Towards algebraic separation logic

Han-Hing Dang, Peter Höfner, Bernhard Möller

Angaben zur Veröffentlichung / Publication details:

Dang, Han-Hing, Peter Höfner, and Bernhard Möller. 2009. "Towards algebraic separation logic." *Lecture Notes in Computer Science* 5827: 59–72.
https://doi.org/10.1007/978-3-642-04639-1_5.

Towards Algebraic Separation Logic

Han-Hing Dang, Peter Höfner, and Bernhard Möller

Institut für Informatik, Universität Augsburg, D-86135 Augsburg, Germany
{h.dang,hoefner,moeller}@informatik.uni-augsburg.de

Abstract. We present an algebraic approach to separation logic. In particular, we give algebraic characterisations for all constructs of separation logic like assertions and commands. The algebraic view does not only yield new insights on separation logic but also shortens proofs and enables the use of automated theorem provers for verifying properties at a more abstract level.

1 Introduction

Two prominent formal methods for reasoning about the correctness of programs are Hoare logic [9] and the wp-calculus of Dijkstra [7]. These approaches, although foundational, lack expressiveness for shared mutable data structures, i.e., structures where updatable fields can be referenced from more than one point (e.g. [19]). To overcome this deficiency Reynolds, O'Hearn and others have developed *separation logic* for reasoning about such data structures [17]. Their approach extends Hoare logic by assertions to express separation within memory, both in the store and the heap. Furthermore the command language is enriched by some constructs that allow altering these separate ranges. The introduced mechanisms have been extended to concurrent programs that work on shared mutable data structures [16].

This paper presents an algebraic approach to separation logic. As a result many proofs become simpler while still being fully precise. Moreover, this places the topic into a more general context and therefore allows re-use of a large body of existing theory.

In Section 2 we recapitulate syntax and semantics of expressions in separation logic and give a formal definition of an update-operator for relations. Section 3 gives the semantics of assertions. After providing the algebraic background in Section 4, we shift from the validity semantics of separation logic to one based on the set of states satisfying an assertion. Abstracting from the set view yields an algebraic interpretation of assertions in the setting of semirings and quantales. In Section 6 we discuss special classes of assertions: pure assertions do not depend on the heap at all; intuitionistic assertions do not specify the heap exactly. We conclude with a short outlook.

2 Basic Definitions

Separation logic, as an extension of Hoare logic, does not only allow reasoning about explicitly named program variables, but also about anonymous variables

in dynamically allocated storage. Therefore a program state in separation logic consists of a *store* and a *heap*. In the remainder we consistently write s for stores and h for heaps.

To simplify the formal treatment, one defines values and addresses as integers, stores and heaps as partial functions from variables or addresses to values and states as pairs of stores and heaps:

$$\begin{aligned} \mathit{Values} &= \mathbb{Z} \ , \\ \{\mathsf{nil}\} \uplus \mathit{Addresses} &\subseteq \mathit{Values} \ , \\ \mathit{Stores} &= V \rightsquigarrow \mathit{Values} \ , \\ \mathit{Heaps} &= \mathit{Addresses} \rightsquigarrow \mathit{Values} \ , \\ \mathit{States} &= \mathit{Stores} \times \mathit{Heaps} \ , \end{aligned}$$

where V is the set of all variables, $\uplus$ denotes the disjoint union on sets and $M \rightsquigarrow N$ denotes the set of partial functions between M and N. With this definition, we slightly deviate from [19] where stores are defined as functions from variables to values of $\mathbb{Z}$ and heaps as functions from addresses into values of $\mathbb{Z}$, while addresses are also values of $\mathbb{Z}$.

The constant nil is a value for pointers that denotes an improper reference like null in programming languages like JAVA; by the above definitions, nil is not an address and hence heaps do not assign values to nil.

As usual we denote the domain of a relation (partial function) R by $\mathsf{dom}(R)$:

$$\mathsf{dom}(R) =_{df} \{x : \exists\, y.(x, y) \in R\} \ .$$

In particular, the domain of a store denotes all currently used program variables and $\mathsf{dom}(h)$ is the set of all currently allocated addresses on a heap h.

As in [14] and for later definitions we also need an *update* operator. It is used to model changes in stores and heaps. We will first give a definition and then explain its meaning.

