AMAI manuscript No.
(will be inserted by the editor)

Automated Verification of Refinement Laws

Peter Höfner · Georg Struth · Geoff Sutcliffe

Received: date / Accepted: date

Abstract Demonic refinement algebras are variants of Kleene algebras. Introduced by von Wright as a light-weight variant of the refinement calculus, their intended semantics are positively disjunctive predicate transformers, and their calculus is entirely within first-order equational logic. So, for the first time, off-the-shelf automated theorem proving (ATP) becomes available for refinement proofs. We used ATP to verify a toolkit of basic refinement laws. Based on this toolkit, we then verified two classical complex refinement laws for action systems by ATP: a data refinement law and Back's atomicity refinement law. We also present a refinement law for infinite loops that has been discovered through automated analysis. Our proof experiments not only demonstrate that refinement can effectively be automated, they also compare eleven different ATP systems and suggest that program verification with variants of Kleene algebras yields interesting theorem proving benchmarks. Finally, we apply hypothesis learning techniques that seem indispensable for automating more complex proofs.

Keywords Refinement calculus · Kleene algebras · Automated deduction · Action systems

1 Introduction

The combination of variants of Kleene algebras with off-the-shelf automated theorem proving (ATP) systems has the potential to augment light-weight algebraic formal methods with heavy-weight automation. The methods are light-weight because they focus on specific analysis tasks, and the automation is heavy-weight because ATP

P. Höfner
Institute of Computer Science, University of Augsburg, Germany
E-mail: hoefner@informatik.uni-augsburg.de

G. Struth
Department of Computer Science, University of Sheffield, UK
E-mail: G.Struth@dcs.shef.ac.uk

G. Sutcliffe
Department of Computer Science, University of Miami, US
E-mail: geoff@cs.miami.edu

is essentially a push-button technology that requires little user interaction. Recent work already investigated the automation of computational logics [14], of relational reasoning [17], of rewriting and termination analysis [32,33], and of hybrid system analysis [13]. It has also been shown that ATP can be very helpful in developing consistent irredundant specifications and axiomatisations of algebraic theories [11,10].

This paper investigates the automated verification of refinement laws. In contrast to the trace- or relation-based models that were so far automatically and abstractly analysed through Kleene algebras, the refinement calculus of Back and von Wright [3] uses a predicate transformer semantics that requires yet another algebra: *Demonic refinement algebras*. They were introduced in two seminal papers by von Wright [42, 43] and have been extensively studied since.

Our main results are as follows. First, we develop a formally verified toolkit of basic refinement laws within demonic refinement algebra. It contains laws, for instance, for deconstructing and reconstructing concurrency, for simulation and for loop refinement. Second, we then use this toolkit for automatically verifying two complex laws for data refinement and atomicity refinement of action systems. Third, we present a new refinement law for infinite loops and two novel termination theorems that have been discovered through ATP experiments.

For verifying the toolkit of basic refinement laws we evaluated eleven of the most successful ATP systems [35] through the standard TPTP input syntax [38] and generally following the established policy for the CADE ATP System Competition [37,25]. Our experiments reveal significant differences in performance. While the three top ATP systems are able to prove all the basic laws, the other systems can prove only less complex statements. The overall best ATP system is McCune's Prover9 [22], validating the less informed choice made for previous experiments [15,32]. We therefore used Prover9 for the more advanced experiments, but the results show that there are several other ATP systems that are well suited to these tasks, and they may be preferred by some users due to differences in their performances and input/output languages. We use the Prover9 syntax for presenting input files within the paper, because it makes algebraic expressions more humanly readable (it can automatically be translated to and from TPTP syntax). Finally, we used the model generator Mace4 [22] in our analysis for finding counterexamples and for hypothesis learning.

The overall outcome of our experiments is positive: the development and analysis of complex refinement laws with off-the-shelf ATP technology is nowadays possible. This is in contrast to the prevailing paradigm that special purpose ATP systems are needed for theorem proving with complex algebraic structures. Most basic refinement laws could be proved from the demonic refinement algebra axioms in a few seconds. Some machine proofs even allowed us to simplify previous manual proofs or to generalise known theorems. Only the atomicity refinement theorem required the introduction of an intermediate lemma. Strictly speaking, therefore, we obtained an automated proof, but a proof search that uses solely the axioms of demonic refinement algebras did not succeed. But even von Wright's manual proof in demonic refinement algebras is almost two pages long; the proof search involved and the complexity of the hypotheses used in the automated proof is substantial. And our automated analysis revealed some errors in his proof.

The results of this paper seem interesting for various reasons: In the context of refinement, they introduce ATP as a new technology that further underpins the usefulness of algebraic approaches and provides a promising alternative to existing formalisations built on model checking or interactive theorem proving. In the context of

automated deduction, they provide useful benchmarks for ATP systems and open up a new application domain in program verification. In the context of formal methods, our work paves the way for a new integration of computational algebras with automated deduction which could make those methods more automated and more user friendly.

For the sake of readability we usually do not display the input/output files and machine proofs. They can all be found at a website [16]. Selected theorems and a number of challenge problems from this paper will be part of the TPTP library from 2009. A brief introduction to the refinement calculus is included to make the paper more accessible for the ATP community.

2 Demonic Refinement Algebras

The idea of stepwise refining specifications to programs is certainly intriguing: Provide software developers with a calculus of meaningful refinement laws and they will deliver code that is dependable, modular and optimised. Since the pioneering work of Dijkstra, Hoare and Back, refinement has matured into an established research area that forms a cornerstone of popular formal software development methods such as Alloy, *B*, *VDM* or *Z*. The development of refinement calculi over the decades can be characterised as an *algebraic turn* that led to considerable simplifications and abstractions. An important step was Back and von Wright's lattice-based approach that abstractly characterises the two fundamental models of refinement: binary relations and predicate transformers [3]. This approach is, however, essentially higher-order; a serious obstacle against automation. More recently, in two seminal papers [42, 43], von Wright reconstructed a substantial part of the refinement calculus in a variant of Kleene algebra. These *demonic refinement algebras* are entirely within first-order equational logic which, for the first time, paves the way to automated deduction. But this intriguing potential has so far not been explored.

The refinement calculi considered in this paper deal with imperative programs that may or may not be executable. The traditional way of reasoning about such a program is to assign preconditions and postconditions and to formulate a notion of (total) program correctness; a program satisfying the precondition will satisfy the postcondition after its execution. Roughly, the relation of stepwise refinement is then a correctness-preserving transformation between programs; the refined program can always safely be substituted in any context without observably different behaviour. Typical examples of such refinements are loop-transformations, program optimisations and data refinements that relate abstract data types such as sets with concrete data types such as arrays, lists or heaps. Typical effects of refinements are elimination of nondeterminism and weakening of preconditions.

In the context of program correctness, the weakest precondition for a given program and a given postcondition is very useful, since it models the most general setting under which a program can establish a desired result. Weakest preconditions can be understood as mappings from postconditions to preconditions, hence from predicates to predicates, that are parametrised by programs. These mappings are therefore called *predicate transformers*. An algebraic predicate transformer calculus reflecting the imperative programming constructs can then be given. The refinement relation turns into an inclusion relation between predicates. The most important constructs are the sequential composition of programs, the choice between programs (needed in specifications as well as for modelling conditionals), and the iteration of a program (needed for

while-loops). Choices can be resolved in two fundamentally different ways: in an *angelic* setting, only one branch of a program is required to satisfy the postcondition, whereas in a *demonic* setting, all branches must fulfil this requirement. Hence demonic choices assume that the worst possible alternative is taken and this view is most compatible with refinement: it must be ensured that all possible program executions respect the specification.

In Back and von Wright's textbook [3], a higher-order algebra of demonic predicate transformers has been developed. It has later been reconstructed in the first-order framework of demonic refinement algebras by von Wright [42, 43]. These publications serve as an excellent source of further information.

Demonic refinement algebras are variants of Kleene algebras [20], which themselves are based on idempotent semirings. We will follow these steps for their introduction.

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. These axioms, except the right zero axiom, are presented as input to ATP systems in Section 3. As usual in algebra, we stipulate that multiplication binds more strongly than addition, and we omit the multiplication symbol. In the context of refinement, elements of S denote actions of a program (in a relational semantics), multiplication denotes sequential composition, addition denotes angelic nondeterministic choice, 0 denotes the destructive action, and 1 the ineffective action.

The relation $\leq$ defined by $x \leq y \Leftrightarrow x + y = y$ for all elements x, y of an idempotent semiring is a partial order. It is (up to isomorphism) the only order with least element 0 and for which addition and multiplication are isotone in both arguments. Therefore, every idempotent semiring is also a semilattice $(S, \leq)$ with addition as join and, for all $x, y, z \in S$,

$$x + y \leq z \Leftrightarrow x \leq z \wedge y \leq z. \tag{1}$$

This law allows a case analysis of sums on left-hand sides of inequalities. There is no similar law for right-hand sides. In the context of refinement, $\geq$ corresponds to the refinement ordering.

