Peter Höfner
Annabelle McIver
Georg Struth (eds.)

1st Workshop
on
Automated Theory Engineering

Wrocław, Poland
July 31, 2011

Proceedings

Simplifying Pointer Kleene Algebra

Han-Hing Dang* Bernhard Möller
Institut für Informatik, Universität Augsburg
D-86135 Augsburg, Germany
{dang,moeller}@informatik.uni-augsburg.de

Abstract

Pointer Kleene algebra has proved to be a useful abstraction for reasoning about reachability properties and correctly deriving pointer algorithms. Unfortunately it comes with a complex set of operations and defining (in)equations which exacerbates its practicability with automated theorem proving systems but also its use by theory developers. Therefore we provide an easier access to this approach by simpler axioms and laws which also are more amenable to automatic theorem proving systems.

1 Introduction

Nowadays many first-order automated theorem proving systems are powerful enough and hence applicable to a large variety of verification tasks. Algebraic approaches especially have already been successfully treated by automatic reasoning in different application ranges [7, 8, 9, 16]. This work concentrates on an algebraic calculus for the derivation of abstract properties of pointer structures, namely pointer Kleene algebra [5]. This approach has proved to be an appropriate abstraction for reasoning about pointer structures and the deduction of various pointer algorithms [12, 13, 14]. Moreover the algebra enables by its (in)equation-based axioms the use of automated theorem provers for deriving and proving assertions about pointer structures, e.g., concerning reachability, allocation and destructive updates. When analysing reachability, projections are used to constrain the links of interest. For instance, in a doubly linked list the set of links in forward and backward direction each must be cycle-free, whereas, of course, the overall link set is cyclic. Unfortunately the existing pointer algebra turns out to be difficult to handle and to hinder automation. Therefore we provide a simplified version of pointer Kleene algebra which, additionally, is more suitable for automation.

The paper is organised as follows: In Section 2, we define a concrete graph-based model of pointer structures and illustrate the operations of pointer algebra in Section 3. Moreover some basics are given on which these specific operations build. Section 4 then gives more suitable axioms for automated deduction systems which are also easier to understand than the original theory. We conclude with an outlook on further applications of this approach in Section 5.

2 A Matrix Model of Pointer Kleene Algebra

We start by giving a concrete model of pointer Kleene algebra that shows how to treat various typical concepts of pointer structures algebraically. This model is based on matrices and represents the graph structure of storage with pointer links between records or objects. In [4] the model is also called a *fuzzy model* because it builds on some notions from the theory of fuzzy sets.

We assume a finite set L of edge labels of a graph with the known operations $\cup, \cap$. Moreover, let V be a finite set of vertices. Then the carrier set $\mathcal{P}(L)^{V \times V}$ of the model consists of matrices

*Research was partially sponsored by DFG Project ALGSEP — Algebraic Calculi for Separation Logic

with vertices in V as indices and subsets of L as entries. An entry $A(x,y) = M \subseteq L$ in a matrix A means that in the represented graph there are $|M|$ edges from vertex x to vertex y, labelled with the elements of M. The complete algebraic structure is $(\mathcal{P}(L)^{V\times V}, \cup, ;, 0, 1, \top)$ with greatest element $\top$ and operators, for arbitrary matrices $A, B \in \mathcal{P}(L)^{V\times V}$ and $x, y \in V$,

$$\begin{array}{rcl} (A \cup B)(x,y) & =_{df} & A(x,y) \cup B(x,y) \ , \\ (A;B)(x,y) & =_{df} & \bigcup_{z\in V}(A(x,z) \cap B(z,y)) \ , \\ \top(x,y) & =_{df} & L \ , \\ 0(x,y) & =_{df} & \emptyset \ , \\ 1(x,y) & =_{df} & \begin{cases} L & \text{if } x = y \ , \\ \emptyset & \text{otherwise} \ . \end{cases} \end{array}$$

As an example consider a tree structure with $L =_{df} \{\mathsf{left}, \mathsf{right}, \mathsf{val}\}$. Clearly, the labels left and right represent links to the left and right son (if present) of a node in a tree, respectively. The destination of the label val is interpreted as the value of such a node. We will use "label" and "link" as synonyms in the following. Figure 1 depicts a sample matrix and its corresponding labelled directed graph, i.e., a tree with $V =_{df} \{v_i, c_i : 1 \le i \le 3\}$.

	v_1	v_2	v_3	c_1	c_2	c_3
v_1	$\emptyset$	{left}	{right}	{val}	$\emptyset$	$\emptyset$
v_2	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	{val}	$\emptyset$
v_3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	{val}
c_1	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
c_2	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
c_3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