Let R and S be partial functions. Then we define

$$R \,|\, S =_{df} R \cup \{(x, y) \mid (x, y) \in S \wedge x \notin \mathsf{dom}(R)\} \ . \tag{1}$$

The relation R updates the relation S with all possible pairs of R in such a way that $R \,|\, S$ is again a partial function. The domain of the right hand side of $\cup$ above is disjoint from that of R. In particular, $R \,|\, S$ can be seen as an extension of R to $\mathsf{dom}(R) \cup \mathsf{dom}(S)$. In later definitions we abbreviate an update $\{(x, v)\} \,|\, S$ on a single variable or address by omitting the set-braces and simply writing $(x, v) \,|\, S$ instead.

Expressions are used to denote values or Boolean conditions on stores and are independent of the heap, i.e., they only need the store component of a given state for their evaluation. Informally, *exp*-expressions are simple arithmetical expressions over variables and values, while *bexp*-expressions are Boolean expressions

over simple comparisons and true, false. Their syntax is given by

$$\begin{aligned} \mathit{var} &::= x \mid y \mid z \mid ... \\ \mathit{exp} &::= 0 \mid 1 \mid 2 \mid ... \mid \mathit{var} \mid \mathit{exp} \pm \mathit{exp} \mid ... \\ \mathit{bexp} &::= \mathsf{true} \mid \mathsf{false} \mid \mathit{exp} = \mathit{exp} \mid \mathit{exp} < \mathit{exp} \mid ... \end{aligned}$$

The semantics e^s of an expression e w.r.t. a store s is straightforward (assuming that all variables occurring in e are contained in $\mathsf{dom}(s)$). For example,

$$c^s = c \quad \forall\, c \in \mathbb{Z}\ , \quad \mathsf{true}^s = \mathsf{true} \quad \text{and} \quad \mathsf{false}^s = \mathsf{false}\ .$$

3 Assertions

Assertions play an important rôle in separation logic. They are used as predicates to describe the contents of heaps and stores and as pre- or postconditions in programs, like in Hoare logic:

$$\begin{aligned} \mathit{assert} ::= \mathit{bexp} &\mid \neg\, \mathit{assert} \mid \mathit{assert} \vee \mathit{assert} \mid \forall\, \mathit{var}.\, \mathit{assert} \mid \\ &\mathsf{emp} \mid \mathit{exp} \mapsto \mathit{exp} \mid \mathit{assert} * \mathit{assert} \mid \mathit{assert} \mathrel{-\!\!*} \mathit{assert}\ . \end{aligned}$$

In the remainder we consistently write p, q and r for assertions. Assertions are split into two parts: the "classical" ones from predicate logic and four new ones that express properties of the heap. The former are supplemented by the logical connectives $\wedge$, $\rightarrow$ and $\exists$ that are defined, as usual, by $p \wedge q =_{df} \neg(\neg p \vee \neg q)$, $p \rightarrow q =_{df} \neg p \vee q$ and $\exists\, v.\, p =_{df} \neg\forall\, v.\, \neg p$.

The semantics of assertions is given by the relation $s, h \models p$ of *satisfaction*. Informally, $s, h \models p$ holds if the state (s, h) satisfies the assertion p; an assertion p is called *valid* iff p holds in every state and, finally, p is *satisfiable* if there exists a state (s, h) which satisfies p. The semantics is defined inductively as follows (e.g. [19]).

$$\begin{array}{lll} s, h \models b & \Leftrightarrow_{df} & b^s = \mathsf{true} \\ s, h \models \neg p & \Leftrightarrow_{df} & s, h \not\models p \\ s, h \models p \vee q & \Leftrightarrow_{df} & s, h \models p \quad \text{or} \quad s, h \models q \\ s, h \models \forall\, v.\ p & \Leftrightarrow_{df} & \forall\, x \in \mathbb{Z} : (v, x) \,|\, s, h \models p \\ s, h \models \mathsf{emp} & \Leftrightarrow_{df} & h = \emptyset \\ s, h \models e_1 \mapsto e_2 & \Leftrightarrow_{df} & h = \{(\, e_1^s\, ,\, e_2^s\,)\} \\ s, h \models p * q & \Leftrightarrow_{df} & \exists\, h_1, h_2 \in \mathit{Heaps} : \mathsf{dom}(h_1) \cap \mathsf{dom}(h_2) = \emptyset \text{ and} \\ & & \quad h = h_1 \cup h_2 \text{ and } s, h_1 \models p \text{ and } s, h_2 \models q \\ s, h \models p \mathrel{-\!\!*} q & \Leftrightarrow_{df} & \forall\, h' \in \mathit{Heaps} : (\mathsf{dom}(h') \cap \mathsf{dom}(h) = \emptyset \text{ and } s, h' \models p) \\ & & \quad \text{implies } s, h' \cup h \models q\ . \end{array}$$

Here, b is a *bexp*-expression, p, q are assertions and e_1, e_2 are *exp*-expressions. The first four clauses do not make any assumptions about the heap and only carry it along without making any changes to it; they are well known from predicate logic or Hoare logic [9].