To model recursive behaviour, an operation of finite iteration can be added. A *Kleene algebra* is a structure $(K, ^*)$ such that K is an idempotent semiring and *star* * is a unary operation axiomatised by the *star unfold* and *star induction* axioms

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

for all $x, y, z \in K$. This axiomatises finite iterations within first-order equational logic as least prefixed points (which are also least fixed points of the expressions $\mu y.xy + z$ and $\mu y.yx + z$).

By the first star unfold axiom, an 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. This form of iteration proceeds from left to right through a sequence of actions. The second star unfold and star induction law correspond to right-to-left iteration. The star is also isotone with respect to the ordering and the star unfold axioms can be strengthened to equations.

In order to capture the (positively disjunctive) predicate transformer semantics of Back and von Wright's refinement calculus [3], some adaptations must be made. First, the right zero axiom $x0 = 0$ must be dropped because it seems unreasonable to assume

that an action 0 could succeed an infinite action x. Second, a *strong iteration* operation that encompasses finite and infinite iteration must be added. This is appropriate for modelling a loop possessed by a demon, which may be finite or infinite. A discussion of the relevance of the particular axioms, their semantics, and their relationship to the refinement calculus, can be found in von Wright's articles [42, 43]. The resulting structures are particularly suitable for modelling action system refinement in situations where the user has no control over loop termination [4].

Formally, a *demonic refinement algebra* is a structure $(K, ^\infty)$ such that K is a Kleene algebra without the right zero axiom, and *strong iteration* $^\infty$ is a unary operation axiomatised by the *strong unfold* and the *strong coinduction* axioms

$$x^\infty = 1 + xx^\infty, \qquad\qquad y \leq z + xy \Rightarrow y \leq x^\infty z,$$

and linked with the star by the *isolation* axiom

$$x^\infty = x^* + x^\infty 0,$$

for all $x, y, z \in K$. The converse strong unfold law, $1 + x^\infty x = x^\infty$, follows from these axioms. Strong iteration is isotone with respect to the ordering. The axioms of demonic refinement algebras are explicitly listed in Section 3.

The isolation axiom excludes relational models. In these models, $x0$ is always zero, so $x^\infty = x^*$ and therefore meaningless. This is important for our considerations in Section 9.

Unfortunately, different notation is used in different communities. The algebraic community in the tradition of Salomaa, Conway and Kozen uses the notation of this paper, whereas the refinement community in the tradition of Back and von Wright uses $\top$ instead of 0, $\sqcap$ instead of $+$, ; instead of $\cdot$, $^\omega$ instead of $^\infty$ and $\sqsupseteq$ instead of $\leq$. In particular, $\sqsubseteq$ is the refinement order.

In the case of Kleene algebras it is well known that the equational theory is decidable [20] whereas the universal Horn theory is undecidable. In the case of demonic refinement algebras, no such results are presently known.

3 Automating Demonic Refinement Algebras

Mechanisation of refinement calculi has a long tradition. This has either been achieved through interactive theorem provers or through model checking. But these approaches are either limited in expressiveness or in automation. Automated deduction could provide an interesting alternative that helps closing the gaps between these approaches.

We evaluated eleven different ATP systems for finding proofs of theorems in demonic refinement algebras: E 0.999 [29], Equinox 1.3 [5], Fampire 1.3[1], Geo 2007f [8], iProver 0.2 [19], leanCoP 2.0 [24], Metis 2.0 [18], Otter 3.3 [21], Prover9 0607 [22], SPASS 3.0 [45], and Vampire 9.0 [28]. Initial tasks attempted with all the systems allowed us to select the most powerful system, which is Prover9, for the more difficult verification tasks.

The systems accept input converted from the standard TPTP syntax [38] for first-order equational logic. The input files contain axioms, for instance the demonic refinement algebra axioms, and a conjecture to be proved. The ATP systems try, by various

[1] Fampire 1.3. is Josef Urban's combination of the Vampire 8.1 prover [28] using the SPASS' FLOTTER clausifier [44].

means, to prove the conjecture from the axioms. ATP systems are, in principle, semi-decision procedures for first-order equational logic: if a conjecture is provable from the axioms then an ATP system can establish this through finite search (though perhaps beyond realistic resources); otherwise the ATP system may run forever. In practice many ATP system sacrifice some completeness in order to obtain better overall performance. When successful, an ATP system may present a proof, but some systems return only an assurance that a proof exists. The ATP systems use sophisticated heuristics and strategies for controlling the huge search spaces that may arise. Some systems allow strategies to be prescribed by the user, either as command line arguments or in the input file. This may have a crucial impact on success. However, we prefer to take a black-box approach that relies on the ATP systems' self-configuration "auto mode" capabilities, to obtain robust results that are more significant for formal software engineering practice. Also, this is common practice in the ATP community for empirical evaluations.

We use the following inequational encoding of demonic refinement algebras in Prover9 syntax.

```
                                    % operator precedences
op(500, infix, "+").                  % addition
op(490, infix, ";").                  % multiplication
op(480, postfix, "*").                % finite iteration
op(480, postfix, "'").                % strong iteration

formulas(sos).                      % idempotent semiring axioms
  x+y = y+x.                          % additive monoid
  x+(y+z) = (x+y)+z.
  x+0 = x.
  x+x = x.                            % idempotency of addition
  x;(y;z) = (x;y);z.                  % multiplicative monoid
  x;1 = x.
  1;x = x.
  x;(y+z) = x;y+x;z.                  % distributivity laws
  (x+y);z = x;z+y;z.
  0;x = 0.                            % left zero law
  x <= y <-> x+y = y.                 % definition of order
end_of_list.

formulas(sos).                      % inequational iteration axioms of DRA
  1+x;x* = x*.                        % Kleene star
  1+x*;x = x*.
  x;y+z <= y -> x*;z <= y.
  y;x+z <= y -> z;x* <= y.
  x' = 1+x;x'.                        % strong iteration
  y <= x;y+z -> y <= x';z.
  x' = x*+x';0.
end_of_list.
```

In the first part of the input, operator precedences are declared. Here, star and strong iteration bind more strongly than multiplication, which itself binds more strongly than addition. The second part of the input contains the idempotent semiring axioms that are used in demonic refinement algebras, that is, without the right zero axiom. The third part of the input contains an inequational encoding of the iteration axioms. Both sets of axioms are declared as *sets of support* (`sos`) for Prover9 – Prover9 accepts multiple sets of support, all of which are treated together as hypotheses for the proof. Finally, Prover9 requires that the goal to be proved is added in a special environment, for instance,

```
formulas(goals).
  x*;x* = x*.
end_of_list.
```

In demonic refinement algebras, inequalities and equations can be defined interchangeably. Every equation $x = y$ can be replaced by $x \leq y$ and $y \leq x$; every inequality $x \leq y$ can be replaced by $x + y = y$. Therefore, different encodings of demonic refinement algebra axioms are possible. Previous experiments with more than 500 theorems in Kleene algebras (e.g. [14,17]) suggest that an equational encoding is usually sufficient for finding ATP proofs of simple theorems. However, more complex theorems often succeed only with the inequational encoding. An equational encoding uses the same idempotent semiring axioms, except the definition of order, but replaces some of the iteration axioms as follows.

```
formulas(sos).                          % equational iteration axioms for DRA
  1+x;x* = x*.                            % Kleene star
  1+x*;x = x*.
  (x;y+z)+y = y -> x*;z+y = y.
  (y;x+z)+y = y -> z;x*+y = y.
  x' = 1+x;x'.                            % strong iteration
  y+(x;y+z) = x;y+z -> y+x';z = x';z.
  x' = x*+x';0.
end_of_list.
```

In practice, in order to obtain ATP proofs of the more difficult refinement laws described in this paper, further lemmas must sometimes be added to the axioms. Useful additions could be reflexivity and transitivity of the order, isotonicity of all operations or the case analysis law (1). We have developed a set of 31 lemmas that are useful for refinement. In the context of formal methods, all additional lemmas must be previously proved from the axioms. To do this, the eleven ATP systems were all given the 31 lemmas to prove. We used computers with a 2.8 GHz Intel Pentium 4 CPU, with 1 GB memory, running a Linux 2.6 operating system. We set a CPU time limit of 300 s, which is known to be sufficient for the ATP systems to prove almost all the theorems they would be able to prove even with a significantly higher limit [39]. The results of this experiment are shown in Table 1; a number indicates that a proof is found in that time; a "–" indicates that the system reaches the time limit or gives up before. The ATP system E proved all 31 lemmas, thus establishing their validity. The last lemma, proved only by E, also appears as the basic refinement law Eq16b in Section 4.

4 A Basic Refinement Toolkit

This section reports on our development of a toolkit of meaningful basic refinement laws within demonic refinement algebras. Since they automatically hold in Back and von Wright's refinement calculus, they are available for a wide range of applications. These laws provide the appropriate level of abstraction for proving more complex refinement laws, and for developing and analysing concrete refinements of programs and software systems. Practitioners of refinement will immediately recognise some of them as standard refinement laws. They can often replace the more low-level and less specific induction or coinduction axioms of demonic refinement algebras.

We have previously used ATP to develop and verify a toolkit of laws for several variants of Kleene algebras with the right zero axiom. Many of these laws are also valid in demonic refinement algebras. The Prover9 output files, for instance, present all hypotheses needed for individual proofs; so this can readily be checked. Table 1 contains some of the laws for demonic refinement algebra that have been automatically verified. For instance, $x \leq 1^\infty$ says that each demonic refinement algebra has a greatest element, namely 1^∞. Additional laws can be found at our website [16].