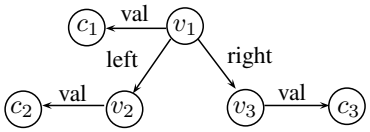

Figure 1: Example matrix and the corresponding labelled graph

Multiplication with the $\top$ matrix turns out to be quite useful for a number of issues. Left-multiplying a matrix A with $\top$ produces a matrix where each column contains the union of the labels in the corresponding column of A. Dually, right-multiplying a matrix A with $\top$ produces a matrix where each row contains the union of the labels in the corresponding row of A. Finally, multiplying A with $\top$ from both sides gives a constant matrix where each entry consists of the union of all labels occurring in A.

$$\top \cdot \begin{pmatrix} L_{11} & \cdots & L_{1n} \\ \vdots & \cdots & \vdots \\ L_{n1} & \cdots & L_{nn} \end{pmatrix} = \begin{pmatrix} L_1 & \cdots & L_n \\ \vdots & \cdots & \vdots \\ L_1 & \cdots & L_n \end{pmatrix} \qquad \begin{pmatrix} L_{11} & \cdots & L_{1n} \\ \vdots & \cdots & \vdots \\ L_{n1} & \cdots & L_{nn} \end{pmatrix} \cdot \top = \begin{pmatrix} M_1 & \cdots & M_1 \\ \vdots & \cdots & \vdots \\ M_n & \cdots & M_n \end{pmatrix} \tag{1}$$

where $L_i = \bigcup_{1\le j\le n} L_{ij}$ and $M_j = \bigcup_{1\le i\le n} L_{ij}$. This entails

$$\top \cdot \begin{pmatrix} L_{11} & \cdots & L_{1n} \\ \vdots & \cdots & \vdots \\ L_{n1} & \cdots & L_{nn} \end{pmatrix} \cdot \top \ = \ \begin{pmatrix} M & \cdots & M \\ \vdots & \cdots & \vdots \\ M & \cdots & M \end{pmatrix} \tag{2}$$

where $M = \bigcup\limits_{1\le j\le n} \bigcup\limits_{1\le i\le n} L_{ij}$. Such matrices can be used to represent sets of labels in the model.

3 Basics and the Original Approach

Abstracting from the matrix model of pointer Kleene algebra we now define some fundamental notions and explain special operations of the algebra. The basic algebraic structure of pointer Kleene algebra in the sense of [4] is a *quantale* [1, 15]. This is, first, an *idempotent semiring*, i.e., a structure $(S, +, \cdot, 0, 1)$, where $+$ abstracts $\cup$ and $\cdot$ abstracts ; , and the following is required:

- $(S, +, 0)$ forms an idempotent commutative monoid and $(S, \cdot, 1)$ a monoid. This means, for arbitrary $x, y, z \in S$,

$$\begin{array}{c} x + (y + z) = (x + y) + z \ , \quad x + y = y + x \ , \quad x + 0 = 0 \ , \quad x + x = x \ , \\ x \cdot (y \cdot z) = (x \cdot y) \cdot z \ , \qquad x \cdot 1 = x \ , \qquad 1 \cdot x = x \ . \end{array}$$

- Moreover, $\cdot$ has to distribute over $+$ and 0 has to be a multiplicative annihilator, i.e., for arbitrary $x, y, z \in S$,

$$\begin{array}{c} x \cdot (y + z) = (x \cdot y) + (x \cdot z) \ , \quad (x + y) \cdot z = (x \cdot z) + (y \cdot z) \ , \\ x \cdot 0 = 0 \ , \qquad\qquad 0 \cdot x = 0 \ . \end{array}$$

The natural order $\leq$ on an idempotent semiring is defined by $x \leq y \Leftrightarrow_{df} x + y = y$. It is easily verified that the matrix model indeed forms an idempotent semiring in which the natural order coincides with pointwise inclusion of matrices.

Second, to form a quantale, $(S, \leq)$ has to be a complete lattice and multiplication $\cdot$ has to distribute over arbitrary joins. By this, $+$ coincides with the binary supremum operator $\sqcup$. The binary infimum operator is denoted by $\sqcap$; in the model it coincides with the pointwise $\cap$ of matrices. The greatest element of the quantale is denoted by $\top$.

