Reasoning Automatically about Termination and Refinement

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

Abstract. We present very short mechanised proofs of Bachmair and Dershowitz's termination theorem in different variants of Kleene algebras. Through our experiments we also discover three novel refinement laws for nested infinite loops. Finally, we introduce novel divergence modules in which full automation could be achieved. These structures seem very promising for automated reasoning about infinite behaviours in programs and discrete dynamical systems.

1 Introduction

In 1986, in a fundamental study of commutation, transformation and termination properties of rewrite systems [4], Bachmair and Dershowitz proved the following, by now classical theorem: Termination of the union of two rewrite systems can be separated into termination of the individual systems if one rewrite system quasicommutes over the other. In this context, rewrite systems are considered as abstract reduction systems which are essentially sets of binary relations. Quasi-commutation models a quite general way of rearranging rewrite sequences that subsumes a number of interesting cases. The termination theorem yields a powerful tool for analysing termination of rewrite systems. It also provides a very general transformation and refinement law for programs, reactive and concurrent systems. The proof sketch contained in the original paper informally analyses infinite rewrite sequences.

Motivated by applications in concurrency control, Ernie Cohen posed this termination theorem as a challenge for variants of Kleene algebras at a Dagstuhl Seminar in 2001 [7], conjecturing that it cannot be proved in this setting, and he repeated this challenge at a DIMACS workshop [8]. Nevertheless, a proof in a variant of Kleene algebra was published in 2006 [19], but it is rather indirect and tedious. This is interesting, since statements of similar complexity could recently be proved fully automatically [14, 15]. So, sharpening Cohen's challenge, can Bachmair and Dershowitz's termination theorem be proved automatically in variants of Kleene algebras?

This paper shows that this is indeed the case. But since the previous proof requires a series of lemmas and a direct proof from the axioms of Kleene algebras does not succeed1, new ways must be explored. We therefore perform

---

1 In all experiments, we used a Toshiba Tecra laptop under Linux with an Intel Pentium 1.73GHz processor with 6.5MB memory available.
proof experiments with the automated theorem prover SPASS [3] to “learn” the hypotheses needed for the proofs, by (manually) discarding computationally expensive axioms and by adding potentially useful lemmas in a rather random way. Hypothesis sets that are too weak can often be detected by a counterexample checker, e.g., Mace4 [2].

In the case of Bachmair and Dershowitz’s termination theorem, a seemingly unrelated special unfold law for nested loops that has previously been automatically proved and used in the context of action system refinement is the key to success. With this law added to the set of hypotheses, SPASS returns a proof in less than 5\text{min}. Retranslating this resolution proof into equational reasoning yields a fully formal proof of the termination theorem in essentially one line.

Moreover, a closer inspection of the equational proof reveals a novel refinement law for nested infinite loops, to which the termination theorem is a trivial corollary. This result is immediately applicable to concurrency control and action system refinement [5]. Since Kleene algebras are very abstract, the result holds in a variety of models relevant for programs and transition systems, including relations, traces, paths and languages.

Using this structural information, the automated proofs can easily be replayed in other variants of Kleene algebras to obtain termination and loop refinement theorems also in these settings.

The first variant—von Wright’s demonic refinement algebras [20]—are appropriate for predicate transformer semantics of refinement, but not for relational models. The results obtained are comparable to the previous ones.

The second variant is based on the Kleene modules studied by Leiß [18] and by Ézik and Kuich [12]. We add a Park-style divergence operator that either models infinite iteration in the context of \(\omega\)-regular languages or, in the context of discrete dynamical systems, that part of a state space from which infinite behaviour may arise. With these novel divergence modules, the proof of the loop refinement theorem can even be fully automated without any axiom restrictions and additional hypotheses in SPASS, i.e., without the loop unfold law mentioned.

The results obtained not only further confirm that variants of Kleene algebras in combination with off the shelf automated theorem provers are very useful as light-weight formal methods with heavy-weight automation for analysing programs and reactive systems. They also provide new structural insights related to Bachmair and Dershowitz’s termination theorem, to action system refinement, to \(\omega\)-regular languages and to discrete dynamical systems.

To make our experiment accessible and reproducible, all theory input files, all axiom restrictions, all additional hypotheses and all SPASS outputs are documented in appendices. Inputs will also be made available in TPTP-format at a web-site [1]. Since we are mainly interested in robust results for a formal methods context, we abstain from tuning the prover, avoid extreme running times and do not use extremely powerful hardware. In particular cases, stronger results can certainly be obtained by experts in theorem proving.
2 Kleene Algebras and Omega Algebras

Omega algebras provide an abstract axiomatisation of the objects and operations needed for specifying and proving Bachmair and Dershowitz’s termination theorem. Relation algebras could be used as well, but omega algebras possess fewer operations and simpler axioms, which is beneficial for automated deduction. Omega algebras are simple extension of Kleene algebras that have recently emerged as foundational structures in computing.

An idempotent semiring is a structure \((S, +, \cdot, 0, 1)\) such that \((S, +, 0)\) is a commutative monoid with idempotent addition, \((S, \cdot, 1)\) is a monoid, multiplication distributes over addition from the left and right and 0 is a left and right zero of multiplication. Let \(S\) be a semiring. For all \(x, y, z \in S\), the semiring axioms are

\[
x + (y + z) = (x + y) + z, \\
x + y = y + x, \\
x + 0 = x, \\
x(yz) = (xy)z, \\
x1 = x, \\
x + 0 = x, \\
x(y + z) = xy + xz, \\
(x + y)z = xz + yc, \\
0x = 0, \\
x0 = 0.
\]

As usual in algebra, we stipulate that multiplication binds more strongly than addition, and we omit the multiplication symbol. The relation \(\leq\) defined by \(x \leq y \iff x + y = x\) for all elements \(x, y\) is a partial order. Every idempotent semiring is therefore also a semilattice \((S, \leq)\) with addition as join and the following splitting law holds, which is very useful for automated deduction.

\[
x \leq z \land y \leq z \iff x + y \leq z. \tag{1}
\]

A Kleene algebra [17] is an idempotent semiring \(K\) extended by the star operation (or finite iteration operation) \(\ast : K \to K\) that satisfies, for all \(x, y, z \in K\), the star unfold and star induction axioms

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

An omega algebra [6] is a Kleene algebra \(K\) extended by the omega operation (or infinite iteration operation) \(\omega : K \to K\) that satisfies, for all \(x, y, z \in K\) the omega unfold and omega coinduction axiom

\[
xx^\omega = x^\omega, \\
y \leq z + xy \Rightarrow y \leq x^\omega + x^* z.
\]
Kleene algebras have originally been conceived as algebras of regular events, i.e., to model the operations of addition (or union), multiplication (or concatenation) and star as they arise in language theory.