Table 1 Comparison of ATP Systems on Basic Refinement Laws

System	E	Famp.	Prover9	Vamp.	Otter	SPASS	Geo	Eq'x	iPro.	l'CoP	Metis
$0 + x = x$	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.0
$x^\infty = 1 + x^\infty x$	3.8	24.4	–	206.7	–	7.5	–	–	–	–	–
$x \leq y \Rightarrow x + z \leq y + z$	0.0	0.2	0.0	0.1	0.2	–	7.3	–	–	–	–
$x \leq y \Rightarrow z + x \leq z + y$	0.0	0.2	0.0	0.1	0.2	–	99.4	–	–	–	–
$x \leq y \Rightarrow xz \leq yz$	0.1	0.2	0.0	0.4	0.5	6.8	–	39.8	20.2	6.5	61.2
$x \leq y \Rightarrow zx \leq zy$	0.1	0.2	0.0	0.4	0.4	7.0	50.7	1.1	8.9	6.5	–
$x \leq y \Rightarrow x^* \leq y^*$	40.7	6.2	50.5	50.7	3.3	10.5	–	–	–	–	–
$x \leq y \Rightarrow x^\infty \leq y^\infty$	5.4	39.2	202.0	27.3	–	–	–	–	–	–	–
$x + y \leq z \Leftrightarrow x \leq z \wedge y \leq z$	0.0	0.2	53.4	0.1	18.5	–	–	–	–	–	–
$x \leq x$	0.0	0.1	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.2	5.6
$x \leq y \wedge y \leq z \Rightarrow x \leq z$	0.0	0.1	0.0	0.1	0.1	0.0	15.6	6.6	–	166.4	68.2
$x = y \Leftrightarrow x \leq y \wedge y \leq x$	0.0	0.1	0.1	0.8	0.1	14.2	0.1	0.0	20.9	0.6	7.2
$1 \leq x^*$	0.0	0.1	0.0	0.9	0.1	0.0	1.5	–	–	–	20.2
$x \leq x^*$	0.0	0.3	0.1	86.1	1.0	0.2	37.4	–	–	–	–
$x^{**} = x^*$	1.6	0.3	199.9	85.8	5.0	12.1	–	–	–	–	–
$x^*x^* = x^*$	0.2	0.5	57.9	0.8	5.3	12.4	–	–	–	–	–
$xx^* \leq x^*$	0.0	0.1	0.0	0.6	0.1	0.0	–	–	–	–	–
$xy \leq y \Rightarrow x^*y \leq y$	25.5	0.1	0.0	1.2	0.1	0.0	–	0.4	0.7	8.2	–
$yx \leq y \Rightarrow yx^* \leq y$	0.2	0.1	0.0	0.6	0.1	0.0	–	0.1	0.7	6.4	–
$0^* = 1$	0.0	0.1	0.0	0.8	0.1	0.0	0.1	0.0	1.0	0.2	0.0
$1^* = 1$	0.0	0.1	1.2	0.8	0.8	0.1	0.2	0.2	0.5	–	–
$x \leq 1^\infty$	0.1	0.1	0.2	1.0	0.1	0.0	1.6	–	176.6	–	–
$1^\infty x = 1^\infty$	1.0	0.1	0.5	1.2	0.2	0.1	0.8	0.6	–	–	–
$x^{\infty\infty} = 1^\infty$	0.3	0.1	2.0	0.8	0.5	1.0	13.8	–	–	–	–
$x^{*\infty} = 1^\infty$	0.4	0.4	1.9	1.0	1.6	0.9	14.6	–	–	–	–
$0^\infty = 1$	0.0	0.1	0.0	0.8	0.1	0.0	0.0	0.0	0.8	0.4	0.0
$x^\infty x^\infty = x^\infty$	19.9	24.1	268.9	–	4.7	14.1	–	–	–	–	–
$1 \leq x^\infty$	0.0	0.1	0.0	0.6	0.1	0.0	3.2	–	–	–	11.0
$x^{\infty *} = x^\infty$	20.0	24.1	269.1	146.4	–	–	–	–	–	–	–
$x^*x^\infty = x^\infty$	0.3	0.5	57.8	80.4	3.7	12.3	–	–	–	–	–
$x^\infty y^\infty \leq (x + y)^\infty$	63.6	–	–	–	–	–	–	–	–	–	–
Proved	31	30	29	29	27	25	17	12	11	10	9

The main focus of this work is the verification of complex refinement laws for concurrent systems, in particular action systems. In this context, the expressions $(x + y)^*$ or $(x + y)^\infty$ denote the repeated concurrent execution of two actions x and y. Concurrency refinement can often be analysed in three phases:

1. the deconstruction of concurrency into interleaving;
2. the transformation and refinement of interleaving;
3. the reconstruction of concurrency.

We have used automated deduction to verify a toolkit of basic refinement laws that supports refinement at all three phases. More precisely, all these laws are theorems of demonic refinement algebras. Most of these laws have already been proved manually by von Wright [42]. Though often short and concise, manual proofs can be surprisingly difficult and require some familiarity with Kleene algebras. The laws are as follows.

– *Sliding laws* slide loops over sequences:

$$x(yx)^* = (xy)^*x, \tag{2}$$
$$x(yx)^\infty = (xy)^\infty x, \tag{3}$$
$$(x^*y)^\infty = y^*(x^*y)^\infty. \tag{4}$$

– *Denesting laws* deconstruct and reconstruct concurrency:

$$(x + y)^* = x^*(yx^*)^*, \tag{5}$$
$$(x + y)^\infty = x^\infty(yx^\infty)^\infty, \tag{6}$$
$$(x + y)^\infty = (x^*y)^\infty x^\infty, \tag{7}$$
$$(x + y)^* = y^*x(x + y)^* + y^*, \tag{8}$$
$$(x + y)^\infty = y^*x(x + y)^\infty + y^\infty. \tag{9}$$

– *Simulation laws* are fundamental for data refinement:

$$yx \leq xz \Rightarrow y^*x \leq xz^*, \tag{10}$$
$$xy \leq zx \Rightarrow xy^* \leq z^*x, \tag{11}$$
$$xy \leq zx \Rightarrow xy^\infty \leq z^\infty x, \tag{12}$$
$$yx \leq xy \Rightarrow y^*x^* \leq x^*y^*, \tag{13}$$
$$yx \leq xy \Rightarrow y^\infty x^\infty \leq x^\infty y^\infty. \tag{14}$$

A dual law of (12), $yx \leq xz \Rightarrow y^\infty x \leq xz^\infty$, does not hold in demonic refinement algebra. Mace4 presents a counterexample with 3 elements.

– *Semicommutation laws* combine denesting with simulation:

$$yx \leq xy \Rightarrow (x + y)^* \leq x^*y^*, \tag{15}$$
$$yx \leq xy \Rightarrow (x + y)^\infty \leq x^\infty y^\infty. \tag{16}$$

Since $x^*y^* \leq (x + y)^*$ and $x^\infty y^\infty \leq (x + y)^\infty$ (this has also been automatically verified), the right-hand sides can even be strengthened to equalities.

– *Blocking laws* express that if y is always blocked by x, then x before an iteration of y reduces to x:

$$xy = 0 \Rightarrow xy^* = x, \tag{17}$$
$$xy = 0 \Rightarrow xy^\infty = x. \tag{18}$$

The eleven ATP systems mentioned in Section 3 were used to search for proofs of these laws. As a general rule of thumb, human reasoning in Kleene algebras and demonic refinement algebras is usually inequational. We therefore split the 17 laws into inequalities where possible, but also kept the equational variants, which are particularly hard for the ATP systems. In sum, this yields 41 expressions as proof goals. We tried to prove them using only the demonic refinement algebra axioms as hypotheses. We used the same machines as in Section 3, again with a 300 s CPU time limit. The results are presented in Table 2. Part "a" of an equation is the case $s \leq t$, part "b" is $t \leq s$, part "c" is the equation $s = t$. Implications containing an equation are split in a similar way, i.e., $u \Rightarrow s = t$ is split to an implication $u \Rightarrow s \leq t$, and an inequality $t \leq s$ without an antecedent.

The different ATP systems show dramatically different behaviours. On these more difficult proof tasks, the best three systems are clearly Prover9, Fampire, and Vampire. These three systems also perform well on the basic lemmas in Section 3, confirming their suitability for these ATP tasks. Prover9 and Fampire each prove some laws that no other system can handle, and in combination they prove 23 of the 41 laws. Most of the laws they cannot prove are equations, and these laws are largely covered by the corresponding pair of inequational proofs. Most of the remaining systems are able to prove only some simple refinement laws. In combination, Prover9 and Fampire appear to have all the ATP power available for proving these types of theorems, and we selected Prover9 for the more complex refinement laws in the remaining sections of this paper. It is interesting that E, which can prove all of the basic lemmas in Section 3, is less capable on these more difficult experiments. The results generally suggest that theorem proving in variants of Kleene algebras presents an interesting challenge for ATP systems, and is already feasible for some ATP systems.