This structure is now enhanced to a *Kleene algebra* [10] by an iteration operator *, axiomatised by the following unfold and induction laws:

$$\begin{array}{l} 1 + x \cdot x^* = x^* \ , \quad x \cdot y + z \leq y \ \Rightarrow \ x^* \cdot z \leq y \ , \\ 1 + x^* \cdot x = x^* \ , \quad y \cdot x + z \leq y \ \Rightarrow \ z \cdot x^* \leq y \ . \end{array}$$

Besides this, the algebra considers special subidentities $p \leq 1$, called *tests*. Multiplication by a test therefore means restriction. Tests can also be seen as an algebraic counterpart of predicates and thus have to form a Boolean subalgebra. The defining property is therefore that a test must have a complement relative to 1, i.e., an element $\neg p$ that satisfies $p + \neg p = 1$ and $p \cdot \neg p = 0 = \neg p \cdot p$. In the matrix model tests are matrices that have non-empty entries at most on their main diagonal; multiplication of a matrix M with a test p means pointwise intersection of the rows or columns of M with the corresponding diagonal entry in p.

3.1 The Original Theory

An essential ingredient of pointer Kleene algebra is an operation that projects all entries of a given matrix to links of a subset $L' \subseteq L$. This is modelled by *scalars*[1] [4], which are tests $\alpha, \beta, \ldots$ that additionally commute with $\top$, i.e., $\alpha \cdot \top = \top \cdot \alpha$. Analogously to (1) and (2) these are diagonal matrices which are constant on the main diagonal. We will use the notation $L(\alpha)$ to denote the unique set of labels that a scalar α comes with. By this, a scalar α in the model corresponds to the matrix

$$\alpha(x, y) = \begin{cases} L(\alpha) & \text{if } x = y \ , \\ \emptyset & \text{otherwise} \ . \end{cases}$$

[1] The origin of this term lies in fuzzy relation theory and has similar behaviour as scalars in vector spaces.

It is immediate from the axioms that 0 and 1 are scalars.

Describing projection needs additional concepts. First, for arbitrary element x and scalar α,

$$\alpha\backslash x = x + \neg\alpha \cdot \top \ . \tag{3}$$

In the matrix model this adds the complement of $L(\alpha)$ to every entry in x. More abstractly, the operation is a special instance of the *left residual* defined by the Galois connection $x \leq y\backslash z \Leftrightarrow_{df} x \cdot y \leq z$; but this is of no further importance here.

The next concepts employed are the *completion* operator $\_^\uparrow$ and its dual $\_^\downarrow$, also known from the algebraic theory of fuzzy sets [18]. For arbitrary $x, y \in S$ they are axiomatised as follows.

$$\begin{array}{ll} x^\uparrow \leq y \ \Leftrightarrow \ x \leq y^\downarrow \ , & (x \cdot y^\downarrow)^\uparrow = x^\uparrow \cdot y^\downarrow \ , \\ \alpha \text{ is a scalar and } \alpha \neq 0 \text{ implies } \alpha^\uparrow = 1 \ , & (x^\downarrow \cdot y)^\uparrow = x^\downarrow \cdot y^\uparrow \ . \end{array} \tag{4}$$

In the matrix model they can be described for a matrix M as follows

$$M^\uparrow(x,y) = \begin{cases} L & \text{if } M(x,y) \neq \emptyset \ , \\ \emptyset & \text{otherwise} \ , \end{cases} \qquad M^\downarrow(x,y) = \begin{cases} L & \text{if } M(x,y) = L \ , \\ \emptyset & \text{otherwise} \ . \end{cases}$$

In both cases each node x is either totally linked or not linked at all with another node y. Such matrices containing only the entries $\emptyset$ and L are also called *crisp* [4]. They behave analogous to Boolean matrices where $\emptyset$ plays the role of 0 and L the one of 1. In the abstract algebra crisp elements are characterised by the equation $x^\uparrow = x$. In particular, $M^\uparrow$ maps a matrix M to the least crisp matrix containing it, while $M^\downarrow$ maps M to the greatest crisp matrix it contains.

Based on these operations and the particular elements of the algebra, *projections* $P_\alpha(\_)$ to label sets $L(\alpha)$ represented by a scalar α can be abstractly defined for arbitrary $x \in S$ by

$$P_\alpha(x) \ =_{df} \ \alpha \cdot (\alpha\backslash x)^\downarrow \ . \tag{5}$$