Kleene algebras also model actions (of a transition system). The constants 0 and 1 model the abortive and the ineffective action. Addition models non-deterministic choice of actions; it therefore has to be idempotent. Multiplication models the composition of actions. The first star unfold axiom, e.g., says that a finite iteration \( x^* \) is either ineffective, whence 1, or it continues after one single \( x \)-action. By the first star induction law, \( x^* \) is the least element with that property. The omega models the strictly infinite iteration of actions. The omega unfold axiom says that prefixing actions \( x \) does not change an infinite iteration \( x^\omega \). The omega coinduction axiom implies that \( x^\omega \) is the greatest element with that property; it also links finite and infinite iteration with respect to some “terminal action" \( z \).

By the star and omega axioms, finite and infinite iteration is expressed within first-order logic with Park-style rules as least and greatest prefixed points (which are also least and greatest fixed points). Operationally, the induction axioms serve as star elimination rules at left-hand sides of equations, the coinduction axioms serve as omega elimination rules at their right-hand sides.

Encodings of omega algebras for theorem proving can be found in Appendix A and at the web-site [1]. It follows from the definition of partial order on Kleene algebras that equational as well as order-based encodings can be used. Experience shows that the order-based encoding, although \( \leq \) is treated as an ordinary predicate symbol for which no specialised inference rules are available, usually yields better results with more complex theorems. We therefore base all our arguments on the order-based encoding.

Some further facts are important for our considerations. First, the unfold axioms can be strengthened to the identities, \( 1 + x x^* = x^* \), \( 1 + x^* x = x^* \) and \( x^\omega = x x^\omega \). Second, all operations are isotone with respect to the ordering \( \leq \), i.e.,

\[
x \leq y \Rightarrow x + z \leq y + z
\]

and likewise for multiplication, star and omega. These properties are also used in the encoding of Kleene algebras for theorem proving.

3 Kleene Algebras and Abstract Reduction Systems

Kleene algebras have a rich model class that includes languages, sets of paths in a graph and sets of program traces. In the present context, however, relational models are our main interest.

So let \( R = 2^{A \times A} \) denote the set of all binary relations over some set \( A \). For all \( r, s \in R \), let \( r + s = r \cup s \), i.e., set union and let \( r \cdot s = \{ (a, b) | \exists c. (a, c) \in r \land (c, b) \in s \} \), i.e., the relational product of \( r \) and \( s \). Let 0 = \( \emptyset \) be the empty relation and let 1 = \( \{(a, a) | a \in A\} \) be the unit relation. Finally, let \( r^* = \bigcup_{i \geq 0} r^i \), where \( r^0 = 1 \) and \( r^{i+1} = r \cdot r^i \). It is easy to see that \( r^* \) models the reflexive transitive closure of \( r \). The following theorem is well-known and easy to verify.
Theorem 3.1. \((R, +, \cdot, 0, 1, *)\) is a Kleene algebra.

It is often called the full relation Kleene algebra over \(A\). Obviously, \(R\) is its maximal element. It follows from basic results of universal algebra that every subalgebra of the full relation Kleene algebra is again a Kleene algebra. See, e.g., [10] for a discussion.

Now each full relation Kleene algebra is complete and, by the Knaster-Tarski theorem, the greatest fixed point of the function \(\lambda y. z + xy\) exists and is equal to \(x^\omega + x^*z\).

Theorem 3.2. \((R, +, \cdot, 0, 1, *, \omega)\) is an omega algebra.

Note, however, that \(x^\omega\) is not necessarily equal to an iteration \(\bigcap_{i \geq 0} x^i \cdot R\), since this would presuppose distributivity of multiplication over arbitrary infima. Nevertheless, \(x^\omega = \bigcap_{i \geq 0} x^i \cdot R\) holds whenever \(A\) is finite. Finally, the following result has been shown (cf. [13] for details).

Proposition 3.3. In every (full) relation semiring, \(r^\omega = 0\) if and only if there are no infinitely ascending \(r\)-chains, that is, if and only if \(r\) terminates.

The analysis of Bachmair and Dershowitz’s termination theorem is entirely based on abstract reduction systems, i.e., it disregards the subterm property which is present in concrete term rewrite systems. Formally, an abstract reduction system is a family \(r_i\) of binary relations on some set \(A\). Every abstract reduction system can therefore be embedded into the full relation omega algebra on \(A\).

Corollary 3.4. Let \(R\) be an abstract reduction system. Then \((R, +, \cdot, 0, 1, *, \omega)\), with the operations defined as before, is a relation omega algebra.

Corollary 3.4 and Proposition 3.3 yield the general justification that termination properties of abstract reduction systems can be analysed in terms of omega algebras.

4 First Proof

Based on the general results from Section 3, we can now abstract the notions of quasicommutation and termination, and the statement of the separation theorem in omega algebra.

We assume that rewrite systems \(x\) and \(y\) are elements of some omega algebra. This is reasonable, since rewrite systems—more precisely abstract reduction systems—are relations, and since relations under union, composition, reflexive transitive closure and infinite iteration together with the empty relation and the unit relation form an omega algebra.

For specifying Bachmair and Dershowitz’s termination theorem, two notions are essential. Let \(x\) and \(y\) be elements of some omega algebra. Then

- \(x\) quasicommutes over \(y\) if \(yx \leq x(x + y)^*\);
- \(x\) terminates if \(x^\omega = 0\).
Termination as absence of infinite $x$-chains has been used by Bachmair and Dershowitz. The termination theorem can now be rephrased as follows.

**Theorem 4.1.** Let $x$ and $y$ be elements of some omega algebra and let $x$ quasi-commute over $y$.

$$(x + y) = 0 \iff x + y = 0.$$ 

It is well known that the right-to-left direction does not require quasicommutation. It can be proved with SPASS in less than 0.13s. The SPASS output of this and all other proofs together with detailed information about restrictions on the axiom set and additional hypotheses used can be found in Appendix E. This information and the axiomatisations from Appendices A, B and C will allow readers to replay all proofs. In this particular case, no restrictions on the axiom set and no additional hypotheses are needed.

A proof of the left-to-right direction of Theorem 4.1 in a full sweep is impossible with the hardware available. Experimenting with different hypothesis sets, as mentioned in the introduction, we can find a proof from a restricted axiom set and with additional hypotheses in about 4min. The key to success is the law

$$(x + y) = y + yx(x + y), \quad (2)$$

which has previously been automatically verified and used in the context of program refinement [14].

An equational proof can be reconstructed from the resolution proofs. For its presentation, the following property is worth mentioning.

$$yx \leq x(x + y) \iff yx \leq x(x + y). \quad (3)$$

The right-to-left direction can be proved in 0.33s without any restrictions. The right-to-left direction took about 27s from a reduced axiom set and an addition hypothesis.