To prove the remaining unproved laws, we used Prover9 with a 3000 s time limit, in some cases removed unnecessary axioms from the input, and in some cases used a *hypothesis learning* approach.

This approach is based on the following fact about first-order logic: *A goal cannot be proved from a set of axioms and lemmas if and only if there is some model in which all axioms and lemmas are true and the goal is false.* For hypothesis learning we use the finite model generator Mace4 [22] for model search. We start with a small set of axioms, lemmas, and the negated conjecture, and run Mace4. If a model is found then we add more axioms or lemmas. This continues until no model is found, at which point an ATP system (Prover9 in this case) is used to search for a proof. If the ATP system is unsuccessful we repeat the procedure to find another selection of axioms and lemmas. When a proof is found, the selection of axioms and lemmas, along with the conjecture, may be passed to the SRASS system [36] to extract a minimal but sufficient set of axioms and lemmas for a proof. In this paper we apply a manual form of hypothesis learning. An automation of these techniques is possible, but left for future work.

Using various combinations of the increased time limit, removal of axioms, and hypothesis learning, we were able to prove all the remaining laws. The results and methods used for each law are shown in Table 3. An equational law was considered to be proved if the pair of corresponding inequalities was proved., e.g., Prover9 could prove Eq02a and Eq02b, which together imply Eq2c (Equation (2)). For these (and all further) proof experiments with Prover9 we used a computer with a 3.0 GHz Intel Pentium 4 CPU, with 2 GB memory and hyper-threading, running a Linux 2.6 operating system. The first column of Table 3 names the law to be proved, and the second column is the CPU time taken for the successful proof. The third column indicates if some

Table 2 Comparison of ATP Systems on Basic Refinement Laws

System	Prover9	Famp.	Vamp.	E	Metis	Otter	SPASS	Geo	Eq'x	l'CoP	iPro.
Eq02a	0.0	–	117.4	–	–	–	–	–	–	–	–
Eq02b	3.6	–	–	–	–	–	–	–	–	–	–
Eq02c	–	–	–	–	–	–	–	–	–	–	–
Eq03a	0.0	–	117.4	–	–	–	–	–	–	–	–
Eq03b	–	–	–	–	–	–	–	–	–	–	–
Eq03c	–	–	–	–	–	–	–	–	–	–	–
Eq04a	0.0	0.2	0.0	0.0	24.5	0.8	0.2	–	–	–	–
Eq04b	245.9	40.4	–	–	–	–	–	–	–	–	–
Eq04c	–	206.6	–	–	–	–	–	–	–	–	–
Eq05a	–	19.9	117.6	–	–	–	–	–	–	–	–
Eq05b	0.1	–	117.4	–	–	–	–	–	–	–	–
Eq05c	–	–	–	–	–	–	–	–	–	–	–
Eq06a	–	–	–	–	–	–	–	–	–	–	–
Eq06b	–	19.6	117.6	–	–	–	–	–	–	–	–
Eq06c	–	–	–	–	–	–	–	–	–	–	–
Eq07a	1.8	–	–	–	–	–	–	–	–	–	–
Eq07b	–	–	–	–	–	–	–	–	–	–	–
Eq07c	–	–	–	–	–	–	–	–	–	–	–
Eq08a	214.7	–	–	–	–	–	–	–	–	–	–
Eq08b	0.0	18.3	117.4	–	–	–	–	–	–	–	–
Eq08c	–	–	–	–	–	–	–	–	–	–	–
Eq09a	1.8	115.7	–	–	–	–	–	–	–	–	–
Eq09b	–	20.4	–	–	–	–	–	–	–	–	–
Eq09c	–	–	–	–	–	–	–	–	–	–	–
Eq10	44.2	88.7	–	–	–	–	–	–	–	–	–
Eq11	4.8	102.9	–	–	–	–	–	–	–	–	–
Eq12	–	–	–	–	–	–	–	–	–	–	–
Eq13	–	–	–	–	–	–	–	–	–	–	–
Eq14	–	–	–	–	–	–	–	–	–	–	–
Eq15a	–	–	–	–	–	–	–	–	–	–	–
Eq15b	177.3	183.9	43.1	110.5	–	–	–	–	–	–	–
Eq15c	–	–	–	–	–	–	–	–	–	–	–
Eq16a	–	–	–	–	–	–	–	–	–	–	–
Eq16b	–	–	–	62.8	–	–	–	–	–	–	–
Eq16c	–	–	–	–	–	–	–	–	–	–	–
Eq17a	0.0	0.1	0.0	0.0	39.4	0.3	0.0	6.2	0.1	201.9	2.2
Eq17b	0.0	0.2	0.1	0.0	66.0	0.7	0.2	7.7	–	–	–
Eq17c	0.1	0.1	0.1	0.0	0.4	0.3	0.3	257.3	1.4	233.1	–
Eq18a	0.2	0.1	0.1	0.3	86.0	0.7	0.3	55.3	11.4	–	–
Eq18b	0.0	0.2	0.1	0.0	57.2	1.2	0.3	23.5	–	–	–
Eq18c	0.2	0.1	0.1	0.3	0.4	0.8	0.3	54.7	11.3	–	–
Proved	19	17	14	9	7	7	7	6	4	2	1

unnecessary axioms were removed from the input: "+" stands for some removal and "–" indicates no removal. The remaining columns indicate if lemmas were added. The fourth column indicates if the case analysis law (1) was added. The fifth column indicates if some isotonicity laws were added. The last column indicates if some previously proved laws were added. We distinguish these cases since the case analysis law and isotonicities are basic properties that are needed quite often and could, in principle, always be added to enhance proof search. Adding these laws alone, Prover9 can verify three additional refinement laws, as Table 3 shows. Again we refer to our website for details on the machine proofs.

Table 3 Requirements for the remaining proofs using Prover9

	t[s]	axioms.	cases	iso.	laws
Eq03b	76.2	+	−	−	+
Eq05a	14.3	−	+	−	−
Eq06a	0.0	−	+	−	+
Eq06b	46.1	+	+	−	+
Eq07b	0.0	+	−	−	+
Eq09c	7.6	+	+	+	−
Eq12	866.6	−	−	−	−
Eq13	0.0	−	−	−	+
Eq14	17.9	−	−	−	+
Eq15a	376.9	+	+	+	−
Eq16a	216.2	+	−	+	−
Eq16b	38.5	−	+	+	−

5 Denesting in Detail

Table 2 shows that no ATP system was able to prove Eq6b directly from the axioms. Here, for the sake of simplicity, we discuss an ATP proof of a symmetric denesting law, namely

$$(x+y)^\infty \leq (x^\infty y)^\infty x^\infty,$$

from which the non-trivial part of Equation (6) follows in one step by applying Equation (3).

We use this example to present a typical machine proof and to discuss its translation into an equational textbook-style proof. The proof could be found by Prover9 without any restrictions, that is, from the axiom set

```
formulas(sos).
   x+y = y+x  &  x+(y+z) = (x+y)+z  &  x+0 = x  &  x+x = x.
   x;(y;z) = (x;y);z  &  x;1 = x  &  1;x = x.
   x;(y+z) = x;y+x;z  &  (x+y);z = x;z+y;z.
   0;x = 0  &  x<=y <-> x+y = y.

   1+x;x* = x*  &  1+x*;x = x*.
   (x;y+z <= y -> x*;z <= y)  &  (y;x+z <= y -> z;x* <= y).
   x' = 1+x;x'  &  (y <= x;y+z -> y <= x';z).
   x' = x*+x';0.
end_of_list.
```

Prover9 immediately found the following proof.

```
1 x + y = y + x & x + (y + z) = (x + y) + z & x + 0 = x & x + x = x
                                                  # label(non_clause)  [ assumption ]
2 x ; (y ; z) = (x ; y) ; z & x ; 1 = x & 1 ; x = x
                                                  # label(non_clause)  [ assumption ]
3 x ; (y + z) = x ; y + x ; z & (x + y) ; z = x ; z + y ; z
                                                  # label(non_clause)  [ assumption ]
4 0 ; x = 0 & x <= y <-> x + y = y                # label(non_clause)  [ assumption ]
5 x' = 1 + x ; x' & (y <= x ; y + z -> y <= x' ; z)   # label(non_clause)  [ assumption ]
6 (x + y)' <= (x' ; y)' ; x'              # label(non_clause) # label(goal)  [ goal ]
7 x + y = y + x                                                    [ clausify 1 ]
8 (x + y) + z = x + (y + z)                                        [ clausify 1 ]
9 x + x = x                                                        [ clausify 1 ]
10 (x ; y) ; z = x ; (y ; z)                                       [ clausify 2 ]
11 x ; 1 = x                                                       [ clausify 2 ]
12 x ; (y + z) = x ; y + x ; z                                     [ clausify 3 ]
13 x ; y + x ; z = x ; (y + z)                                  [ copy 12, flip ]
14 (x + y) ; z = x ; z + y ; z                                     [ clausify 3 ]
15 x ; y + z ; y = (x + z) ; y                                  [ copy 14,flip ]
```

```
16 x <= y | x + y != y                                                    [ clausify 4 ]
17 x' = 1 + x ; x'                                                        [ clausify 5 ]
18 1 + x ; x' = x'                                                      [ copy 17,flip ]
19 -(x <= y ; x + z) | x <= y' ; z                                        [ clausify 5 ]
20 -((c1 + c2)' <= (c1' ; c2)' ; c1')                                        [ deny 6 ]
21 x + (y + z) = y + (x + z)                                        [ para 7 8, rewrite 8 ]
22 x ; (1 + y) = x + x ; y                                             [ para 11 13, flip]
23 x <= x                                                                   [ hyper 16 9]
24 -((c1 + c2)' <= c1' ; (1 + c2 ; (c1 + c2)'))                  [ ur 19 20, rewrite 10 7 22 ]
25 -((c1 + c2)' <= (c1 + c2)')                                  [ ur 19 24, rewrite 21 15 18 ]
26 $F                                                                     [ resolve 25 23 ]
```

Of course, this proof represents only a tiny part of the proof search, which consisted of about 250 inferences.