In the matrix model, projections w.r.t. scalars α are used to restrict each entry of a matrix exactly to $L(\alpha)$. As an example consider the resulting matrix of the following projection with $L(\alpha) = \{\mathsf{left}, \mathsf{right}\}$

$$P_{\{\mathsf{left},\mathsf{right}\}}\left(\begin{pmatrix} \emptyset & \{\mathsf{right}\} & \{\mathsf{left}\} & \{\mathsf{val}\} \\ \emptyset & \emptyset & \emptyset & \{\mathsf{left},\mathsf{right}\} \\ \{\mathsf{val}\} & \{\mathsf{left},\mathsf{right}\} & \emptyset & \emptyset \\ \emptyset & \{\mathsf{val}\} & \emptyset & \emptyset \end{pmatrix}\right) = \begin{pmatrix} \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{\mathsf{left},\mathsf{right}\} \\ \emptyset & \{\mathsf{left},\mathsf{right}\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix} .$$

To see that this is achieved by Equation (5) consider first the term $\alpha\backslash x$ in the matrix model. This can be rewritten using Equation (3) into the term $x + \neg\alpha \cdot \top$. Hence residuals add to each entry of x all labels not in $L(\alpha)$. The operation $\_^\downarrow$ is then used to keep only those entries in $\alpha\backslash x$ that contain the full set of labels L while all other label sets will be mapped to $\emptyset$. Finally the multiplication with the scalar α again reduces all remaining L entries to $L(\alpha)$.

Finally we turn to the most important operator of pointer Kleene algebra. It calculates all reachable nodes from a given set of nodes. The definition uses domain and codomain operations $\ulcorner\_$ and $\urcorner$. They are characterised by the following equations, for arbitrary element x and test p,

$$x \leq \ulcorner x \cdot x \ , \qquad \ulcorner(p \cdot x) \leq p \ , \qquad\qquad x \leq x \cdot \urcorner x \ , \qquad (x \cdot p)\urcorner \leq p \ .$$

We note that a third pair of axioms, the modality laws $\ulcorner(x \cdot y) = \ulcorner(x \cdot \ulcorner y)$ and $(x \cdot y)\urcorner = (x \cdot \urcorner y)\urcorner$, is omitted here, because they are implied by another special property that holds for the matrix

model and also the algebra of binary relations. For domain, it reads $x \cdot \top = \lceil x \cdot \top$ and is called *subordination of domain*, since the direction $x \cdot \top \leq \lceil x \cdot \top$ follows from the above axioms, but the reverse one does not. This is equivalent [4] to $\lceil x = x \cdot \top \sqcap 1$, which we postulate as an additional axiom. This and the dual one for codomain then entail the above-mentioned modality laws.

Both operations are mappings from general elements to tests, i.e., the resulting matrices in the model are diagonal matrices. More concretely, the domain operation extracts for every vertex the set of labels on its outgoing edges while the codomain operator returns the labels on the incoming edges. As a simple example for domain consider

$$\lceil \begin{pmatrix} \emptyset & \{\mathsf{right}\} & \{\mathsf{left}\} \\ \emptyset & \emptyset & \{\mathsf{left},\mathsf{right}\} \\ \{\mathsf{val}\} & \{\mathsf{left},\mathsf{right}\} & \emptyset \end{pmatrix} = \begin{pmatrix} \{\mathsf{left},\mathsf{right}\} & \emptyset & \emptyset \\ \emptyset & \{\mathsf{left},\mathsf{right}\} & \emptyset \\ \emptyset & \emptyset & \{\mathsf{left},\mathsf{right},\mathsf{val}\} \end{pmatrix} .$$

In [4] reachability is now defined by a mapping that takes two arguments: One argument represents the set of starting nodes or addresses from which the reachable vertices are computed. The other argument represents the graph structure in which the reachability analysis takes place. Address sets are represented in this approach by crisp tests: an address belongs to the set represented by p iff the corresponding entry in the main diagonal of p is L. Now assume that m represents an address set. Then $reach(m, x)$ is defined using the iteration operator * by

$$reach(m, x) = (m \cdot (x^{\uparrow})^*)^{\urcorner} . \tag{6}$$

The mapping *reach* calculates a test representing the set of vertices that are reachable from m using arbitrarily labelled graph links of x. This "forgetting" of the concrete label sets is modelled by completing the graph using the completion operator $\_^{\uparrow}$. Sample properties one wants to prove about *reach* are

$$reach(m + n, x) = reach(m, x) + reach(n, x) , \qquad reach(reach(m, x), x) = reach(m, x) .$$

To restrict reachability analysis to a subset of labels, projections are combined with *reach* into the mapping $reach(m, P_\alpha(x))$ for a scalar α. By this, non-α links are deleted from x.

