, that $y + (x(x+y)^* + y)x \leq x(x+y)^* + y$. To prove this claim, we calculate

$$\begin{aligned} y + (x(x+y)^* + y)x &= y + x(x+y)^*x + yx \\ &\leq y + x(x+y)^*(x+y) + x(x+y)^* + y \\ &= x(x+y)^* + y. \end{aligned}$$

The individual steps use standard properties of Kleene algebras. □

This fact has already been observed in regular algebra [11]. An automated proof could be found in less than a minute by Prover9.

We can now state and prove the DBW-theorem.

Theorem 8.2 *Termination of elements x and y of some ∇-Kleene module over a Boolean algebra can be separated if x lazily commutes over y.*

Proof First, termination of $x+y$ implies termination of x and y by the results (and the automated proof) of the previous section.

So let us consider the converse implication. We abbreviate $\nabla = (x+y)^\triangledown$. By termination of x and y via Corollary 5.3, a sufficient condition for $\nabla = 0$ is $\Omega_y(\Omega_x(\nabla)) = 0$, which, by definition of Ω_y, amounts to showing that

$$\Omega_x(\nabla) \leq y\Omega_x(\nabla). \tag{16}$$

To prove this claim, we further use the identities

$$\Omega_x(p) \leq p, \tag{17}$$

$$(x+y)^*\nabla = \nabla. \tag{18}$$

Identity (17) holds by Boolean algebra, whereas identity (18), a special case of $x^\triangledown = x^* x^\triangledown$, holds in every ∇-Kleene module. We can now calculate

$$\begin{aligned} \nabla &= x\nabla + y\nabla \\ &\leq x\nabla + yx^*\Omega_x(\nabla) \\ &\leq x\nabla + x(x+y)^*\Omega_x(\nabla) + y\Omega_x(\nabla) \\ &\leq x\nabla + x(x+y)^*\nabla + y\Omega_x(\nabla) \\ &= x\nabla + y\Omega_x(\nabla). \end{aligned}$$

The first step uses ∇-unfold. The second step uses (10), which is equivalent to Noethericity of x. The third step uses the assumption of lazy commutation. The fourth step uses (17). The fifth step uses (18) and idempotency of addition. Now the claim (16) follows from the Galois connection (1) and the definition of Ω_x. $\square$

We could automate the proof of (16) with Prover9. As an alternative to the proofs in the previous section, we worked in the setting of ∇-Kleene algebras instead of the slightly more general module-based setting to demonstrate the applicability of both approaches. (17) could be proved automatically in less than a second. The "$\leq$" direction of (18) could be proved in less than two minutes. Its converse needed less than a second; isotonicity of diamonds, locality of domain and $xx^* = x^*x$ were used as additional hypotheses. Proofs with less additional hypotheses could probably be obtained with more patience. Using (17) and (18) together with further axioms of ∇-Kleene algebras as additional hypotheses and restricting the set of axioms, the claim (16) could then be proved in less than two minutes. A full automation of Theorem 8.2 only from the axioms of ∇-Kleene algebras or ∇-Kleene modules remains a challenge for ATP systems.

The proof of the DBW-theorem is loosely inspired by another proof in a second-order modal algebra [12]. It is, however, much shorter, more structured and, for the first time, suitable for automation. It is also by far simpler than the original proof by Doornbos, Backhouse and van der Woude. Moreover, our study in the setting of divergence Kleene module makes the DBW-theorem valid beyond relations in trace, path and language models and for combined finite and infinite behaviours. This opens the door to various applications.

An obvious question is whether the proof given could be translated into omega algebras. The direct answer is negative. The function Ω and the identity (10), which are used in the proof, are only available when the second component of the module admits the Boolean difference operation. But there also is no omega algebra with less than 10 elements on which the DBW-theorem fails. Therefore, in a wider sense, the question remains open.

Another interesting question is whether the DBW-theorem is again a corollary to some loop refinement theorem similar to Theorem 7.1. But we could so far neither prove such a theorem nor find a counterexample to (13). This question certainly deserves further investigation.