The machine proof, based on ordered resolution and paramodulation, is not intended to be readable for humans, but can easily be retranslated into a textbook-style equational proof. The essential part of the machine proof starts at line 24. First, to prove the goal, it suffices by strong coinduction to show that

$$(x+y)^\infty \leq x^\infty y(x+y)^\infty + x^\infty = x^\infty(1 + y(x+y)^\infty).$$

This step corresponds to line 24 of the machine proof, using the instance of the distributivity law in line 22.

Second, applying strong coinduction again, it suffices to show that

$$(x+y)^\infty \leq x(x+y)^\infty + y(x+y)^\infty + 1.$$

This corresponds to the main step leading to line 25. The additional rewriting steps performed use the expressions at line 21, 15 and 18, that is, associativity, distributivity and strong unfold, to transform the right-hand side of this equation via

$$x(x+y)^\infty + y(x+y)^\infty + 1 = (x+y)(x+y)^\infty + 1 = (x+y)^\infty$$

into the right-hand side at line 25. This finishes the equational proof, whereas in the resolution proof, line 25 contradicts reflexivity of $\leq$ and yields the empty clause by resolution. Interestingly, the equational reconstruction of the machine proof is simpler and shorter than a previous manual proof [42].

6 Data Refinement

Data refinement is the correctness-preserving replacement of abstract data types like sets in specifications by concrete ones, for instance arrays, lists or heaps, in implementations. Data types are observable through the effects of their operations on states, they can also be initialised and finalised with respect to a global state space. An abstract data type is refined by a concrete one if all sequences of operations on the abstract data type (including initialisation and finalisation) can safely be replaced by a corresponding sequence on the concrete data type. This criterion can be made local by introducing abstraction relations between inputs and outputs of the operations at the abstract level and those at the concrete level. Further information can be found in the books by Back and von Wright [3] and by de Roever and Engelhardt [9].

Back and von Wright have presented several laws for data refinement of action systems in the predicate transformer setting of their refinement calculus [3]. One of these laws has already been translated into demonic refinement algebras by von Wright. Here,

we present an automated proof of this law. We now leave the level of axiomatic demonic refinement algebras and predominantly work at the more abstract level of basic refinement laws. Hypothesis learning becomes essential for guiding the ATP system. This analysis of the data refinement law illustrates the applicability of our refinement toolkit, and, based on previous work on diagrammatic reasoning with Kleene algebras [31, 12], it also shows that the usual refinement diagrams can in principle be recovered from the ATP output.

The data refinement law can be written as follows.

Theorem 6.1 *In any demonic refinement algebra, let* $b^\infty = b^*$, $za' \leq az$, $zb \leq z$, $s' \leq sz$ *and* $ze' \leq e$. *Then*

$$s'(a'+b)^\infty e' \leq sa^\infty e.$$

The first hypothesis says that b cannot loop infinitely (we call this behaviour *weak termination* in Section 9). The second hypothesis says that a is data refined by a' with respect to upward simulation z. By the third hypothesis, 1 is data refined by b. The fourth and fifth condition expresses the standard data refinement of initialisations and finalisations.

Manual hypothesis learning can greatly be simplified by some knowledge about data refinement. It can be expected that some forms of denesting and semicommutation will suffice for deconstructing and reconstructing concurrency. For the transformation of interleaving, simulation laws seem highly relevant, whereas the laws of the additive monoid seem rather irrelevant and could be discarded. The effect of the star and strong iteration axioms might be captured by simulation laws, so that these axioms could be discarded, too. Based on trial and error, it turns out that the following set of demonic refinement algebra axioms and additional refinement laws suffices for a proof. From the original set of axioms, we use only the multiplicative associativity axiom and a simple version of one of the star induction laws. As in Section 5, we use reflexivity and transitivity. We also use isotonicity of multiplication, finite and strong iteration, and we add the Equations (6) and (12).

```
formulas(sos).
  x;(y;z) = (x;y);z.                % commutativity
  x;y<=x -> x;y*<=x.                % simple induction
  x <= x.                           % reflexivity
  x<=y & y<=z -> x<=z.              % transitivity.
  x<=y -> x;z<=y;z.                 % isotonicity
  x<=y -> z;x<=z;y.
  x<=y -> x*<=y*.
  x<=y -> x'<=y').
  (x+y)' = y';(x;y')'.              % Equation  (6)
  z;y<=x;z -> z;y'<=x';z.           % Equation (12)
end_of_list.
```

From this set of support, Prover9 found a 36-step proof that could again be retranslated into an equational proof. Initialisation and finalisation can be separated from the loop refinement. They imply that it suffices to show that $sz(a'+b)^\infty e' \leq sa^\infty ze'$ and therefore, by isotonicity,

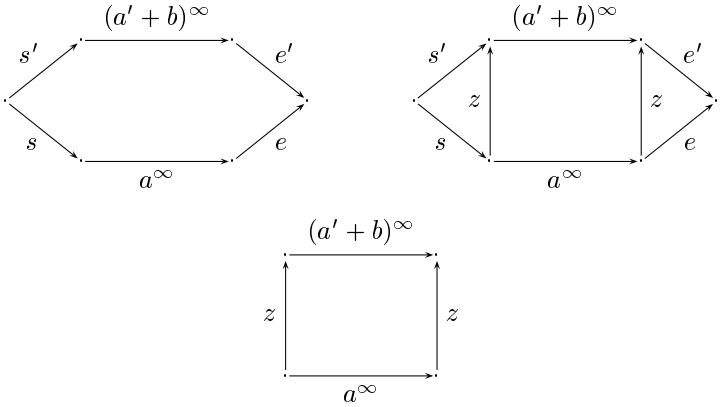
$$z(a'+b)^\infty \leq a^\infty z.$$

The left-hand side of this expression can be denested and, using the assumptions $b^\infty = b^*$, $zb \leq z$ and the simulation law, be simplified to

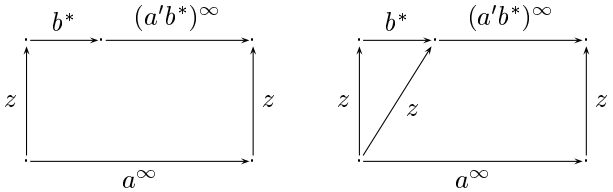
$$z(a'+b)^\infty = zb^\infty(a'b^\infty)^\infty = zb^*(a'b^*)^\infty \leq z(a'b^*)^\infty.$$

But $z(a'b^*)^\infty \leq a^\infty z$ follows from the strong simulation law and $za'b^* \leq azb^* \leq az$.

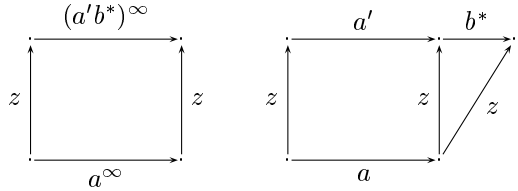
This algebraic reasoning can easily be translated into diagrams, as used in term rewriting (cf. [40]) and refinement (cf. [3]). The usefulness of Kleene algebras for diagrammatic reasoning has been demonstrated in [12]. In particular, glueing of diagrams along edges corresponds to isotonicity reasoning. Therefore, the following diagrams describe the transition from the succedent of the data refinement theorem to the analysis of the infinite loop.



The next sequence of diagrams describes essentially the reasoning in the above sequence of equations.



The last two diagrams describe the calculation from the simulation assumption to the loop via the strong simulation law (12).



The correspondence between algebra and diagrams could be made more precise by putting the refinement laws into diagrammatic form. Demonic refinement algebras would then yield an algebraic semantics for these diagrams and ATP would automate this semantics. A further elaboration of this correspondence seems very promising for proof visualisation.

7 Atomicity Refinement

In 1989 Back proved a rather complex atomicity refinement theorem for action systems by reasoning over sequences of program states in infinitary logic [2]. This proof spreads

over several pages. It has been replayed later at the level of predicate transformers and in demonic refinement algebras [42,43], where it still fills almost two pages. Our automated analysis reveals some errors in this proof and allows us to eliminate some unnecessary hypotheses. The following theorem presents a cleaned-up version of Back's atomicity refinement theorem in demonic refinement algebras.