The remaining lines describe the new parts in separation logic: For an arbitrary state (s, h), emp ensures that the heap h is empty and contains no addressable cells. An assertion $e_1 \mapsto e_2$ characterises states with the singleton heap that has exactly one cell at the address e_1^s with the value e_2^s. To reason about more complex heaps, the *separating conjunction* $*$ is used. It allows expressing properties of heaps that result from merging smaller disjoint heaps, i.e., heaps with disjoint domains.

The *separating implication* $p \mathrel{-\!\!*} q$ guarantees, that if the current heap h is extended with a heap h' satisfying p, the merged heap $h \cup h'$ satisfies q (cf. Figure 1). If the heaps are not disjoint, the situation is interpreted as an error case and the assertion is not satisfied.

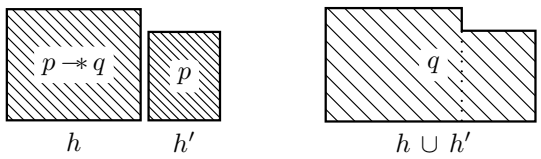

Fig. 1. Separating implication [1]

4 Quantales and Residuals

To present our algebraic semantics of separation logic in the next section we now prepare the algebraic background.

A *quantale* [20] is a structure $(S, \leq, 0, \cdot, 1)$ where $(S, \leq)$ is a complete lattice and $\cdot$ is completely disjunctive, i.e., $\cdot$ distributes over arbitrary suprema. Moreover 0 is the least element and 1 is the identity of the $\cdot$ operation. The infimum and supremum of two elements $a, b \in S$ are denoted by $a \sqcap b$ and $a + b$, resp. The greatest element of S is denoted by $\top$. The definition implies that $\cdot$ is strict, i.e., that $0 \cdot a = 0 = a \cdot 0$ for all $a \in S$. The notion of a quantale is equivalent to that of a *standard Kleene algebra* [3] and a special case of the notion of an idempotent semiring.

A quantale is called *Boolean* if its underlying lattice is distributive and complemented, whence a Boolean algebra. Equivalently, a quantale S is Boolean if it satisfies the Huntington axiom $a = \overline{\overline{a} + \overline{b}} + \overline{\overline{a} + b}$ for all $a, b \in S$ [12, 11]. The infimum is then defined by the de Morgan duality $a \sqcap b =_{df} \overline{\overline{a} + \overline{b}}$. An important Boolean quantale is REL, the algebra of binary relations over a set under set inclusion, relation composition and set complement.

A quantale is called *commutative* if $\cdot$ commutes, i.e., $a \cdot b = b \cdot a$ for all a, b.

[1] The right picture might suggest that the heaps are adjacent after the join. But the intention is only to bring out abstractly that the united heap satisfies q.

In any quantale, the *right residual* $a\backslash b$ [1] exists and is characterised by the Galois connection

$$x \leq a\backslash b \;\Leftrightarrow_{df}\; a \cdot x \leq b \;.$$

Symmetrically, the *left residual* b/a can be defined. However, if the underlying quantale is commutative then both residuals coincide, i.e., $a\backslash b = b/a$. In REL, one has $R\backslash S = \overline{R^\smile}\,;\overline{S}$ and $R/S = \overline{\overline{R}\,;S^\smile}$, where $^\smile$ denotes relational converse and ; is relational composition.