9 Conclusion

Based on divergence Kleene modules and divergence Kleene algebras, we analysed some modal notions of divergence and termination that are interesting for discrete dynamical systems based on relation, trace, path, or language semantics. We developed equational characterisations for divergence and termination, and studied the interplay of finite and infinite traces, paths and languages in an algebraic way. Developing the appropriate concepts and tools presents one of the main achievements of this paper. We then applied these modal tools by automatically verifying a refinement law for infinite loops in the presence of quasicommutation, which rearranges actions of a system. We also automatically verified two separation or refinement laws for termination, one of which is again based on quasitermination, whereas the second one depends on a more general notion of lazy commutation. The proof of this second law, which is rather complex, is much shorter and more structured that previous proofs in the second-order setting of regular algebras. Moreover, because of the generality of our approach, these laws become available in a large class of computationally interesting models, including relations, traces, paths and languages, and therefore in more diverse applications.

Our results suggest that the modal tools developed suitably complement those traditionally used for analysing the refinement, in particular separability and termination, of programs and systems, and that this analysis can be supported by automated theorem provers. By the integration of computational algebras with ATP systems, it seems possible to bridge the prevailing gap between model checking and interactive theorem proving in refinement and to obtain novel light-weight formal methods with heavy-weight automation for formal software engineering.

Acknowledgement

I am very grateful to Nachum Dershowitz, Jules Desharnais and Bernhard Möller for inspiring discussions and suggestions.

References

[1] *Prover9 and Mace4*, http://www.cs.unm.edu/~mccune/prover9.

[2] *Spass 3.0*, http://spass.mpi-inf.mpg.de/.

[3] Bachmair, L. and N. Dershowitz, *Commutation, transformation, and termination*, in: J. H. Siekmann, editor, *8th International Conference on Automated Deduction*, LNCS **230** (1986), pp. 5–20.

[4] Blackburn, P., M. de Rijke and Y. Venema, "Modal Logic," Cambridge University Press, 2001.

[5] Cohen, E., *Separation and reduction*, in: R. Backhouse and J. N. Oliveira, editors, *Mathematics of Program Construction (MPC 2000)*, LNCS **1837** (2000), pp. 45–59.

[6] Cohen, E., *Omega algebra: the good, the bad, and the ugly*, in: R. Backhouse, D. Kozen and B. Möller, editors, *Applications of Kleene Algebra, Report of the Dagstuhl Seminar 01081*, 2001, p. 5.

[7] Dershowitz, N., *Personal communication*, October 2007.

[8] Desharnais, J., B. Möller and G. Struth, *Termination in modal Kleene algebra*, in: J.-J. Lévy, E. W. Mayr and J. C. Mitchell, editors, *IFIP TCS2004* (2004), pp. 647–660, revised version: *Algebraic Notions of Termination.* Technical Report 2006-23, Institut für Informatik, Universität Augsburg, 2006.

[9] Desharnais, J., B. Möller and G. Struth, *Kleene algebra with domain*, ACM Trans. Computational Logic **7** (2006), pp. 798–833.

[10] Desharnais, J. and G. Struth, *Domain semirings revisited*, Technical Report CS-08-01, Department of Computer Science, University of Sheffield (2008), accepted for MPC 2008.

[11] Doornbos, H., R. C. Backhouse and J. van der Woude, *A calculational approach to mathematical induction*, Theoretical Computer Science **179** (1997), pp. 103–135.

[12] Doornbos, H. and B. von Karger, *On the union of well-founded relations*, L. J. of the IGPL **6** (1998), pp. 195–201.

[13] Ehm, T., B. Möller and G. Struth, *Kleene modules*, in: R. Berghammer, B. Möller and G. Struth, editors, *Relational and Kleene-Algebraic Methods in Computer Science*, LNCS **3051** (2004), pp. 112–123.

[14] Geser, A., "Relative Termination," Ph.D. thesis, Fakultät für Mathematik und Informatik, Universität Passau (1990).

[15] Höfner, P. and G. Struth, *Nontermination in idempotent semirings*, in: R. Berghammer, B. Möller and G. Struth, editors, *Relations and Kleene Algebras in Computer Science*, LNCS **4988**, 2008, pp. 206–220.

[16] Jónsson, B. and A. Tarski, *Boolean algebras with operators, Part I*, American Journal of Mathematics **73** (1951), pp. 891–939.

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

[18] Leiß, H., *Kleene modules and linear languages*, Journal of Logic and Algebraic Programming **66** (2006), pp. 185–194.

[19] Struth, G., *Abstract abstract reduction*, Journal of Logic and Algebraic Programming **66** (2006), pp. 239–270.