Theorem 7.1 *In any demonic refinement algebra, let* $s \leq sq$, $a \leq qa$, $qb = 0$, $rb \leq br$, $(a + r + b)l \leq l(a + r + b)$, $rq \leq qr$, $ql \leq lq$, $r^* = r^\infty$ *and* $q \leq 1$. *Then*

$$s(a + r + b + l)^\infty q \leq s(ab^\infty q + r + l)^\infty.$$

A discussion and explanation of this law can be found in Back's original article [2]. For ATP, a semantic understanding is irrelevant. Given the length of von Wright's proof in demonic refinement algebra it is no surprise that ATP systems do not succeed in one full sweep. However, an automated proof with Prover9 up to the reconstruction of concurrency is possible. We therefore split Theorem 7.1 into two steps.

Lemma 7.1 *In any demonic refinement algebra,*

(i) $s(a + r + b + l)^\infty q \leq sl^\infty qr^\infty q(ab^\infty qr^\infty)^\infty$ *follows from the hypotheses of Theorem 7.1 except* $q \leq 1$*;*

(ii) if $q \leq 1$. *then* $sl^\infty qr^\infty q(ab^\infty qr^\infty)^\infty \leq s(ab^\infty q + r + l)^\infty$.

Obviously, the hypothesis $q \leq 1$ is particularly dangerous in proof search, since 1 can be added everywhere in sequences and then weakened to q. As before, the proofs heavily rely on hypothesis learning. The following hypotheses can be used for both proofs.

```
formulas(sos).
  x+(y+z) = (x+y)+z  &  x+y = y+x.
  x;1 = x & 1;x = x  &  x;(y;z) = (x;y);z  &  0;x = 0.
  x <= x  &  (x<=y & y<=z -> x<=z).
  (x<=y -> x;z<=y;z)  &  (x<=y -> z;x<=z;y)  &   (x<=y -> x'<=y').
end_of_list.

formulas(sos).
  (x';y') <= (y+x)'.                    % trivial part of semicommutation (page 5)
  x;(y;x)' = (x;y)';x.                  % Equation  (3)
  (x+y)' = x';(y;x')'.                  % Equation  (6)
  y;x<=x;z -> y*;x<=x;z*.               % Equation (10)
  x;y<=z;x -> x;y'<=z';x.               % Equation (12)
  y;x<=x;y -> (x+y)'=x';y'.             % Equation (16) (strengthened)
  x;y=0 -> x;y'=x.                      % Equation (18)
end_of_list.
```

The goal for Lemma 7.1(i) is

```
formulas(goals).
  all a all b all l all q all r all s(
    a<=q;a  &  r;b<=b;r  &  (a+(r+b));l<=l;(a+(r+b))  &  r;q<=q;r
    &  r*=r'  &  q;l<=l;q  &  q;b=0  &  s<=s;q
    ->
    s;(((a+(r+b))+l)';q) <= (s;(l';q));((r';q);((a;b');(q;r')))')).
end_of_list.
```

We also keep the full set of assumption in the proof of Lemma 7.1(ii), although this is not strictly necessary.

```
formulas(goals).
  all a all b all l all q all r all s(
     a<=q;a  &  r;b<=b;r  &  (a+(r+b));l<=l;(a+(r+b))  &  r;q<=q;r
     &  r*=r'  &  q;l<=l;q  &  q;b=0  &  s<=s;q  &  q<=1
     ->
     (s;(l';q));((r';q);((a;b');(q;r')))') <= s;(((a;b');q+r)+l)').
end_of_list.
```

A machine proof of Lemma 7.1(i) required about 1100 s and 75 steps.

Although this resolution and superposition proof is not particularly readable for humans, it inspires the following equational proof that gives an impression of the complexity of reasoning involved.

$$\begin{aligned}
s(a+b+r+l)q &= sl^\infty(a+b+r)^\infty q \\
&= sl^\infty(b+r)^\infty a(b+r)^\infty)^\infty q \\
&= sl^\infty b^\infty r^\infty(ab^\infty r^\infty)^\infty q \\
&\leq sl^\infty b^\infty r^\infty(qab^\infty r^\infty)^\infty q \\
&= sl^\infty b^\infty r^\infty q(ab^\infty r^\infty q)^\infty \\
&\leq sql^\infty b^\infty r^\infty q(ab^\infty r^\infty q)^\infty \\
&\leq sl^\infty qb^\infty r^\infty q(ab^\infty r^\infty q)^\infty \\
&\leq sl^\infty qr^\infty q(ab^\infty r^\infty q)^\infty \\
&= sl^\infty qr^\infty q(ab^\infty r^* q)^\infty \\
&\leq sl^\infty qr^\infty q(ab^\infty qr^*)^\infty \\
&= sl^\infty qr^\infty q(ab^\infty qr^\infty)^\infty.
\end{aligned}$$

The first step uses the semicommutation law (16). The second step uses the denesting law (6). The third step uses again the semicommutation law (16) . The fourth step uses the assumption $a \leq qa$. The fifth step uses the sliding law (3). The sixth step uses the assumption $s \leq sq$. The seventh step uses the simulation law (12). The eighth step uses the disabledness law (18) and the assumption $qb = 0$. The ninth step uses the assumption $r^\infty = r^*$. The tenth step uses the star simulation law (10). The eleventh step uses again $r^\infty = r^*$.

Given the length of the equational proof, the success of ATP seems rather impressive. In general we find that machine proofs and their equational reconstructions in textbook style differ by a length factor between 5 and 10. A main reason is that term rearrangements due to associativity or commutativity are usually not displayed in hand-written proofs.

The automated proof of Lemma 7.1(ii) is much simpler than that of (i). It required a few milliseconds, has 30 steps and can be found at our web site. Its translation into an equational proof is not particularly interesting. The combination of the two parts of Lemma 7.1 has not been achieved, since the introduction of the additional assumption $q \leq 1$, which is needed in the second proof, leads to an explosion of the search space.

It might be possible to obtain a fully automated proof with a chaining-based prover, which provides a more effective treatment of inequational reasoning. This, however, is not available in state-of-the-art ATP systems. Therefore a fully automated proof search for Back's atomicity refinement law (Theorem 7.1) can be seen as a challenge for ATP systems. Again, the proofs can be translated into diagrams, but we do not further pursue this direction.

8 Atomicity Refinement Light

Our attempts to prove the atomicity refinement theorem led us to consider simplified variants. Setting $r = l = 0$, an automated proof can be obtained in a few seconds. Setting only $l = 0$, Theorem 7.1 simplifies as follows.

Proposition 8.1 *Let $s = sq$, $a = qa$, $qb = 0$, $(a + b)l \leq l(a + b)$, $ql \leq lq$ and $q \leq 1$. Then*

$$s(a + b + l)^\infty q = s(ab^\infty q + l)^\infty.$$

A fully automated proof of this statement is now possible. The input file, based on hypothesis learning, is

```
formulas(sos).
  x;1 = x  &  1;x = x  &  x;(y;z) = (x;y);z.
  x <= x  &  (x<=y & y<=z -> x<=z).
end_of_list.

formulas(sos).
  y<=1 -> (x;y);z<= x;z.                  % one with context
  z;(x';y')<=z;(y+x)'.                    % trivial part of semicommutation with context
  x;(y;x)'=(x;y)';x.                      % Equation  (3)
  (x+y)'= y';(x;y')'.                     % Equation  (6)
  x;y<=z;x -> (v;(x;y'));w<=(v;(z';x));w. % Equation (12) with context
  y;x <= x;y -> (y+x)'=x';y'.             % Equation (16)
  x;y = 0 -> x;y'=x.                      % Equation (18)
end_of_list.
```

To shorten the proof search, contexts are hard-coded into the rules to avoid the free generation of unnecessary contexts by the isotonicity laws. The proof goal is

```
formulas(goals).
  all s all a all b all l all q
    (s=s;q  &  a=q;a  &  q;b = 0  &  (a+b);l<=l;(a+b)  &  q;l <= l;q  &  q<=1
    ->
    s;(((a+b)+l)';q)<=s;((a;b');q+l)').
end_of_list.
```

The proof took 187 *s* and has 46 steps. Since the machine proof is again not particularly readable, we will only present its equational translation and refer the interested reader to our web site.

$$\begin{aligned}
s(a + b + l)^\infty q &= sl^\infty (a + b)^\infty q \\
&= sl^\infty b^\infty (ab^\infty)^\infty q \\
&= sl^\infty b^\infty (qab^\infty)^\infty q \\
&= sl^\infty b^\infty q(ab^\infty q)^\infty \\
&= sql^\infty b^\infty q(ab^\infty q)^\infty \\
&\leq sl^\infty qb^\infty q(ab^\infty q)^\infty \\
&= sl^\infty qq(ab^\infty q)^\infty \\
&\leq sl^\infty (ab^\infty q)^\infty \\
&= s(ab^\infty q + l)^\infty.
\end{aligned}$$

The first step applies the semicommutation law (16). The second step uses the denesting law (6). The third step uses the assumption $a = qa$. The fourth step uses the sliding law (3). The fifth step uses the assumption $sq = s$. The sixth step uses the simulation law (12). The seventh step uses the assumption $qb = 0$ and the disabledness law (18). The eight step uses the assumption $q \leq 1$. The last step uses isotonicity and $x^\infty x^\infty = x^\infty$.