In a Boolean quantale, the *right detachment* $a\lfloor b$ can be defined based on the left residual as

$$a\lfloor b \;=_{df}\; \overline{\overline{a}/b} \;.$$

In REL, $R\lfloor S = R\,;S^\smile$. By de Morgan's laws, the Galois connection for / transforms into the exchange law

$$a\lfloor b \leq x \;\Leftrightarrow\; \overline{x} \cdot b \leq \overline{a} \tag{exc}$$

for $\lfloor$ that generalises the Schröder rule of relational calculus. An important consequence is the Dedekind rule [13]

$$a \sqcap (b \cdot c) \leq (a\lfloor c \sqcap b) \cdot c \;. \tag{Ded}$$

5 An Algebraic Model of Assertions

We now give an algebraic interpretation for the semantics of separation logic. The main idea is to switch from the satisfaction-based semantics for single states to an equivalent set-based one where every assertion is associated with the set of all states satisfying it. This simplifies proofs considerably.

For an arbitrary assertion p we therefore define its set-based semantics as

$$[\![\, p \,]\!] =_{df} \{(s,h) : s,h \models p\} \;.$$

The sets $[\![\, p \,]\!]$ of states will be the elements of our algebra. By this we then have immediately the connection $s,h \models p \;\Leftrightarrow\; (s,h) \in [\![\, p \,]\!]$. This validity assertion can be lifted to set of states by setting, for $A \subseteq \mathit{States}$, $A \models p \;\Leftrightarrow\; A \subseteq [\![p]\!]$. The embedding of the standard Boolean connectives is given by

$$\begin{aligned}
[\![\, \neg p \,]\!] &= \{(s,h) : s,h \not\models p\} = \overline{[\![\, p \,]\!]} \;, \\
[\![\, p \vee q \,]\!] &= [\![\, p \,]\!] \cup [\![\, q \,]\!] \;, \\
[\![\, \forall v.\, p \,]\!] &= \{(s,h) : \forall x \in \mathbb{Z}\,.\, (v,x) \,|\, s,h \models p\} \;.
\end{aligned}$$

Using these definitions, it is straightforward to show that

$$\begin{aligned}
[\![\, p \wedge q \,]\!] = [\![p]\!] \cap [\![q]\!] \;, \qquad [\![\, p \to q \,]\!] = \overline{[\![p]\!]} \cup [\![q]\!] \;, \quad \text{and} \\
[\![\, \exists v.\, p \,]\!] = \overline{[\![\, \forall v.\, \neg p \,]\!]} = \{(s,h) : \exists x \in \mathbb{Z}\,.\, (v,x) \,|\, s,h \models p\} \;,
\end{aligned}$$

where $|$ is the update operation defined in (1).

The emptiness assertion emp and the assertion operator $\mapsto$ are given by

$$\begin{aligned} [\![\,\mathsf{emp}\,]\!] &=_{df} \{(s,h) : h = \emptyset\} \\ [\![e_1 \mapsto e_2]\!] &=_{df} \{(s,h) : h = \{(e_1^s, e_2^s)\}\} \ . \end{aligned}$$

Next, we model the separating conjunction $*$ algebraically by

$$\begin{aligned} [\![\,p * q\,]\!] &=_{df} [\![\,p\,]\!] \uplus [\![\,q\,]\!], \text{where} \\ P \uplus Q &=_{df} \{(s, h \cup h') : (s,h) \in P \wedge (s,h') \in Q \wedge \mathsf{dom}(h) \cap \mathsf{dom}(h') = \emptyset\} \ . \end{aligned}$$

In this way inconsistent states as well as "erroneous" merges of non-disjoint heaps are excluded.

These definitions yield an algebraic embedding of separation logic.

Theorem 5.1 *The structure* $\mathsf{AS} =_{df} (\mathcal{P}(States), \subseteq, \emptyset, \cup, \uplus, [\![\,\mathsf{emp}\,]\!])$ *is a commutative and Boolean quantale with* $P + Q = P \cup Q$.

The proof is by straightforward calculations; it can be found in [4]. It is easy to show that $[\![\mathsf{true}]\!]$ is the greatest element in the above quantale, i.e., $[\![\mathsf{true}]\!] = \top$, since every state satisfies the assertion true. This implies immediately that $[\![\mathsf{true}]\!]$ is the neutral element for $\sqcap$. However, in contrast to addition $\cup$, multiplication $\uplus$ is in general not idempotent.