Finally, an important operation in connection with pointer structures is *overriding* one structure x by another one y, denoted $y|x =_{df} y + \neg\lceil y \cdot x$. Here entries of x for which also y provides a definition are replaced by these and only the part of x not affected by y is preserved. This operator enjoys a rich set of derived algebraic properties. For instance, it is interesting to see in which way an overwriting affect reachability. One sample law for this is

$$\lceil y^{\uparrow} \sqcap reach(m, x) = 0 \Rightarrow reach(m, y|x) = reach(m, x) ,$$

i.e., when the domain of the overriding structure is not reachable from the overridden one it does not affect reachability.

3.2 A Discussion on Automation

One sees that it is very onerous to define the domain specific operations of pointer Kleene algebra from the basic operations. For example projections already include special subidentities of the algebra, residuals and completion and its dual, where each operator itself comes with lots of (in)equations defining behaviour. Furthermore by Equation (6) *reach* also uses crisp subidentities and moreover includes the $\_^{\uparrow}$, iteration * and the codomain operation. The inclusion of that many axioms often irritates the proof systems and additionally increases the search space.

A naive encoding of these operations in the first-order automated theorem proving system[2] Prover9 [11] already revealed that only a small set of basic properties for projections and reachability calculations could be derived automatically. We used this theorem prover since it performs best for automated reasoning tasks within the presented basic algebraic structures [2]. Moreover, it comes with the useful counterexample search engine Mace4 and outputs semi-readable proofs, which often provide helpful hints. Of course, any other first-order ATP system could also be used with the abstract algebraic approach, e.g., through the TPTP problem library [17].

The resulting ineffective automation could be due to the indirect axiomatisation of $\_^{\downarrow}$ through $\_^{\uparrow}$ by a Galois connection (cf. Definition (4)). In particular, deriving theorems for $\_^{\downarrow}$ will often require to show results for $\_^{\uparrow}$ and vice versa. Furthermore, encoding subtypes as tests and scalars by predicates may be another hindrance to a simple treatment of the axioms by ATP systems. Such an encoding is also inappropriate for ATP systems like Waldmeister [6] which works completely equation-based.

Moreover, using the Galois characterisation of residuals instead of the explicit characterisation in Equation (3) seems to additionally exacerbate the proof search. However, including that characterisation does not seem to simplify the proof search significantly. Therefore we also got a similar result with the application of ATP system when reasoning about restricted reachability.

Finally the given axiomatisations of the specific operators for this particular domain are also difficult to grasp and handle for theory developers due to their complexity. Therefore, in the next section we provide a simpler approach to pointer Kleene algebra which is easier to understand and more amenable to ATP systems.

4 A Simpler Theory for Pointer Kleene Algebra

The preceding section has shown that the original pointer algebra uses quite a number of concepts and ingredients. The present section is intended to show that one can do with a smaller toolbox which also is more amenable to automation. As a first simplification we drop the assumption that address sets need to be interpreted by crisp elements. Plain test elements also suffice to represent source nodes for $reach(p, x)$ since x is by definition already completed.

4.1 Projection

We continue with the notion of projecting a graph to a subgraph that is restricted to a set of labels. For this we first want to find representatives for sets of labels in our algebra.

It is clear that constant matrices and sets of labels are in one-to-one correspondence. By intersecting a constant matrix with the identity matrix one obtains a test which in the main diagonal contains the represented set M of labels, see (1). Multiplying another matrix A with this test from either side intersects all entries in A with M and hence projects A to the label set M. The considered test can also be got by taking the domain of the resulting matrix. Note that neither scalars nor residuals nor the operator $\_^{\downarrow}$ are involved here.

The difference of the just described way to project matrices and projections $P_\alpha(x)$ can be made clear in the example given after Equation (5). By our approach only the $\{\mathsf{val}\}$ entries of the original matrix will be deleted while single $\{\mathsf{left}\}$ and $\{\mathsf{right}\}$ entries remain. It is reasonable also to consider such entries in reachability calculations.

[2]We abbreviate the term "automated theorem proving system" to ATP system in the following.

4.2 Domain and Codomain

As a further simplification, plain test elements can be represented in the algebra by domain or codomain elements. In particular, such elements form Boolean subalgebras, resp. We give their axiomatisation through $a(x)$ and $ac(x)$ which are the complements of the domain and codomain of x, resp. From these domain and codomain can be retrieved as $\lceil x = a(a(x))$ and $\overline{x}^{\urcorner} = ac(ac(x))$. The axioms read as follows:

$$\begin{array}{cc} a(x) \cdot x = 0 \ , & x \cdot ac(x) = 0 \ , \\ a(a(x)) + a(x) = 1 \ , & ac(ac(x)) + ac(x) = 1 \ , \\ a(x \cdot y) \leq a(x \cdot a(a(y))) \ , & ac(x \cdot y) \leq ac(ac(ac(x)) \cdot y) \ . \end{array}$$

The idea with this approach is to avoid explicit subsorting, i.e., introducing the set of tests as a sort of its own, say by using predicates that assert that certain elements are tests, and to characterise the tests implicitly as the images of the antidomain/anticodomain operators. The axioms entail that those images coincide and form a Boolean algebra with $+$ and $\cdot$ as join and meet, resp.