The equational proof of Theorem 4.1 is then a one-liner:

Proof. (of Theorem 4.1)

$$(x + y) = y + yx(x + y) \leq y + x(x + y) = y + x(x + y).$$

The first step is by Equation (2); the second step by quasicommutation and Equation (3); the third step uses the identity $x^* x^w = x^w$. Then

$$(x + y) \leq x^w + x^* y \quad (4)$$

follows by omega coinduction. Now $(x + y) = 0$ immediately follows from $x^w = 0$ and $y^w = 0$.

The converse direction follows—without quasicommutation—from $x \leq x + y$, $y \leq x + y$, isotonicity of omega and the fact that $z^w z^w = z^w$. \qed

This results, obtained from experimenting with SPASS, is certainly surprising. It is much simpler and much more direct than the previous proof in [19] and its circumstantial mechanisation with Prover9 in [14]. However, from the puristic point of view, it is still not satisfactory since it relies on axiom restrictions and an additional lemma.
5 A Novel Loop Refinement Law

A closer look at the equational proof of Theorem 4.1 reveals Equation (4)—a refinement law for infinite loops in the presence of quasicommutation—which is interesting in its own right and of which the termination theorem turns out to be just a special case.

Theorem 5.1. Let \( x \) and \( y \) be elements of some omega algebra and let \( x \) quasi-commute over \( y \). Then

\[
(x + y)\omega = x\omega + x^\ast y\omega. \tag{5}
\]

Proof. For \((x + y)\omega \leq x\omega + x^\ast y\omega\) replay the proof of Theorem 4.1 to equation (4). This direction depends on quasicommutation.

The converse direction follows from \( x \leq x + y, y \leq x + y \), isotonicity of omega, \( x^\ast x\omega = x\omega \) and the fact that \( z\omega z\omega = z\omega \). \( \square \)

The left-to-right direction could be proved in 13s from a restricted axiom set and with additional hypotheses. The right-to-left direction could be proved in 13min35s without any restrictions.

Intuitively, Equation (5) says that a strictly infinite repetition of actions \( x \) or \( y \) chosen non-deterministically can be separated into the non-deterministic execution of a strictly infinite repetition of \( x \)-actions or a finite (possibly empty) repetition of \( x \)-actions followed by a strictly infinite repetition of \( y \)-actions.

The assumption of quasicommutation is quite general; it is implied by other notions of commutation like \( ba \leq a + b^\ast \), where \( a^\ast = aa^* \), \( ba \leq ab \) or \( ba = ab \). All these conditions model meaningful properties of systems: inequalities typically model preference or priority properties whereas equations model independence properties.

Theorem 4.1 now follows from Theorem 5.1 by setting \( x\omega = 0 = y\omega \). The proof with SPASS takes 0.04s without any restrictions or additional hypotheses.

6 Demonic Refinement Algebras

We now provide an alternative proof of a variant of the above loop refinement theorem and of Bachmair and Dershowitz’s termination theorem in another variant of Kleene algebra called demonic refinement algebra [20]. Formally, these are structures \((K, \\infty)\) such that \( K \) is a Kleene algebra without the right zero axiom and the operation \( \\infty \) of strong iteration is axiomatised by the strong unfold, the strong coinduction and the isolation axiom

\[
x^{\infty} = 1 + xx^{\infty}, \quad y \leq z + xy \Rightarrow y \leq x^{\infty}z, \quad x^{\infty} = x^\ast + x^{\infty}0
\]

for all \( x, y, z \in K \). The converse strong unfold law, \( 1 + x^{\infty}x = x^{\infty} \), follows from the axioms and strong iteration is isotone with respect to the ordering.

The particular axioms of demonic refinement algebra can easily be motivated from the predicate transformer model of refinement with infinite iteration, cf. [20]. It has been shown in Back and von Wright’s refinement calculus [5] that
\(x^\infty = x^\infty + x^*\). The same proof trivially holds in demonic refinement algebra. Therefore, strong iteration comprises finite and strictly infinite iteration. It is also immediately obvious that demonic refinement algebras do not capture the relational semantics of programs, since in the presence of the right zero axiom (which is satisfied in relational models), the isolation axiom collapses strong iteration to finite iteration. Therefore, the results of the following section are not directly related to Bachmair and Dershowitz's termination theorem. But as refinement theorems within the refinement calculus they are certainly interesting in their own right.

Also, all theorems of demonic refinement algebra that do not mention strong iterations are also theorems of Kleene algebra.

The code for demonic refinement algebras in SPASS can again be found in Appendix B.

7 Second Proof

In the context of demonic refinement algebras, quasicommutation can of course be written as before. But there are now two different notions of termination. We say that

- \(x\) weakly terminates if \(x^\infty = x^*\);
- \(x\) strongly terminates if \(x^\infty 0 = 0\).

**Lemma 7.1.** In every demonic refinement algebra, strong termination implies weak termination, but the converse need not hold.

The implication of weak termination by strong termination can be shown with SPASS in less than 0.06s without any restrictions or additional hypotheses. For the converse direction, the counterexample generator Mace4 [2] finds a counterexample with three elements. It is presented in Appendix D.

It has already been shown automatically that a variant of Equation (2) holds in demonic refinement algebra [1].

\[(x + y)^\infty = y^\infty + y^*x(x + y)^\infty.\] (6)

Moreover, \(x^*x^\infty = x^\infty\), so that—up to the coinduction step—the equational proof of Theorem 5.1 can be translated into demonic refinement algebra. With the strong coinduction law as a last step, we then obtain the following loop refinement law.

**Theorem 7.2.** Let \(x\) and \(y\) be elements of some demonic refinement algebra and let \(x\) quasicommute over \(y\). Then

\[(x + y)^\infty = x^\infty y^\infty.\] (7)

The right-to-left direction follows immediately from isotonicity and the identity \(x^\infty x^\infty = x^\infty\). The proof from left to right with SPASS takes 0.48s. It reuses the
information of the corresponding proof of Theorem 5.1, i.e., a restricted set of axioms and additional hypotheses. The right-to-left proof takes 11s. It also uses a restricted set of axioms and additional hypotheses.

Due to the two variants of termination, we now obtain two variants of the termination theorem as corollaries.

**Theorem 7.3.** Let \( x \) and \( y \) be elements of some demonic refinement algebra and let \( x \) quasicommute over \( y \). Then

(i) \( x^\infty = x^* \land y^\infty = y^* \Rightarrow (x+y)^\infty = (x+y)^* \);

(ii) \( x^\infty 0 + y^\infty 0 = 0 \Rightarrow (x+y)^\infty 0 = 0 \).

For (i), to show that \( (x+y)^\infty \leq (x+y)^* \) follows from the hypotheses takes 13s. It uses a restricted axiom set and an additional hypothesis. \( (x+y)^* \leq (x+y)^\infty \) can be proved in 0.04s without any restrictions or additional hypotheses. The proof of (ii) takes 0.05s, again without any restrictions or additions.

For the converse direction, we can prove the following statement.