Example 5.2 In AS,

$$[\![\,(x \mapsto 1) * (x \mapsto 1)\,]\!] = [\![\,x \mapsto 1\,]\!] \uplus [\![\,x \mapsto 1\,]\!] = \emptyset \ .$$

This can be shown by straightforward calculations using the above definitions.

$$\begin{aligned} & [\![\,(x \mapsto 1) * (x \mapsto 1)\,]\!] \\ = \ & [\![\,(x \mapsto 1)\,]\!] \uplus [\![\,(x \mapsto 1)\,]\!] \\ = \ & \{(s, h \cup h') : (s,h), (s,h') \in [\![\,x \mapsto 1\,]\!] \wedge \mathsf{dom}(h) \cap \mathsf{dom}(h') = \emptyset\} \\ = \ & \emptyset \ . \end{aligned}$$

$[\![\,x \mapsto 1\,]\!]$ is the set of all states that have the single-cell heap $\{(s(x), 1)\}$. The states (s,h) and (s,h') have to share this particular heap. Hence the domains of the merged heaps would not be disjoint. Therefore the last step yields the empty result. □

As a check of the adequacy of our definitions we list a couple of properties.

Lemma 5.3 *In separation logic, for assertions* p, q, r, *we have*

$$\frac{}{(p \wedge q) * r \ \Rightarrow \ (p * r) \wedge (q * r)} \qquad \textit{and} \qquad \frac{p \ \Rightarrow \ r \quad q \ \Rightarrow \ s}{p * q \ \Rightarrow \ r * s} \ .$$

The second property denotes isotony of separating conjunction. Both properties together are, by standard quantale theory, equivalent to isotony of separating conjunction.

More laws and examples can be found in [4].

For the separating implication the set-based semantics extracted from the definition in Section 3 is

$$[\![p \mathrel{-\!\!*} q]\!] =_{df} \{(s,h) : \forall\, h' \in \mathit{Heaps} : (\, \mathsf{dom}(h) \cap \mathsf{dom}(h') = \emptyset \wedge (s,h') \in [\![p]\!]) \Rightarrow (s, h \cup h') \in [\![q]\!]\} \; .$$

This implies that separating implication corresponds to a residual.

Lemma 5.4 *In* AS, $[\![p \mathrel{-\!\!*} q]\!] = [\![p]\!] \backslash [\![q]\!] = [\![q]\!] / [\![p]\!]$.

Proof. We first show the claim for a single state. By definition above, set theory and definition of $\cup$, we have

$$\begin{array}{ll} & (s,h) \in [\![p \mathrel{-\!\!*} q]\!] \\ \Leftrightarrow & \forall\, h' : ((s,h') \in [\![p]\!] \wedge \mathsf{dom}(h) \cap \mathsf{dom}(h') = \emptyset \Rightarrow (s, h \cup h') \in [\![q]\!]) \\ \Leftrightarrow & \{(s, h \cup h') : (s,h') \in [\![p]\!] \wedge \mathsf{dom}(h) \cap \mathsf{dom}(h') = \emptyset\} \subseteq [\![q]\!] \\ \Leftrightarrow & \{(s,h)\} \cup [\![p]\!] \subseteq [\![q]\!] \; . \end{array}$$

and therefore, for arbitrary set R of states,

$$\begin{array}{ll} & R \subseteq [\![p \mathrel{-\!\!*} q]\!] \\ \Leftrightarrow & \forall\, (s,h) \in R : (s,h) \in [\![p \mathrel{-\!\!*} q]\!] \\ \Leftrightarrow & \forall\, (s,h) \in R : \{(s,h)\} \cup [\![p]\!] \subseteq [\![q]\!] \\ \Leftrightarrow & R \cup [\![p]\!] \subseteq [\![q]\!] \; . \end{array}$$

Hence, by definition of the residual, $[\![p \mathrel{-\!\!*} q]\!] = [\![p]\!] \backslash [\![q]\!]$. The second equation follows immediately since multiplication $\cup$ in AS commutes (cf. Section 2). □

Now all laws of [19] about $\mathrel{-\!\!*}$ follow from the standard theory of residuals (e.g. [2]). Many of these laws are proved algebraically in [4]. For example, the two main properties of separating implication, namely the currying and decurrying rules, are nothing but the transcriptions of the defining Galois connection for right residuals.