The given characterisation seems to be still difficult to handle for ATP systems in that form. Often a lot of standard Boolean algebra properties have to be derived first. Therefore we propose an equivalent but more "efficient" axiomatisation at the end of the next section.

4.3 Completion

Next, we turn to the completion operator $\_^{\uparrow}$ which is useful in analysing link-independent reachability in graphs. Rather than axiomatising it indirectly through a Galois connection we characterise it jointly with its complement $\_^{\overline{\uparrow}}$ similarly to domain and antidomain in Section 4.2. In particular, we axiomatise $x^{\overline{\uparrow}}$ as a left annihilator of x w.r.t. $\sqcap$, paralleling the statement that $a(x)$ is a left annihilator for x w.r.t. composition $\cdot$.

The axioms show some similarity to the domain/antidomain ones, but also substantial differences on which we will comment below. However, we still have, analogously to $\lceil x = a(a(x))$, that $x^{\uparrow} = (x^{\overline{\uparrow}})^{\overline{\uparrow}}$; for better readability we use this as an abbreviation in the axioms:

$$\begin{array}{ccc} 1^{\uparrow} \leq 1 \ , & x^{\overline{\uparrow}} \sqcap x \leq 0 \ , & \top \leq 0^{\overline{\uparrow}} \ , \\ x^{\overline{\uparrow}} \sqcap y^{\overline{\uparrow}} = (x + y)^{\overline{\uparrow}} \ , & & x^{\overline{\uparrow}} + y^{\overline{\uparrow}} = (x^{\uparrow} \sqcap y^{\uparrow})^{\overline{\uparrow}} \ , \\ (x \cdot y^{\uparrow})^{\uparrow} = x^{\uparrow} \cdot y^{\uparrow} \ , & & (x^{\uparrow} \cdot y)^{\uparrow} = x^{\uparrow} \cdot y^{\uparrow} \ . \end{array}$$

Notice that the inequations can be strengthened to equations. The most striking difference to antidomain are the De-Morgan-like axioms and the axioms concerning multiplication. We cannot use the axiom $x \cdot y^{\overline{\uparrow}} \leq (x \cdot y^{\uparrow})^{\overline{\uparrow}}$ instead because it is not valid in the matrix model. Therefore the De Morgan laws do not follow but rather have to be put as additional axioms. They clearly state that the image of $\_^{\overline{\uparrow}}$ is closed under $\sqcap$ and $+$.

We list a number of useful consequences of the axioms; they are all very quickly shown by Prover9. A sample input file can be found in the appendix.

$$\begin{array}{ccc} x \leq x^{\uparrow} \ , & \top^{\uparrow} = \top \ , & 0^{\uparrow} = 0 \ , \\ (x^{\overline{\uparrow}})^{\uparrow} = x^{\overline{\uparrow}} \ , & \top^{\overline{\uparrow}} = 0 \ , & 0^{\overline{\uparrow}} = \top \ , \\ (x^{\overline{\uparrow}} + y^{\overline{\uparrow}})^{\overline{\uparrow}} = x^{\uparrow} \sqcap y^{\uparrow} \ , & & (x^{\overline{\uparrow}} \sqcap y^{\overline{\uparrow}})^{\overline{\uparrow}} = x^{\uparrow} + y^{\uparrow} \ , \\ (x^{\uparrow} + y^{\uparrow})^{\overline{\uparrow}} = x^{\overline{\uparrow}} \sqcap y^{\overline{\uparrow}} \ , & & (x^{\uparrow} \sqcap y^{\uparrow})^{\overline{\uparrow}} = x^{\overline{\uparrow}} + y^{\overline{\uparrow}} \ , \\ x \leq y \Rightarrow y^{\overline{\uparrow}} \leq x^{\overline{\uparrow}} \ , & & x \leq y \Rightarrow x^{\uparrow} \leq y^{\uparrow} \ , \\ x^{\uparrow} = 0 \Leftrightarrow x = 0 \ , & x^{\overline{\uparrow}} = \top \Leftrightarrow x = 0 \ , & x^{\uparrow} = 0 \Leftrightarrow x = 0 \ , \\ (x^{\uparrow})^{\uparrow} = x^{\uparrow} \ , & (x^{\uparrow} \cdot y^{\uparrow})^{\uparrow} = x^{\uparrow} \cdot y^{\uparrow} \ , & (x + y)^{\uparrow} = x^{\uparrow} + y^{\uparrow} \ . \end{array}$$