**Theorem 7.4.** Let \( x \) and \( y \) be elements of some demonic refinement algebra. Then

\( (x+y)^\infty 0 = 0 \Rightarrow x^\infty 0 = 0 \land y^\infty 0 = 0 \).

This can be proved in 1min3s without any restrictions or additions.

However, a similar statement for strong termination does not hold. Mace4 yields a counterexample with three elements, which is presented in Appendix D.

### 8 Kleene Modules

We will now show how notions of divergence and termination can be specified in the setting of Kleene modules. Such structures were first studied by Ésik and Kuich [12] and by Leiß [18]. However, an operator for modelling program divergence as a Park-style fixed point operator has, to our knowledge, not yet been given.

Divergence and termination have already been investigated in the context of modal Kleene algebras [10, 9]. Every modal Kleene algebra is also a Kleene module, but not vice versa [11]. So it is necessary to reconsider the notions of termination and divergence. We study them in this more general setting because it simplifies automated deduction and provides some structural insights.

A **Kleene module** [18] \((K, L, :)\) is a two-sorted structure of a Kleene algebra \( K \), a semilattice \( L = (L, +, 0) \) with zero and a scalar product \( : \) from \( K \times L \) to \( L \) that satisfies the following axioms.

\[
\begin{align*}
(x + y)p &= xp + yp, \quad &x(p + q) &= xp + xq, \\
(xy)p &= x(yp), \quad &1p &= p, \quad &x0 &= 0, \\
xp + q &\leq r \Rightarrow x^*q \leq r,
\end{align*}
\]

for all \( x, y \in K \) and \( p, q, r \in L \). We usually omit the scalar product symbol.
A divergence module $(K, L, ;, ^\triangleright)$ is a structure such that $(K, L, ;)$ is a Kleene module and $^\triangleright : K \to L$ a mapping that satisfies the divergence-unfold and the divergence-coinduction axioms

$$x^\triangleright \leq xx^\triangleright, \quad p \leq xp + q \Rightarrow p \leq x^\triangleright + x^\ast q,$$

for all $x \in K$ and $p, q \in L$. Previously, the notation $\nabla(x)$ has been used to denote the divergence of $x$ [9]. Here, we use $x^\triangleright$ to emphasise the similarity to the omega and the strong divergence operator.

This novel definition of divergence modules is very general; it admits at least two interesting interpretations.

Under the first interpretation, $xp$ models the preimage of a set $p$ under a relation $x$ and $x^\triangleright$ models the set of all states from which infinite $x$-sequences may emanate. This divergence set is stable under $x$-actions and it is the greatest set with that property (0 being the lest such set). Then $x$ terminates if $x^\triangleright = 0$. This definition is consistent with the standard set-theoretic notion of Noethericity. In set theory $p - xp$ models the set of $x$-maximal elements of $p$, i.e., the set of those elements from which no further $x$-action is possible. Now $p - xp = 0$, which is equivalent to $p \leq xp$, says that $p$ has no $x$-maximal elements. Then, if $x$ is Noetherian, the empty set 0 is the only element with that property; whence $p \leq ap \Rightarrow p = 0$. See [9] for further discussion in the setting of modal Kleene algebras.

Under the second interpretation, elements of $K$ model finite computations or actions of a program whereas elements of $L$ model infinite ones. The scalar product relates finite and infinite computations in a reasonable way that makes it impossible to compose an infinite element at its right-hand side with any other element. In this setting, the divergence operation maps finite elements to infinite ones. The divergence axioms are precisely typed (or sorted) variants of the unfold and coinduction axioms of omega algebra. So divergence acts as the appropriate omega operator under this interpretation and therefore, $x^\triangleright = 0$ means again absence of infinite iteration.

The two interpretations of omega make this operation very versatile and applicable in different contexts. Beyond modal reasoning, the first interpretation is interesting for the analysis of infinite behaviours in discrete dynamical systems. The second one is more compatible with the definition of $\omega$-regular languages than omega algebra. It seems challenging to obtain a completeness theorem with respect to $\omega$-regular languages from this setting.

Since here, relational models are again admitted, divergence modules capture again Bachmair and Dershowitz’s termination theorem. The correspondence between divergence and termination is even more transparent than for omega algebra. A discussion of the general correspondence between divergence and omega (for modal Kleene algebras) can be found in [13].

As a special case, the carriers of $K$ and $L$ can be the same and $^\triangleright$ becomes an endomorphism. This immediately yields the following fact.

**Proposition 8.1.** Every theorem of divergence modules is a theorem of omega algebra (modulo translation).
The converse direction does, of course, not hold. $x^2 0 = 0$ holds in omega algebra for every element $x$, but a corresponding identity cannot even be written down in divergence modules.

It follows immediately from $(x + y)p = xp + yp$, from $x(p + q) = xp + xq$ and from the definition of the partial ordering that scalar products are isotone in both arguments, i.e.,

$$x \leq y \Rightarrow xp \leq yp \quad \text{and} \quad p \leq q \Rightarrow xq \leq xq.$$ 

This can easily be proved from an equational encoding of divergence modules in SPASS. We will add these properties together with the other isotonicity laws to the prover input files. Since the equational encoding is of no further interest, we neither document this encoding nor the proofs in this paper. An order-based encoding of divergence modules in SPASS can be found in Appendix C.

9 Third Proof and Full Automation

The proofs of variants of the loop refinement theorem and of Bachmair and Dershowitz's termination theorem in Kleene modules is particularly simple and can therefore be automated without any restrictions or additional hypotheses.

Analogously to the previous sections, we can prove a variant of the special unfold law for divergence modules. Since it has not yet been considered, we present it as a lemma.

**Lemma 9.1.** Let $x$ and $y$ be elements of some divergence module. Then

$$(x + y)\nabla = y\nabla + y^* x(x + y)\nabla.$$ (8)

The left-to-right direction takes 1 min 25 s; its converse 2 min 53 s. Both directions are proved from the full axiom set and need no additional hypotheses. This law is interesting in its own right as a refinement law, but we will not need it in further proofs.

The next statement is an analogue to the loop refinement laws Theorem 5.1 and Theorem 7.2 that hold in omega algebras and demonic refinement algebras. We display an equational proof to demonstrate that it is precisely along the lines of omega algebras.

**Theorem 9.2.** Let $x$ and $y$ be elements of some divergence module and let $x$ quasicommute over $y$. Then

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

**Proof.** We calculate

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

The claim then follows from divergence-coinduction. The identity $x^\nabla = x^* x^\nabla$ can easily be verified. ∎
In contrast to previous approaches, this statement can now be proved without any restrictions on the axiom set and without any additional hypotheses with SPASS. The left-to-right direction that uses quasicommutation takes 1min51s. Its converse takes 3min6s.