[20] von Wright, J., *Towards a refinement algebra*, Science of Computer Programming **51** (2004), pp. 23–45.

A Input for SPASS

```
begin_problem(modules).

list_of_descriptions.
name({*Kleene Modules*}).
author({*Georg Struth*}).
status(satisfiable).
description({*Axioms for divergence modules, some derived properties*}).
end_of_list.

list_of_symbols.
  functions[(kplus,2),(ktimes,2),(k0,0),(k1,0),(star,1),
            (lplus,2),(l0,0),
            (scalar,2),
            (nabla,1)].
  predicates[(kleq,2),(lleq,2)].
  sorts[kleene,slat].
```

```
end_of_list.

list_of_declarations.
  kleene(k0).
  kleene(k1).
  forall([kleene(x),kleene(y)],kleene(kplus(x,y))).
  forall([kleene(x),kleene(y)],kleene(ktimes(x,y))).
  forall([kleene(x)],kleene(star(x))).

  slat(l0).
  forall([slat(x),slat(y)],slat(lplus(x,y))).

  forall([kleene(x),slat(p)],slat(scalar(x,p))).

  forall([kleene(x)],slat(nabla(x))).
end_of_list.

list_of_formulae(axioms).

% kleene additive monoid
  formula(forall([kleene(x),kleene(y),kleene(z)],
    equal(kplus(kplus(x,y),z),kplus(x,kplus(y,z))))).
  formula(forall([kleene(x)],equal(kplus(x,k0),x))).
  formula(forall([kleene(x),kleene(y)],equal(kplus(x,y),kplus(y,x)))).
  formula(forall([kleene(x)],equal(plus(x,x),x))).

% kleene multiplicative monoid
  formula(forall([kleene(x),kleene(y),kleene(z)],
    equal(ktimes(ktimes(x,y),z),ktimes(x,ktimes(y,z))))).
  formula(forall([kleene(x)],equal(ktimes(x,k1),x))).
  formula(forall([kleene(x)],equal(ktimes(k1,x),x))).

% kleene distributivity laws
  formula(forall([kleene(x),kleene(y),kleene(z)],
    equal(ktimes(x,kplus(y,z)),kplus(ktimes(x,y),ktimes(x,z))))).
  formula(forall([kleene(x),kleene(y),kleene(z)],
    equal(ktimes(kplus(x,y),z),kplus(ktimes(x,z),ktimes(y,z))))).

% kleene zero axioms
  formula(forall([kleene(x)],equal(ktimes(x,k0),k0))).
  formula(forall([kleene(x)],equal(ktimes(k0,x),k0))).

% kleene preorder axioms
  formula(forall([kleene(x)],kleq(x,x))).
  formula(forall([kleene(x),kleene(y),kleene(z)],
    implies(and(kleq(x,y),kleq(y,z)),kleq(x,z)))).

% kleene star axioms
  formula(forall([kleene(x)],equal(star(x),kplus(k1,ktimes(x,star(x)))))).
  formula(forall([kleene(x)],equal(star(x),kplus(k1,ktimes(star(x),x))))).
  formula(forall([kleene(x),kleene(y),kleene(z)],
    implies(kleq(kplus(y,ktimes(x,z)),z),kleq(ktimes(star(x),y),z)))).
  formula(forall([kleene(x),kleene(y),kleene(z)],
    implies(kleq(kplus(y,ktimes(z,x)),z),kleq(ktimes(y,star(x)),z)))).

% kleene isotonicity laws
  formula(forall([kleene(x),kleene(y),kleene(z)],
    implies(kleq(x,y),kleq(kplus(z,x),kplus(z,y))))).
  formula(forall([kleene(x),kleene(y),kleene(z)],
    implies(kleq(x,y),kleq(kplus(x,z),kplus(y,z))))).
  formula(forall([kleene(x),kleene(y),kleene(z)],
    implies(kleq(x,y),kleq(ktimes(z,x),ktimes(z,y))))).
    formula(forall([kleene(x),kleene(y),kleene(z)],
      implies(kleq(x,y),kleq(ktimes(x,z),ktimes(y,z))))).
    formula(forall([kleene(x),kleene(y)],
      implies(kleq(x,y),kleq(star(x),star(y))))).

  % kleene splitting law
    formula(forall([kleene(x),kleene(y),kleene(z)],
      equiv(kleq(kplus(x,y),z),and(kleq(x,z),kleq(y,z))))).

  % semilattice axioms
    formula(forall([slat(x),slat(y),slat(z)],
      equal(kplus(lplus(x,y),z),lplus(x,kplus(y,z))))).
    formula(forall([slat(x)],equal(lplus(x,l0),x))).
    formula(forall([slat(x),slat(y)],equal(lplus(x,y),lplus(y,x)))).

  % semilattice preorder axioms
    formula(forall([slat(x)],lleq(x,x))).
    formula(forall([slat(x),slat(y),slat(z)],
      implies(and(lleq(x,y),lleq(y,z)),lleq(x,z)))).

  % semilattice isotonicity laws
    formula(forall([slat(x),slat(y),slat(z)],
      implies(lleq(x,y),lleq(lplus(z,x),lplus(z,y))))).
    formula(forall([slat(x),slat(y),slat(z)],
      implies(lleq(x,y),lleq(lplus(x,z),lplus(y,z))))).

  % kleene splitting law
    formula(forall([slat(x),slat(y),slat(z)],
      equiv(lleq(lplus(x,y),z),and(lleq(x,z),lleq(y,z))))).

  % module axioms
    formula(forall([kleene(x),kleene(y),slat(p)],
      equal(scalar(kplus(x,y),p),lplus(scalar(x,p),scalar(y,p))))).
    formula(forall([kleene(x),slat(p),slat(q)],
      equal(scalar(x,lplus(p,q)),lplus(scalar(x,p),scalar(x,q))))).
    formula(forall([kleene(x),kleene(y),slat(p)],
      equal(scalar(kplus(x,y),p),scalar(x,scalar(y,p))))).
    formula(forall([slat(p)],equal(scalar(k1,p),p))).
    formula(forall([kleene(x)],equal(scalar(x,l0),l0))).
    formula(forall([kleene(x),slat(p),slat(q),slat(r)],
      implies(lleq(lplus(scalar(x,p),q),r),lleq(scalar(star(x),q),r)))).

  % module isotonicity laws
    formula(forall([kleene(x),kleene(y),slat(p)],
      implies(kleq(x,y),lleq(scalar(x,p),scalar(y,p))))).
    formula(forall([kleene(x),slat(p),slat(q)],
      implies(lleq(p,q),lleq(scalar(x,p),scalar(x,q))))).

  % divergence axioms
    formula(forall([kleene(x)],lleq(nabla(x),scalar(x,nabla(x))))).
    formula(forall([kleene(x),slat(p),slat(q)],
      implies(lleq(p,lplus(scalar(x,p),q)),
              lleq(p,lplus(nabla(x),scalar(star(x),q)))))).

  end_of_list.

  list_of_formulae(conjectures). %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  % to be added

  end_of_list.
  end_problem.
```