In particular, the image of $\_^{\overline{\uparrow}}$ and hence of $\_^{\uparrow}$, i.e., the set of crisp elements, forms a Boolean algebra. Moreover, $\_^{\uparrow}$ is a closure operator. By general results therefore $\_^{\uparrow}$ preserves all existing suprema and, in a complete lattice, has an upper adjoint, which of course is $\_^{\downarrow}$. To axiomatise $\_^{\downarrow}$ we can use the Galois adjunction $x^{\uparrow} \leq y \Leftrightarrow x \leq y^{\downarrow}$. This entails standard laws as e.g.

$$(x \cdot y^{\downarrow})^{\uparrow} = x^{\uparrow} \cdot y^{\downarrow} \ , \quad (x^{\downarrow} \cdot y)^{\uparrow} = x^{\uparrow} \cdot y^{\downarrow} \ , \quad (x^{\downarrow})^{\uparrow} \leq x \ , \quad x \leq (x^{\uparrow})^{\downarrow} \ .$$

With the help of $\_^{\downarrow}$ one can show that the image of $\_^{\overline{\uparrow}}$ is also closed under the iteration *. A last speciality concerns the domain/codomain operation. By the subordination axiom for domain it can be verified that $\ulcorner(x^{\uparrow}) = (\ulcorner x)^{\uparrow}$.

Altogether we retrieve Ehm's result that, in a semiring with domain, the image of $\_^{\overline{\uparrow}}$, forms again a Kleene algebra with domain. Extending that, our axiomatisation yields that this algebra is even Boolean.

Inspired by the above axiomatisation, we now present a new axiomatisation of antidomain and anticodomain that explicitly states De-Morgan-like dualities to facilitate reasoning.

$$\begin{array}{cc} a(x) \cdot x = 0 \ , \quad a(0) = 1 \ , & x \cdot ac(x) = 0 \ , \quad ac(0) = 1 \ , \\ a(x) \cdot a(y) = a(x + y) \ , & ac(x) \cdot ac(y) = ac(x + y) \ , \\ a(x) + a(y) = a(a(a(x)) \cdot a(a(y))) \ , & ac(x) + ac(y) = ac(ac(ac(x)) \cdot ac(ac(y))) \ , \\ a(x \cdot y) \leq a(x \cdot a(a(y))) \ , & ac(x \cdot y) \leq ac(ac(ac(x)) \cdot y) \ . \end{array}$$

Using Prover9 we have shown that this axiomatisation is equivalent to the standard axiomatisation of antidomain/anticodomain. And indeed, the automatic proofs of the Boolean algebra properties of tests as well as of the mentioned properties of *reach* are much faster with it.

5 Outlook

This work provides a more suitable axiomatisation of special operations used in pointer Kleene algebra. These axioms are more applicable for ATP systems since less operations and less subtypes of the algebra has to be considered. Moreover the (in)equations of our approach enables a simpler encoding of the algebra for ATP systems due to explicit subsorting. This will also partially include the usage of ATP systems as Waldmeister for such particular problem domains.

It can be seen that the axiomatisations of antidomain (or anticodomain) and anticompletion are very similar. This motivates to further abstract from the concrete involved operations and to define a restriction algebra that replays general derivations.

Moreover, this approach will be used to characterise sharing and sharing patterns in pointer structures within an algebraic approach to separation logic [3]. This will include the ability to reason algebraically about reachability in abstractly defined data structures.

References

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

[2] H.-H. Dang and P. Höfner. First-order theorem prover evaluation w.r.t. relation- and Kleene algebra. In R. Berghammer, B. Möller, and G. Struth, editors, *Relations and Kleene Algebra in Computer Science — PhD Programme at RelMiCS 10/AKA 05*, number 2008-04 in Technical Report, pages 48–52. Institut für Informatik, Universität Augsburg, 2008.

[3] H.-H. Dang, P. Höfner, and B. Möller. Algebraic separation logic. *Journal of Logic and Algebraic Programming*, 80(6):221–247, 2011.

[4] T. Ehm. *The Kleene Algebra of Nested Pointer Structures: Theory and Applications*. PhD thesis, Fakultät für Angewandte Informatik, Universität Augsburg, 2003.