Corollary 5.5 *In separation logic the following inference rules hold:*

$$\frac{p * q \;\Rightarrow\; r}{p \;\Rightarrow\; (q \mathrel{-\!\!*} r)} \; , \qquad \text{(currying)} \qquad\qquad \frac{p \;\Rightarrow\; (q \mathrel{-\!\!*} r)}{p * q \;\Rightarrow\; r} \; . \qquad \text{(decurrying)}$$

This means that $q \mathrel{-\!\!*} r$ is the weakest assertion guaranteeing that a state in $[\![q \mathrel{-\!\!*} r]\!]$ merged with a state in $[\![q]\!]$ yields a state in $[\![r]\!]$.

As far as we know, in his works Reynolds only states that these laws follow directly from the definition. We are not aware of any proof of the equalities given in Lemma 5.4, although many authors state this claim and refer to Reynolds.

As a further example we prove the algebraic counterpart of the inference rule

$$\overline{q \;*\; (q \mathrel{-\!\!*} p) \;\Rightarrow\; p} \; .$$

Lemma 5.6 *Let S be a quantale. For $a, b \in S$ the inequality $q \cdot (q \setminus p) \leq p$ holds.*

Proof. By definition of residuals we immediately get

$$q \cdot (q \setminus p) \leq p \;\Leftrightarrow\; q \setminus p \leq q \setminus p \;\Leftrightarrow\; \mathsf{true} \; .$$

□

6 Special Classes of Assertions

Reynolds distinguishes different classes of assertions [19]. We will give algebraic characterisations for three main classes, namely *pure*, *intuitionistic* and *precise assertions*. Pure assertions are independent of the heap of a state and therefore only express conditions on store variables. Intuitionistic assertions do not describe the domain of a heap exactly. Hence, when using these assertions one does not know whether the heap contains additional anonymous cells. In contrast, precise assertions point out a unique subheap which is relevant to its predicate.

6.1 Pure Assertions

An assertion p is called *pure* iff it is independent of the heaps of the states involved, i.e.,

$$p \text{ is pure } \Leftrightarrow_{df} (\forall\, h, h' \in Heaps : s, h \models p \Leftrightarrow s, h' \models p) \ .$$

Theorem 6.1 *In* AS *an element* $[\![p]\!]$ *is pure iff it satisfies, for all* $[\![q]\!]$ *and* $[\![r]\!]$,

$$[\![p]\!] \cup [\![\mathsf{true}]\!] \subseteq [\![p]\!] \quad \textit{and} \quad [\![p]\!] \cap ([\![q]\!] \cup [\![r]\!]) \subseteq ([\![p]\!] \cap [\![q]\!]) \cup ([\![p]\!] \cap [\![r]\!]) \ .$$

Before we give the proof, we derive a number of auxiliary laws. The above theorem motivates the following definition.

Definition 6.2 In an arbitrary Boolean quantale S an element p is called *pure* iff it satisfies, for all $a, b \in S$,

$$p \cdot \top \leq p \ , \tag{2}$$

$$p \sqcap (a \cdot b) \leq (p \sqcap a) \cdot (p \sqcap b) \ . \tag{3}$$

The first equation models upwards closure of pure elements. It can be strengthened to an equation since its converse holds for arbitrary Boolean quantales. The second equation enables pure elements to distribute over meet and is equivalent to downward closure.

Lemma 6.3 *Property* (3) *is equivalent to* $p\lfloor\top \leq p$, *where* $p\lfloor\top$ *forms the downward closure of* p.

Proof. ($\Leftarrow$): Using Equation (Ded), isotony and the assumption, we get

$$p \sqcap a \cdot b \leq (p\lfloor b \sqcap a) \cdot b \leq (p\lfloor\top \sqcap a) \cdot b \leq (p \sqcap a) \cdot b$$

and the symmetric formula $p \sqcap a \cdot b \leq a \cdot (p \sqcap b)$. From this the claim follows by

$$p \sqcap (a \cdot b) \ = \ p \sqcap p \sqcap (a \cdot b) \leq p \sqcap ((p \sqcap a) \cdot b) \leq (p \sqcap a) \cdot (p \sqcap b) \ .$$

($\Rightarrow$): By Axiom (3) we obtain $p \sqcap (\overline{p} \cdot \top) \leq (p \sqcap \overline{p}) \cdot (p \sqcap \top) = 0$ and hence, by shunting and the exchange law (exc), $p\lfloor\top \leq p$. □

Corollary 6.4 *p is pure iff $p \cdot \top \leq p$ and $\overline{p} \cdot \top \leq \overline{p}$.*

Corollary 6.5 *Pure elements form a Boolean lattice, i.e., they are closed under $+$, $\sqcap$ and $\overline{\ }$.*

Moreover we get a fixed point characterisation if the underlying quantale commutes.