The fact that proof automation is particularly simple in divergence modules might at first sight seem surprising, since the axiom set of divergence modules is more complex than those of omega algebras and demonic refinement algebras. However, in the two-sorted setting, operations are applied to terms in a more restrictive way, especially the computationally expensive rearrangements due to associativity and commutativity of addition and to the fixed-point laws for finite and infinite iterations that allow self-substitutions can be better controlled. This certainly explains the success of SPASS which can manage sorts in an efficient way.

Theorem 9.2 is quite general and admits many different interpretations beyond rewrite systems. Under the modal interpretation, since the divergence $x^\nabla$ models the basin of non-termination of $x$ in the state space $L$, these basins of non-termination can be separated by Theorem 9.2. This is certainly relevant to the analysis of discrete dynamical systems.

Under the interpretation with finite and infinite actions, it models again loop separation, which is interesting for program verification.

A third variant of Bachmair and Dershowitz’s termination theorem now follows as a corollary, as before.

Theorem 9.3. Let $x$ and $y$ be elements of some divergence module and let $x$ quasicommute over $y$. Then

$$(x + y)^\nabla = 0 \iff x^\nabla + y^\nabla = 0.$$ 

The left-to-right direction takes 3min9s; its converse, assuming Theorem 9.2, 0.11s. The proofs are again obtained from the full axiom set without additional hypotheses.

Since every modal Kleene algebra is a Kleene module, our results holds a fortiori in the former setting. The relationship between the two approaches can intuitively be described as follows. First, instead of defining Kleene modules over a semilattice, we could use a Boolean algebra in the second component. The resulting structures have been investigated in [11]; they are strongly related to dynamic algebras, which are algebraic variants of propositional dynamic logics, and they are Boolean algebras with operators in the sense of Jónsson and Tarski [16]. Second, to obtain modal Kleene algebras, the Boolean algebra can be embedded into the subalgebra bounded by 0 and 1 of the Kleene algebra such that mixed terms between Kleenean and Boolean elements can be written down. The axiom set for modal operators can then be reduced to three simple equations. The precise connection has been set up in [11].

However, due to the more complex axiomatisation of dynamic logics, Boolean algebras with operators and modal Kleene algebras, it cannot be expected to obtain a similar degree of automation. Moreover, due to the abstractness and
universality of variants of Kleene algebras, our results hold in models including relations, program traces, paths and languages.

10 Conclusion

We solved a sharpened variant of Cohen’s challenge by proving Bachmair and Dershowitz’s termination theorem mechanically in variants of Kleene algebras and, in particular, fully automatically in the setting of divergence modules. Through our proof experiments that involve hypothesis selection, we found particularly simple proofs that could be retranslated into fully formal equational proofs with essentially one line of calculation. This is in sharp contrast to the original argument by Bachmair and Dershowitz, a formalisation of which would certainly require several pages, and which seems infeasible to automation.

The concise formalism of Kleene algebras and the discipline of proof enforced in this setting also revealed some structural insight in the setting of Bachmair and Dershowitz’s theorem. Through the equational proof we discovered a new refinement theorem for nested infinite loops to which the termination theorem is a simple corollary.

Using this structural insight, we replayed our proofs in further variants of Kleene algebras and were particularly successful in the newly developed setting of divergence modules.

The simple treatment of the termination theorem in the context of Kleene algebras is based, of course, on a significant amount of abstraction. The formalisation gap between concrete rewrite systems and Kleene algebras is, however, closed once and for all by the well-known proofs that abstract reduction systems form omega algebras or divergence modules. The proofs obtained are short relative to that abstraction.

When starting our proof experiments, we used McCune’s Prover9 [2], but then moved to SPASS when proofs in the two-sorted setting of divergence modules seemed infeasible. We then replayed all proofs for omega algebras and demonic refinement algebras with SPASS, to be able to present more coherent results. Since we do not want to overload this paper, we do not present a comparison of Prover9 and SPASS. Let us only mention that for omega algebras and demonic refinement algebras they were comparable.

From a more general point of view, this work contributes to a series of papers devoted to the automation of first-order algebraic structures with applications in program development, refinement and verification [14, 15]. Our results suggest that the combination of off the shelf automated theorem proving with domain-specific algebras has considerable potential to further establish first-order theorems as a feasible alternative to model checking and interactive proof assistants. Due to the abstraction and universality provided by the algebras, we believe that light-weight formal methods with heavy-weight automation can be obtained.

This line of work also leads to interesting research questions in automated deduction. A first strand is the integration and implementation of solvers and
decision procedures for concrete data types as they arise in verification scenarios, e.g., arithmetics, lists, queues, arrays in automated theorem provers. A second strand is the implementation of order-based reasoning through ordered chaining calculi. Order-based reasoning often highly advantageous for automated algebraic proofs but rather neglected by the theorem proving community. A third strand is the development of focused inference rules for the algebras under consideration, which would further help to guide proof search and allow one to prove relevant theorems of even greater complexity. Finally our algebraic approach provides challenging benchmarks for first-order theorem provers that are both computationally hard and practically relevant. We will therefore make all inputs available in TPTP-format [1].

While preparing the final version of this paper, Peter Höfner and the author were even able to push some of the results from this paper a step further. Bachmair and Dershowitz’s termination theorem (Theorem 4.1) could now be proved in some minutes; the loop refinement theorem in demonic refinement algebra (Theorem 7.2) could be proved in a couple of hours without any axiom restrictions or additional lemmas. These unexpected results were obtained by running Prover9 with an equational axiomatisation of omega algebras and demonic refinement algebras. These results are documented at the web-site [1]. They are in contrast to our previous experience that an order-based approach work better with more complex theorems. Further consideration of these novel results and a comparison between different approaches is planned for an extended version of this paper.

References

A Omega Algebras in SPASS

begin_problem(ka).

list_of_descriptions.
name({'Kleene Algebras'}). author({'Georg Struth'}). status(satisfiable).

description({'Axioms for Kleene algebras, some derived properties,
some lemmas'}). end_of_list.

list_of_symbols.
functions[(plus,2),(times,2),(zero,0),(one,0),(star,1),(omega,1)].
predicates[(leq,2)]. end_of_list.

list_of_formulae(axioms).

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

% multiplicative monoid
 formula(forall([x,y,z],equal(times(times(x,y),z),times(x,times(y,z))))).
 formula(forall([x],equal(times(x,one),x))).
 formula(forall([x],equal(times(one,x),x))).

% distributivity laws
 formula(forall([x,y,z],
   equal(times(x,plus(y,z)),plus(times(x,y),times(x,z))))).
 formula(forall([x,y,z],
   equal(times(plus(x,y),z),plus(times(star(x),y),times(y,z))))).

% zero axioms
 formula(forall([x],equal(times(x,zero),zero))).
 formula(forall([x],equal(times(zero,x),zero))).

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

% star axioms
 formula(forall([x],equal(star(x),plus(one,times(x,star(x)))))).
 formula(forall([x],equal(star(x),plus(one,times(star(x)),x)))).
 formula(forall([x,y,z],
   implies(leq(plus(y,times(x,z)),z),leq(times(star(x),y),z)))).
formula(forall([x,y,z],
    implies(leq(plus(y,times(z,x)),z),leq(times(y,star(x)),z)))).