The assumptions $a = qa$ and $sq = s$ could again, as in the case of Theorem 7.1, be weakened to $s \leq sq$ and $a \leq qa$.

9 A New Loop Refinement Law

Using different variants of Kleene algebras, Bachmair and Dershowitz's classical termination theorem, which can be expressed and proved in variants of Kleene algebras [31], has recently been automated [32]. This theorem states that *termination of the union of two rewrite system can be separated into termination of the individual systems if one rewrite system quasicommutes over the other.* The proof in Kleene algebra is justified since the termination theorem appeals solely to relational aspects of rewriting and because relations under union, relational product and the reflexive-transitive-closure operation together with the empty relation and the unit relation form a Kleene algebra. As in the case of Back's atomicity refinement theorem, Bachmair and Dershowitz's original proof was based on an intricate semi-formal analysis of infinite chains of rewrite steps and a later formal proof [26] involves an infinitary variant of Ramsey's theorem. So the abstract algebraic proofs present a considerable abstraction and simplification and it is no surprise that they solved a well-known challenge in the area [6].

In this section we generalise the previous proofs for demonic refinement algebras from quasicommutation to a notion of weak quasicommutation and therefore obtain and automatically prove stronger statements. Since we are working in demonic refinement algebras, for reasons discussed in Section 2, this generalisation does not extend to relational models, whence not to rewriting. Nevertheless, the results are immediately relevant as refinement laws in Back and von Wright's refinement calculus.

Generalising Bachmair and Dershowitz's relational notion, we say that an element x of a demonic refinement algebra *quasicommutes* over an element y if

$$yx \leq x(x+y)^*. \tag{19}$$

For demonic refinement algebras, the following weaker concept is even more interesting. An element x *weakly quasicommutes* over an element y if

$$yx \leq x(x+y)^\infty. \tag{20}$$

The following fact is an immediate consequence of the isolation axiom; Prover9 needed about 150 *s* for a proof.

Lemma 9.1 *Let x and y be elements of some demonic refinement algebra. Then x quasicommutes over y if x weakly quasicommutes over y.*

Mace4 showed that a similar relationship does not hold in other variants of Kleene algebras, including those providing the relational semantics that underly Bachmair and Dershowitz's termination theorem.

Termination, as appropriate for rewrite systems, can be expressed in omega algebras and divergence modules by adding an operation for strictly infinite iteration, but termination can of course be expressed in demonic refinement algebras as well, and even in two different ways. We say that

- *x weakly terminates* if $x^\infty = x^*$,
- *x strongly terminates* if $x^\infty 0 = 0$.

Lemma 9.2 *If an element of a demonic refinement algebra strongly terminates, then it weakly terminates, but not conversely.*

Table 4 Demonic refinement algebra with 3 elements

$+$	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

$\cdot$	0	1	2
0	0	0	0
1	0	1	2
2	2	2	2

$*$	
0	1
1	1
2	2

∞	
0	1
1	2
2	2

Implication of weak termination by strong termination could be shown in almost no time by Prover9. Mace4 presented a demonic refinement algebra with three elements that refutes the implication of strong termination by weak termination; it is shown in Table 4.

Strong and weak termination yield two variants of the termination theorem that are relevant to the refinement calculus. We first consider that for strong termination.

Theorem 9.1 *Let x and y be elements of some demonic refinement algebra and let x weakly quasicommute over y. Then $x + y$ strongly terminates if and only if x and y strongly terminate.*

An automated proof from the full set of demonic refinement algebra axioms with Prover9 took about 1000 *s* for the right-to-left direction and 37 *s* for its converse, which is a trivial application of isotonicity. The machine proof of the non-trivial right-to-left direction is surprisingly short. For presenting a corresponding equational proof, the following fact about weak quasicommutation is very useful:

$$yx \leq x(x+y)^\infty \Leftrightarrow y^*x \leq x(x+y)^\infty. \tag{21}$$

The right-to-left direction trivially follows from $y \leq y^*$. For the converse direction, it suffices by star induction to show that $x+yx(x+y)^\infty \leq x(x+y)^\infty$. The first summand of the left-hand side is smaller than the right-hand side, since $1 \leq (x+y)^\infty$. For the second summand, $yx(x+y)^\infty \leq x(x+y)^\infty(x+y)^\infty = x(x+y)^\infty$ by quasicommutation. Prover9 needed almost no time for a proof.

With this preparation, the proof of Theorem 9.1 is then a one-liner.

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

The first step uses (9). The second step uses (21). The third step uses $x^\infty x^\infty = x^\infty$. The fourth step uses strong coinduction. Now strong termination of x and y implies $(x+y)^\infty 0 = x^\infty y^\infty 0 = x^\infty 0 = 0$, which finishes the proof.

The first part of the equational proof of Theorem 9.1, before termination comes into play, is interesting in its own right. So we turn it into a theorem.

Theorem 9.2 *Let x and y be elements of some demonic refinement algebra and let x weakly quasicommute over y. Then*

$$(x+y)^\infty = x^\infty y^\infty.$$

Prover9 needed about 600 *s* from the full axiom set without any additional hypotheses.

Theorem 9.2 is a new refinement theorem for (potentially) infinite loops. It says that in the presence of weak quasicommutation, loops in which actions x and y are nondeterministically executed can be separated into loops of x followed by loops of y. The assumption of weak quasicommutation is quite general. It subsumes not only

quasicommutation, but also conditions of the form $yx = xy$, which express independence of actions x and y, and conditions of the form $yx \leq xy$, $yx \leq xy^*$, $yx \leq xy^\infty$, $yx \leq xx^*y^*$ or $yx \leq xx^\infty y^\infty$, which express preference of x over y.

Theorem 9.2 immediately implies a variant of Theorem 9.1 for weak termination.

Corollary 9.1 *Let x and y be elements of some demonic refinement algebra and let x weakly quasicommute over y. Then $x+y$ weakly terminates if x and y weakly terminate.*

Prover9 needed 4000 *s* to prove this corollary from Theorem 9.2. A manual proof is as follows. Let $x^\infty = x^*$ and $y^\infty = y^*$. Then, by Theorem 9.2,

$$(x+y)^\infty \leq x^\infty y^\infty = x^*y^* \leq (x+y)^*(x+y)^* \leq (x+y)^*,$$

whereas $(x+y)^* \leq (x+y)^\infty$ follows from isolation.

The converse of Corollary 9.1 does not hold. Mace4 presented again the counterexample from Table 4. Therefore the theorem for weak termination is indeed weaker than that for strong termination.

Since, by Lemma 9.2, weak quasicommutation implies quasicommutation, all statements proved in this section hold a forteriori for quasicommutation. These less general statements and statements related to standard Kleene algebras with an operation for strictly infinite iteration have previously been automated [32]. Work based on an influential paper of Podelski and Rybalchenko [26] shows that such termination and refinement theorems have immediate relevance for the termination analysis of concrete programs. The development and verification of such laws is therefore more than an abstract mathematical exercise.

10 Conclusion

We have shown that a substantial part of the calculus of demonic refinement algebras can be mechanised in ATP systems, that a useful toolkit of basic refinement laws can be developed and be automatically verified in this setting, and that some refinement laws of considerable complexity can automatically be developed and verified. In particular, this verification led to the simplification of some known proofs, to the discovery of new refinement laws and to the discovery of some errors in previous theorems and proofs (in the relational setting, an error in a soundness proof for data refinement in the book of de Roever and Engelhardt has also been discovered with the use of ATP and counterexample search [17]). This suggests that large parts of Back and von Wright's original refinement calculus can also effectively be automated and that automated deduction provides an innovative technology for program refinement. We expect that these result scale to the analysis and refinement of probabilistic programs and protocols that can be based on similar algebraic formalisms [23].

A second contribution consists in the evaluation and comparison of eleven state-of-the-art ATP systems on the algebraic refinement proofs in the context of Kleene algebras. Beyond the overall success, our experiments reveal significant differences in their performance. While some ATP systems were able to prove a substantial number of theorems within reasonable time limits, most of the systems could only prove some simple theorems. One might therefore conclude that automated theorem proving with Kleene algebras presents an interesting challenge for ATP systems. Because of the immediate relevance of these proofs for formal methods and automated termination

analysis, one might further argue that the combination of Kleene algebras and similar algebraic structures with off-the-shelf ATP systems presents a novel verification challenge with considerable potential for formal software development.

A further significant contribution of this paper consists of the research questions that arise from these results.

From the refinement point of view, certainly the most interesting question is how far the present automation results for refinement laws scale to the refinement of concrete algorithms and protocols. To this end, extensive case studies must be undertaken and it might be helpful to integrate the automated approach into interactive theorem proving.

From the ATP point of view, the integration of order-based reasoning [1,30] might drastically improve the performance on verification tasks that are strongly based on inequational reasoning, for instance with isotonicity. The example of refinement shows that automated reasoning with inequalities is perhaps more difficult, but surely not less interesting than its equational counterpart. Also, an integration of concrete data structures and data types, such as numbers, lists, arrays, queues and their handling through standard interfaces would be very useful. This supports research in the SMT style of automated reasoning [27]. Moreover, ATP systems should be complemented by tools that automatically select the necessary axioms and useful lemmas, since that seems indispensable for more advanced proof tasks. Tools such as SRASS and MaLARea [41] are steps in this direction. Finally, further experiments seem necessary to identify a standard "working set" of hypotheses about demonic refinement algebras that is most useful and powerful in practice.