B Input for Prover9

```
op(500, infix, "+").
op(490, infix, ";").
op(480, postfix, "*").
op(480, postfix, "'").
op(500, infix, "-").

formulas(sos).
% Kleene algebra axioms  %%%%%%%%%%%%%%%%%%%%%%

    x+y = y+x.
    x+0 = x.
    x+(y+z) = (x+y)+z.
    x;1 = x.
    1;x = x.
    x;(y;z) = (x;y);z.
    0;x = 0.
    x;0 = 0.
    x;(y+z) = x;y+x;z.
    (x+y);z = x;z+y;z.
    x+x = x.

    x <= y <-> x+y = y.

    1+x;x* = x*.
    1+x*;x = x*.
    z+x;y <= y -> x*;z <= y.
    z+y;x <= y -> z;x* <= y.

    x'=x;x'.
    y<=x;y+z -> y<= x'+x*;z.

% Boolean domain axioms %%%%%%%%%%%%%%%%%%%%%%%

    a(x);x = 0.
    a(x;y) = a(x;a(a(y))).
    a(a(x))+a(x)=1.

% divergence %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    d(x;div(x)) = div(x).
    d(y) <= d(x;d(y)) -> d(y) <= div(x).

% added %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    d(x) = a(a(x)).
    lsc(x,y) <-> y;x* <= x;(x+y)*+y.

end_of_list.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

formulas(goals).

    % to be added

end_of_list.
```