% omega axioms
formula(forall([x],equal(omega(x),times(x,omega(x))))).
formula(forall([x,y,z],
    implies(leq(z,plus(y,times(x,z))),leq(z,plus(omega(x),times(star(x),y))))).

% isotonicity laws
formula(forall([x,y,z],implies(leq(x,y),leq(plus(z,x),plus(z,y))))).
formula(forall([x,y,z],implies(leq(x,y),leq(plus(x,z),plus(y,z))))).
formula(forall([x,y,z],implies(leq(x,y),leq(times(z,x),times(z,y))))).
formula(forall([x,y,z],implies(leq(x,y),leq(times(x,z),times(y,z))))).
formula(forall([x,y],implies(leq(x,y),leq(star(x),star(y))))).
formula(forall([x,y],implies(leq(x,y),leq(omega(x),omega(y))))).

% splitting law
formula(forall([x,y,z],
    equiv(leq(plus(x,y),z),and(leq(x,z),leq(y,z))))).

% additional laws
formula(forall([x,y,z],
    equal(omega(plus(x,y)),
        plus(omega(y),times(star(y),times(x,omega(plus(x,y)))))))).

end_of_list.

list_of_formulae(conjectures).

% this example presents bachmair dershowitz's termination theorem
% using the additional law above
formula(forall([x,y],implies(
    and(leq(times(y,x),times(x,star(plus(x,y)))),
        equal(omega(x),zero),
        equal(omega(y),zero)
    ),
    leq(omega(plus(x,y)),zero)
))).

end_of_list.
end_problem.

B  Demonic Refinement Algebras in SPASS

begin_problem(dra).
list_of_descriptions.
name({*Demonic Refinement Algebras*}).
author({*Georg Struth*}).
description({*Axioms for demonic refinement algebras, some derived properties*}).
end_of_list.

list_of_symbols.
  functions[{(plus,2),(times,2),(zero,0),(one,0),(star,1),(infty,1)].
predicates[{leq,2}].
end_of_list.

list_of_formulae(axioms).
% additive monoid
  formula(forall([x,y,z],
    equal(plus(plus(x,y),z),plus(x,plus(y,z))))).
  formula(forall([x],equal(plus(x,zero),x))).
  formula(forall([x,y],equal(plus(x,y),plus(y,x)))).
% multiplicative monoid
  formula(forall([x,y,z],equal(times(times(x,y),z),times(x,times(y,z))))).
  formula(forall([x],equal(times(x,one),x))).
  formula(forall([x],equal(times(one,x),x))).
% distributivity laws
  formula(forall([x,y,z],
    equal(times(x,plus(y,z)),plus(times(x,y),times(x,z))))).
  formula(forall([x,y,z],
    equal(times(plus(x,y),z),plus(times(x,z),times(y,z))))).
% zero axiom
  formula(forall([x],equal(times(zero,x),zero))).
% preorder axioms
  formula(forall([x],leq(x,x))).
  formula(forall([x,y,z],implies(and(leq(x,y),leq(y,z)),leq(x,z)))).
% star axioms
  formula(forall([x],equal(star(x),plus(one,times(x,star(x))))).
  formula(forall([x],equal(star(x),plus(one,times(star(x),x))))).
  formula(forall([x,y,z],implies(leq(plus(y,times(x,z)),z),leq(times(star(x),y),z)))).
  formula(forall([x,y,z],implies(leq(plus(y,times(z,x)),z),leq(times(y,star(x)),z)))).
% infty axioms
  formula(forall([x],equal(infty(x),plus(one,times(x,infty(x)))))).
\[
\forall x, \text{equal}(\infty(x), plus(\text{one}, times(\infty(x), x))).
\]

\[
\forall x, y, z, \text{implies}(\text{leq}(z, plus(y, times(x, z))), \text{leq}(z, times(\infty(x), y))).
\]

\[
\forall x, y, z, \text{equal}(\infty(x), plus(\text{star}(x), times(\infty(x), \text{zero}))).
\]

\[
\% \text{ isotonicity laws}
\]

\[
\forall x, y, z, \text{implies}(\text{leq}(x, y), \text{leq}(\text{plus}(z, x), \text{plus}(z, y))).
\]

\[
\forall x, y, z, \text{implies}(\text{leq}(x, y), \text{leq}(\text{plus}(x, z), \text{plus}(y, z))).
\]

\[
\forall x, y, z, \text{implies}(\text{leq}(x, y), \text{leq}(\text{times}(x, z), \text{times}(y, z))).
\]

\[
\forall x, y, z, \text{implies}(\text{leq}(x, y), \text{leq}(\text{times}(x, z), \text{times}(y, z))).
\]

\[
\forall x, x, \text{implies}(\text{leq}(x, y), \text{leq}(\text{star}(x), \text{star}(y))).
\]

\[
\forall x, y, \text{implies}(\text{leq}(x, y), \text{leq}(\infty(x), \infty(y))).
\]

\[
\% \text{ splitting law}
\]

\[
\forall x, y, z, \text{equiv}(\text{leq}(\text{plus}(x, y), z), \text{and}(\text{leq}(x, z), \text{leq}(y, z))).
\]

end_of_list.

list_of_formulae(conjectures).

\% to be added

diso

dend.

eend_problem.

C Divergence Modules in SPASS

begin_problem(modules).

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

dend.

eend_list.

eend_list.

eend_list.

list_of_symbols.
functions[[kplus,2],[ktimes,2],[k0,0],[k1,0],[star,1],
(plus,2),(0,0),
(scalar,2),
(nabla,1)].
predicates[[kleq,2],[l0eq,2]].
sorts[kleene,slat].
ediso

edend.

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(10).
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(x,z)),z),kleq(ktimes(star(x),y),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),kleene(z)],
  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,10),10))).
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.

D Counterexamples

– Lemma 7.1: Weak termination does not imply strong termination.

interpretation( 3, [number=1, seconds=0], [
    function(c1, [ 2 ]),
    function('(_), [ 1, 2, 2 ]),
    function(*(_), [ 1, 1, 2 ]),
    function(+(_,_), [
        0, 1, 2,
        1, 1, 2,
        2, 2, 2 ]),
    function(;(_,_), [
        0, 0, 0,
        0, 1, 2,
        2, 2, 2 ])
  )).

– \((x + y)^\infty = (x + y)^*\) does not imply \(x^\infty x^* \land y^\infty = y^*\).

interpretation( 3, [number=1, seconds=0], [
    function(c1, [ 1 ]),
    function(c2, [ 2 ]),
    function('(_), [ 1, 2, 2 ]),
    function(*(_), [ 1, 1, 2 ]),
    function(+(_,_), [ 
E  SPASS Proof Outputs

As a default, SPASS presents some proof data, but, in order to speed up proof-search, a resolution proof is only optionally displayed. In all experiments, we used a Toshiba Tecra laptop under Linux with an Intel Pentium 1.73GHz processor with 6.5MB memory available.