[5] T. Ehm. Pointer Kleene algebra. In R. Berghammer, B. Möller, and G. Struth, editors, *RelMiCS*, volume 3051 of *Lecture Notes in Computer Science*, pages 99–111. Springer, 2004.

[6] T. Hillenbrand, A. Buch, R. Vogt, and B. Löchner. WALDMEISTER - High-Performance Equational Deduction. *Journal of Automated Reasoning*, 18:265–270, 1997.

[7] 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.

[8] P. Höfner and G. Struth. On automating the calculus of relations. In A. Armando, P. Baumgartner, and G. Dowek, editors, *Automated Reasoning (IJCAR 2008)*, volume 5159 of *Lecture Notes in Computer Science*, pages 50–66. Springer, 2008.

[9] P. Höfner, G. Struth, and G. Sutcliffe. Automated verification of refinement laws. *Annals of Mathematics and Artificial Intelligence*, 55:35–62, February 2009.

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

[11] W. McCune. Prover9 and Mace4. <http://www.cs.unm.edu/~mccune/prover9>. (accessed July 26, 2011).

[12] B. Möller. Some applications of pointer algebra. In M. Broy, editor, *Programming and Mathematical Method*, number 88 in NATO ASI Series, Series F: Computer and Systems Sciences, pages 123–155. Springer, 1992.

[13] B. Möller. Calculating with pointer structures. In *Proceedings of the IFIP TC2/WG 2.1 International Workshop on Algorithmic Languages and Calculi*, pages 24–48. Chapman & Hall, 1997.

[14] B. Möller. Calculating with acyclic and cyclic lists. *Information Sciences*, 119(3-4):135–154, 1999.

[15] K. I. Rosenthal. *Quantales and their Applications*, volume 234 of *Pitman Research Notes in Mathematics Series*. Longman Scientific & Technical, 1990.

[16] G. Struth. Reasoning automatically about termination and refinement. In S. Ranise, editor, *6th International Workshop on First-Order Theorem Proving*, volume Technical Report ULCS-07-018, Department of Computer Science, pages 36–51. University of Liverpool, 2007.

[17] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[18] M. Winter. A new algebraic approach to L-fuzzy relations convenient to study crispness. *Information Sciences*, 139(3-4):233–252, 2001.

A Prover9 Encoding of Pointer Kleene Algebra

```
op(450, infix, "+").   % Addition
op(440, infix, ";").   % Multiplication
op(420, infix, "^").   % Meet
op(400, postfix, "*"). % Iteration
op(410, prefix, "@").  % Antidomain
op(410, prefix, "!").  % Anticodomain
op(410, prefix, "?").  % Anticompletion

% --- Additive commutative and idempotent monoid
x+(y+z) = (x+y)+z.
x+y = y+x.
x+0 = x.
x+x = x.
% --- Order
x <= y <-> x+y = y.
% --- Definition of top
x <= T.
```

```
% --- Multiplicative monoid
x;(y;z) = (x;y);z.
1;x = x.
x;1 = x.

% --- Distributivity laws
x;(y+z) = x;y + x;z.
(x+y);z = x;z + y;z.

% ---Annihilation
0;x = 0.
x;0 = 0.

% --- Definition of meet
(x<=y & x<=z) <-> x <= y^z.
x^(y+z) = x^y + x^z.

% --- Definition of domain
@0 = 1.
@x;x = 0.
@x;@y = @(x+y).
@x+@y = @(@@x;@@y).
@(x;y) = @(x;@@y).
@@x = (x;T) ^ 1. % --- Subordination of domain

% --- Definition of anticodomain
!0 = 1.
x;!x = 0.
!x;!y = !(x+y).
!x+!y = !(!!x;!!y).
!(x;y) = !(!!x;y).
!!x = (T;x) ^ 1. % --- Subordination of codomain

% --- Definition of completion
??1 = 1.
?0 = T.
?x^x = 0.
?x ^ ?y = ?(x + y).
?x + ?y = ?(??x ^ ??y).
??(x ; ??y) = ??x ; ??y.
??(??x ; y) = ??x ; ??y.

% --- Iteration - Unfold laws
1 + x ; x* = x*.
1 + x* ; x = x*.
% --- Iteration - Induction laws
x;y + z <= y -> x* ; z <= y.
y;x + z <= y -> z ; x* <= y.

% --- Reachability
reach(x,y) = !!(@@x;(??y)*).
% --- Projection
P(x,y) = @@((T;x);T);y.
```