An obvious question concerns the relevance and trustworthiness of machine proofs that occur in our approach. The first question is rather philosophical: Should unreadable machine proofs replace human proofs? The second question seems more pragmatic: Should we trust the results provided by ATP systems that are certainly complex and therefore error-prone? The answer to the first question depends on the context. For program verification, when the existence of a proof is more important than its content, machine proofs are usually acceptable. As to the second question, current research [7] addresses the post-verification of ATP output by interactive proof checkers, which are much simpler than ATP systems such that their soundness can easily be checked by humans. Semantic verification of proofs by trusted ATP systems provides a similar assurance of correctness [34]. Due to the fine granularity of machine proofs, this post verification can be entirely automatic.

Acknowledgements We are very grateful to Tony Hoare for his comments and encouragement.

References

1. L. Bachmair and H. Ganzinger. Ordered chaining calculi for first-order theories of transitive relations. *J. ACM*, 45(6):1007–1049, 1998.
2. R.-J. Back. A method for refining atomicity in parallel algorithms. In E. Odijk, M. Rem, and J.-C. Syr, editors, *Parallel Architectures and Languages Europe*, volume 366 of *Lecture Notes in Computer Science*, pages 199–216. Springer, 1989.
3. R.-J. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction*. Graduate Texts in Computer Science. Springer, 1998.
4. R.-J. Back and J. von Wright. Reasoning algebraically about loops. *Acta Informatica*, 36(4):295–334, 1999.

5. K. Claessen. Equinox, A New Theorem Prover for Full First-Order Logic with Equality. 2005.
6. E. Cohen. 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*, page 5, 2001.
7. E. Contejean, P. Courtieu, J. Forest, O. Pons, and X. Urbain. Certification of automated termination proofs. In B. Konev and F. Wolter, editors, *Frontiers of Combining Systems, 6th International Symposium*, volume 4720 of *Lecture Notes in Artificial Intelligence*, pages 148–162. Springer, 2007.
8. H. de Nivelle and J. Meng. Geometric Resolution: A Proof Procedure Based on Finite Model Search. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 303–317. Springer, 2006.
9. W.-P. de Roever and K. Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*. Cambridge University Press, 2001.
10. J. Desharnais and G. Struth. Domain axioms for a family of near-semirings. In J. Meserguer and G. Roşu, editors, *Algebraic Methodology and Software Technology (AMAST 2008)*, volume 5140 of *Lecture Notes in Computer Science*, pages 330–345. Springer, 2008.
11. J. Desharnais and G. Struth. Modal semirings revisited. In P. Audebaud and C. Paulin-Mohrig, editors, *Mathematics of Program Construction (MPC 2008)*, volume 5133 of *Lecture Notes in Computer Science*, pages 360–387. Springer, 2008.
12. M. Ebert and G. Struth. Diagram chase in relational system development. *Electronic Notes in Theoretical Computer Science*, 127:87–105, 2005.
13. P. Höfner. Automated reasoning for hybrid systems — Two case studies. In R. Berghammer, B. Möller, and G. Struth, editors, *Relations and Kleene Algebra in Computer Science*, volume 4988 of *Lecture Notes in Computer Science*, pages 191–205. Springer, 2008.
14. P. Höfner and G. Struth. Automated reasoning in Kleene algebra. In F. Pfenning, editor, *Automated Deduction (CADE 21)*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 279–294. Springer, 2007.
15. P. Höfner and G. Struth. Can refinement be automated? *Electronic Notes in Theoretical Computer Science*, 201:197–222, 2007.
16. P. Höfner and G. Struth. Algebraic reasoning with Prover9 (proof database). <`http://www.dcs.shef.ac.uk/~georg/ka`>, 10 September 2008.
17. P. Höfner and G. Struth. On automating the calculus of relations. In A. Armando, P. Baumgartner, and G. Dowek, editors, *Automated Deduction (IJCAR 2008)*, volume 5196 of *Lecture Notes in Computer Science*, pages 50–66. Springer, 2008.
18. J. Hurd. First-Order Proof Tactics in Higher-Order Logic Theorem Provers. In M. Archer, B. Di Vito, and C. Munoz, editors, *Proceedings of the 1st International Workshop on Design and Application of Strategies/Tactics in Higher Order Logics*, number NASA/CP-2003-212448 in NASA Technical Reports, pages 56–68, 2003.
19. K. Korovin. Implementing an Instantiation-based Theorem Prover for First-order Logic. In C. Benzmüller, B. Fischer, and G. Sutcliffe, editors, *Proceedings of the 6th International Workshop on the Implementation of Logics*, number 212 in CEUR Workshop Proceedings, pages 63–63, 2006.
20. D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994.
21. W.W. McCune. Otter 3.3 Reference Manual. Technical Report ANL/MSC-TM-263, Argonne National Laboratory, Argonne, USA, 2003.
22. W.W. McCune. Prover9 and Mace4. <`http://www.cs.unm.edu/~mccune/prover9`>, 10 September 2008.
23. A. K. McIver, C. Gonzalia, E. Cohen, and C. C. Morgan. Using probabilistic Kleene algebra pKA for protocol verification. *J. Logic and Algebraic Programming*, 76(1):90–111, 2008.
24. J. Otten and W. Bibel. leanCoP: Lean Connection-Based Theorem Proving. *Journal of Symbolic Computation*, 36(1-2):139–161, 2003.
25. F.J. Pelletier, G. Sutcliffe, and C.B. Suttner. The Development of CASC. *AI Communications*, 15(2-3):79–90, 2002.
26. A. Podelski and A. Rybalchenko. Transition invariants. In *LICS '04: Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science*, pages 32–41. IEEE Computer Society, 2004.

27. S. Ranise and C. Tinelli. Satisfiability Modulo Theories. *Trends and Controversies - IEEE Intelligent Systems Magazine*, 21(6):71–81, 2006.
28. A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.
29. S. Schulz. E: A Brainiac Theorem Prover. *AI Communications*, 15(2-3):111–126, 2002.
30. G. Struth. Deriving focused calculi for transitive relations. In A. Middeldorp, editor, *Rewriting Techniques and Applications, 12th International Conference*, volume 2051 of *Lecture Notes in Computer Science*, pages 291–305. Springer, 2001.
31. G. Struth. Abstract abstract reduction. *Journal of Logic and Algebraic Programming*, 66(2):239–270, 2006.
32. G. Struth. Reasoning automatically about termination and refinement. In S. Ranise, editor, *6th International Workshop on First-Order Theorem Proving*, Technical Report ULCS-07-018, Department of Computer Science, pages 36–51. University of Liverpool, 2007.
33. G. Struth. Modal tools for separation and refinement. *Electronic Notes in Theoretical Computer Science*, 214C:81–101, 2008.
34. G. Sutcliffe. Semantic Derivation Verification. *International Journal on Artificial Intelligence Tools*, 15(6):1053–1070, 2006.
35. G. Sutcliffe. The CADE-21 Automated Theorem Proving Competition. *AI Communications*, 21(1):71–82, 2008.
36. G. Sutcliffe and Y. Puzis. SRASS - A Semantic Relevance Axiom Selection System. In F. Pfenning, editor, *Proceedings of the 21st International Conference on Automated Deduction*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 295–310. Springer, 2007.
37. G. Sutcliffe and C. Suttner. The State of CASC. *AI Communications*, 19(1):35–48, 2006.
38. G. Sutcliffe and C.B. Suttner. The TPTP problem library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.
39. G. Sutcliffe and C.B. Suttner. Evaluating General Purpose Automated Theorem Proving Systems. *Artificial Intelligence*, 131(1-2):39–54, 2001.
40. Terese, editor. *Term Rewriting Systems*. Cambridge University Press, 2003.
41. J. Urban. MaLARea: a Metasystem for Automated Reasoning in Large Theories. In J. Urban, G. Sutcliffe, and S. Schulz, editors, *Proceedings of the CADE-21 Workshop on Empirically Successful Automated Reasoning in Large Theories*, pages 45–58, 2007.
42. J. von Wright. From Kleene algebra to refinement algebra. In E. A. Boiten and B. Möller, editors, *Mathematics of Program Construction*, volume 2386 of *Lecture Notes in Computer Science*, pages 233–262. Springer, 2002.
43. J. von Wright. Towards a refinement algebra. *Science of Computer Programming*, 51(1-2):23–45, 2004.
44. C. Weidenbach, B. Gaede, and G. Rock. SPASS and FLOTTER. In M. McRobbie and J.K. Slaney, editors, *Proceedings of the 13th International Conference on Automated Deduction*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 141–145. Springer, 1996.
45. C. Weidenbach, R. Schmidt, T. Hillenbrand, R. Rusev, and D. Topic. SPASS Version 3.0. In F. Pfenning, editor, *Proceedings of the 21st International Conference on Automated Deduction*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 514–520. Springer, 2007.