All additional hypotheses mentioned below have been previously verified, many of them with Prover9 [2]. Information about these proofs can be found at a web-site [1].

- Theorem 4.1.
  - right-to-left
    1. SPASS output:
       SPASS V 3.0c
       SPASS beiseite: Proof found.
       Problem: ka/ka.dfg
       SPASS derived 871 clauses, backtracked 0 clauses and kept 348 clauses.
       SPASS allocated 695 KBytes.
       SPASS spent 0:00:00.13 on the problem.
       0:00:00.00 for the input.
       0:00:00.01 for the FLOTTER CNF translation.
       0:00:00.01 for inferences.
       0:00:00.00 for the backtracking.
       0:00:00.07 for the reduction.

  2. Axiom set: full
  3. Additional Hypotheses: none

  - left-to-right
    1. SPASS output:
       SPASS V 3.0c
       SPASS beiseite: Proof found.
       Problem: ka/ka.dfg
       SPASS derived 70555 clauses, backtracked 0 clauses and kept 29344 clauses.
       SPASS allocated 17935 KBytes.
       SPASS spent 0:03:59.60 on the problem.
       0:00:00.00 for the input.
       0:00:00.00 for the FLOTTER CNF translation.
       0:00:01.59 for inferences.
0:00:00.00 for the backtracking.  
0:03:55.47 for the reduction.

2. Axiom set: left unit of addition, associativity of multiplication, right unit of multiplication, right zero, reflexivity, transitivity, omega coinduction, isotonicity of addition and multiplication

3. Additional hypotheses: Equation 2, $x^*x\omega = x\omega$, $x^{*\omega} = x\omega$, $yx \leq xz \Rightarrow y^*x \leq x^*z$

- Equation 3

• right-to-left
1. SPASS output:

SPASS V 3.0c
SPASS beiseite: Proof found.  
Problem: ka/ka.dfg  
SPASS derived 3027 clauses, backtracked 0 clauses and kept 928 clauses.  
SPASS allocated 1015 KBytes.  
SPASS spent 0:00:00.33 on the problem. 
0:00:00.00 for the input.  
0:00:00.01 for the FLOTTER CNF translation.  
0:00:00.03 for inferences.  
0:00:00.00 for the backtracking.  
0:00:00.23 for the reduction.

2. Axiom set: full
3. Additional hypotheses: none

• left-to-right
1. SPASS output:

SPASS V 3.0c
SPASS beiseite: Proof found.  
Problem: ka/ka.dfg  
SPASS derived 13284 clauses, backtracked 0 clauses and kept 9062 clauses.  
SPASS allocated 6267 KBytes.  
SPASS spent 0:00:26.89 on the problem.  
0:00:00.00 for the input.  
0:00:00.01 for the FLOTTER CNF translation.  
0:00:00.03 for inferences.  
0:00:00.00 for the backtracking.  
0:00:00.45 for the reduction.

2. Axiom set: associativity of multiplication, right unit of multiplication, reflexivity and transitivity, $1+xx^* \leq x^*$, $y+xz \leq z \Rightarrow x^*y \leq z$, isotonicity of multiplication, the splitting law (1)

3. Additional hypotheses: $x^*x^* = x^*$

• left-to-right
1. SPASS output
SPASS V 3.0c
SPASS beiseite: Proof found.
Problem: ka/ka.dfg
SPASS derived 12509 clauses, backtracked 0 clauses and kept 4896 clauses.
SPASS allocated 8394 KBytes.
SPASS spent 0:00:12.52 on the problem.
0:00:00.00 for the input.
0:00:00.00 for the FLOTTER CNF translation.
0:00:00.67 for inferences.
0:00:00.00 for the backtracking.
0:00:11.48 for the reduction.

2. Axiom set: associativity of multiplication, reflexivity, omega coinduction, isotonicity of addition and multiplication
3. Additional hypotheses: Equation (2), $x^{**} = x^*$, $x^* x^\omega = x^\omega$
   - right-to-left
1. SPASS output:
   SPASS V 3.0c
   SPASS beiseite: Proof found.
   Problem: ka/ka.dfg
   SPASS derived 145602 clauses, backtracked 0 clauses and kept 38679 clauses.
   SPASS allocated 19666 KBytes.
   SPASS spent 0:13:34.56 on the problem.
   0:00:00.00 for the input.
   0:00:00.01 for the FLOTTER CNF translation.
   0:00:01.95 for inferences.
   0:00:00.00 for the backtracking.
   0:13:30.03 for the reduction.

2. Axiom set: full
3. Additional hypotheses: none
   - Theorem 4.1 from Theorem 5.1
1. SPASS output:
   SPASS V 3.0c
   SPASS beiseite: Proof found.
   Problem: ka/ka.dfg
   SPASS derived 29 clauses, backtracked 0 clauses and kept 36 clauses.
   SPASS allocated 559 KBytes.
   SPASS spent 0:00:00.04 on the problem.
   0:00:00.00 for the input.
   0:00:00.01 for the FLOTTER CNF translation.
   0:00:00.00 for inferences.
   0:00:00.00 for the backtracking.
   0:00:00.00 for the reduction.

2. Axiom set: full
3. Additional hypotheses: none
- Lemma 7.1
  1. SPASS output:
     SPASS V 3.0c
     SPASS beiseite: Proof found.
     Problem: dra/dra.dfg
     SPASS derived 88 clauses, backtracked 0 clauses and kept 56 clauses.
     SPASS allocated 553 KBytes.
     SPASS spent 0:00:00.06 on the problem.
     0:00:00.00 for the input.
     0:00:00.03 for the FLOTTER CNF translation.
     0:00:00.00 for inferences.
     0:00:00.00 for the backtracking.
     0:00:00.00 for the reduction.

2. Axiom set: full
3. Additional hypotheses: none
- Theorem 7.2
  • left-to-right
    1. SPASS output:
       SPASS V 3.0c
       SPASS beiseite: Proof found.
       Problem: dra/dra.dfg
       SPASS derived 1356 clauses, backtracked 0 clauses and kept 788 clauses.
       SPASS allocated 1628 KBytes.
       SPASS spent 0:00:00.48 on the problem.
       0:00:00.00 for the input.
       0:00:00.00 for the FLOTTER CNF translation.
       0:00:00.05 for inferences.
       0:00:00.00 for the backtracking.
       0:00:00.37 for the reduction.

2. Axiom set: analogous to the proof of Theorem 5.1
3. Additional hypotheses: analogous to the proof of Theorem 5.1
  • right-to-left
    1. SPASS output:
       SPASS V 3.0c
       SPASS beiseite: Proof found.
       Problem: dra/dra.dfg
       SPASS derived 8497 clauses, backtracked 0 clauses and kept 5622 clauses.
       SPASS allocated 4324 KBytes.
       SPASS spent 0:00:10.24 on the problem.
       0:00:00.00 for the input.
       0:00:00.00 for the FLOTTER CNF translation.
       0:00:00.14 for inferences.
       0:00:00.00 for the backtracking.
       0:00:09.98 for the reduction.
2. Axiom set: reflexivity, transitivity, splitting, isotonicity of multiplication and strong iteration

3. Additional hypotheses: $x^\infty x^\infty = x^\infty$

- Theorem 7.3(i)
  - $(x + y)^\infty \leq (x + y)^*$ follows from hypotheses

1. SPASS output:
   
   SPASS V 3.0c
   SPASS beiseite: Proof found.
   Problem: dra/dra.dfg
   SPASS derived 10363 clauses, backtracked 0 clauses and kept 6520 clauses.
   SPASS allocated 4478 KB\text{Bytes}.
   SPASS spent 0:00:12.88 on the problem.
   0:00:00.00 for the input.
   0:00:00.01 for the FLOTTER CNF translation.
   0:00:00.17 for inferences.
   0:00:00.00 for the backtracking.
   0:00:12.54 for the reduction.

2. Axiom set: reflexivity, transitivity, left star unfold and star induction, isotonicity of multiplication and star

3. Additional hypotheses: $x^* x^* = x^*$

- $(x + y)^* \leq (x + y)^\infty$

1. SPASS output:

   SPASS V 3.0c
   SPASS beiseite: Proof found.
   Problem: dra/dra2.dfg
   SPASS derived 123 clauses, backtracked 0 clauses and kept 62 clauses.
   SPASS allocated 555 KB\text{Bytes}.
   SPASS spent 0:00:00.04 on the problem.
   0:00:00.00 for the input.
   0:00:00.01 for the FLOTTER CNF translation.
   0:00:00.00 for inferences.
   0:00:00.00 for the backtracking.
   0:00:00.00 for the reduction.

2. Axiom set: full

3. Additional hypotheses: none

- Theorem 7.3(ii)

1. SPASS output:

   SPASS V 3.0c
   SPASS beiseite: Proof found.
   Problem: dra/dra.dfg
   SPASS derived 355 clauses, backtracked 0 clauses and kept 116 clauses.
   SPASS allocated 564 KB\text{Bytes}.
   SPASS spent 0:00:00.05 on the problem.
2. Axiom set: full
3. Additional hypotheses: none
   - Theorem 7.4
     1. SPASS output:
        SPASS V 3.0c
        SPASS beiseite: Proof found.
        Problem: dra/dra2.dfg
        SPASS derived 50049 clauses, backtracked 0 clauses and kept 15302 clauses.
        SPASS allocated 6697 KBytes.
        SPASS spent 0:01:03.03 on the problem.
        0:00:00.00 for the input.
        0:00:00.01 for the FLOTTER CNF translation.
        0:00:00.01 for inferences.
        0:00:00.00 for the backtracking.
        0:00:00.01 for the reduction.

2. Axiom set: full
3. Additional hypotheses: none
   - Theorem 9.1
     • left-to-right
       1. SPASS output:
          SPASS V 3.0c
          SPASS beiseite: Proof found.
          Problem: modules/module.dfg
          SPASS derived 16669 clauses, backtracked 0 clauses and kept 5805 clauses.
          SPASS allocated 18087 KBytes.
          SPASS spent 0:01:24.40 on the problem.
          0:00:00.00 for the input.
          0:00:00.01 for the FLOTTER CNF translation.
          0:00:00.54 for inferences.
          0:00:00.00 for the backtracking.
          0:01:01.94 for the reduction.

2. Axiom set: full
3. Additional hypotheses: none
   • right-to-left
  1. SPASS output:
     SPASS V 3.0c
     SPASS beiseite: Proof found.
     Problem: modules/module.dfg
     SPASS derived 20744 clauses, backtracked 0 clauses and kept 7271 clauses.
SPASS allocated 23529 KBytes.
SPASS spent 0:02:52.43 on the problem.
0:00:00.00 for the input.
0:00:00.02 for the FLÖTTER CNF translation.
0:00:00.40 for inferences.
0:00:00.00 for the backtracking.
0:02:51.33 for the reduction.

2. Axiom set: full
3. Additional hypotheses: none
– Theorem 9.2
  • left-to-right
  1. SPASS output:
     SPASS beiseite: Proof found.
     Problem: modules/module.dfg
     SPASS derived 21099 clauses, backtracked 0 clauses and kept 7390 clauses.
     SPASS allocated 5687 KBytes.
     SPASS spent 0:01:50.65 on the problem.
     0:00:00.00 for the input.
     0:00:00.02 for the FLÖTTER CNF translation.
     0:00:00.35 for inferences.
     0:00:00.00 for the backtracking.
     0:01:49.81 for the reduction.

2. Axiom set: full
3. Additional hypotheses: none
• right-to-left
  1. SPASS output:
     SPASS V 3.0c
     SPASS beiseite: Proof found.
     Problem: modules/module.dfg
     SPASS derived 20974 clauses, backtracked 0 clauses and kept 7325 clauses.
     SPASS allocated 23942 KBytes.
     SPASS spent 0:03:05.72 on the problem.
     0:00:00.00 for the input.
     0:00:00.02 for the FLÖTTER CNF translation.
     0:00:00.40 for inferences.
     0:00:00.00 for the backtracking.
     0:03:04.63 for the reduction.

2. Axiom set: full
3. Additional hypotheses: none
– Theorem 9.3
  • left-to-right
  1. SPASS output:
     SPASS V 3.0c
     SPASS beiseite: Proof found.
     Problem: modules/module.dfg
SPASS derived 21212 clauses, backtracked 0 clauses and kept 7486 clauses.
SPASS allocated 24095 KBytes.
SPASS spent 0:03:08.25 on the problem.
    0:00:00.00 for the input.
    0:00:00.02 for the FLOTTER CNF translation.
    0:00:00.41 for inferences.
    0:00:00.00 for the backtracking.
    0:03:07.13 for the reduction.

2. Axiom set: full
3. Additional hypotheses: none
   • right-to-left (from Theorem 9.2)

1. SPASS output:
   SPASS V 3.0c
   SPASS beiseite: Proof found.
   Problem: modules/module.dfg
   SPASS derived 161 clauses, backtracked 0 clauses and kept 137 clauses.
   SPASS allocated 738 KBytes.
   SPASS spent 0:00:00.11 on the problem.
    0:00:00.00 for the input.
    0:00:00.02 for the FLOTTER CNF translation.
    0:00:00.00 for inferences.
    0:00:00.00 for the backtracking.
    0:00:00.01 for the reduction.

2. Axiom set: full
3. Additional hypotheses: none