Lemma 6.6 *In a Boolean quantale, an element p is pure iff $p = (p \sqcap 1) \cdot \top$ holds.*

Proof. We first show that $p = (p \sqcap 1) \cdot \top$ follows from Inequations (2) and (3). By neutrality of $\top$ for $\sqcap$, neutrality of 1 for $\cdot$, meet-distributivity (3) and isotony, we get

$$p \;=\; p \sqcap \top \;=\; p \sqcap (1 \cdot \top) \leq (p \sqcap 1) \cdot (p \sqcap \top) \leq (p \sqcap 1) \cdot \top \;.$$

The converse inequation follows by isotony and Inequation (2):

$$(p \sqcap 1) \cdot \top \leq p \cdot \top \leq p \;.$$

Next we show that $p = (p \sqcap 1) \cdot \top$ implies the two inequations $p \cdot \top \leq p$ and $\overline{p} \cdot \top \leq \overline{p}$ which, by Corollary 6.4, implies the claim. The first inequation is shown by the assumption, the general law $\top \cdot \top = \top$ and the assumption again:

$$p \cdot \top \;=\; (p \sqcap 1) \cdot \top \cdot \top \;=\; (p \sqcap 1) \cdot \top \;=\; p \;.$$

For the second inequation, we note that in a Boolean quantale the law $\overline{s \cdot \top} = (\overline{s} \sqcap 1) \cdot \top$ holds for all subidentities s ($s \leq 1$) (e.g. [6]). From this we get

$$\overline{p} \cdot \top \;=\; \overline{(p \sqcap 1) \cdot \top} \cdot \top \;=\; (\overline{p} \sqcap 1) \cdot \top \cdot \top \;=\; (\overline{p} \sqcap 1) \cdot \top \;=\; \overline{(p \sqcap 1) \cdot \top} \;=\; \overline{p} \;. \qquad \square$$

Corollary 6.7 *The set of pure elements forms a complete lattice.*

Proof. Lemma 6.6 characterises the pure elements as the fixed points of the isotone function $f(x) = (x \sqcap 1) \cdot \top$ on the quantale. By Tarski's fixed point theorem these form a complete lattice. $\square$

Proof of Theorem 6.1. By Lemma 6.6 and definition of the elements of AS it is sufficient to show that the following formulas are equivalent in separation logic

$$\forall\, s \in \mathit{Stores}, \forall\, h, h' \in \mathit{Heaps} : (s, h \models p \Leftrightarrow s, h' \models p) \;, \tag{4}$$

$$\forall\, s \in \mathit{Stores}, \forall\, h \in \mathit{Heaps} : (s, h \models p \Leftrightarrow s, h \models (p \wedge \mathsf{emp}) * \mathsf{true}) \;. \tag{5}$$

Since both assertions are universally quantified over states we omit that quantification in the remainder and only keep the quantifiers on heaps. Before proving this equivalence we simplify $s, h \models (p \wedge \mathsf{emp}) * \mathsf{true}$. Using the definitions of Section 3, we get for all $h \in \mathit{Heaps}$

$$
\begin{aligned}
& s,h \models (p \wedge \mathsf{emp}) * \mathsf{true} \\
\Leftrightarrow\ & \exists\, h_1, h_2 \in \mathit{Heaps} : \mathsf{dom}(h_1) \cap \mathsf{dom}(h_2) = \emptyset \text{ and } h = h_1 \cup h_2 \\
& \text{ and } s,h_1 \models p \text{ and } s,h_1 \models \mathsf{emp} \text{ and } s,h_2 \models \mathsf{true} \\
\Leftrightarrow\ & \exists\, h_1, h_2 \in \mathit{Heaps} : \mathsf{dom}(h_1) \cap \mathsf{dom}(h_2) = \emptyset \text{ and } h = h_1 \cup h_2 \\
& \text{ and } s,h_1 \models p \text{ and } h_1 = \emptyset \\
\Leftrightarrow\ & \exists\, h_2 \in \mathit{Heaps} : h = h_2 \text{ and } s,\emptyset \models p \\
\Leftrightarrow\ & s,\emptyset \models p\ .
\end{aligned}
$$

The last line shows that a pure assertion is independent of the heap and hence, in particular, has to be satisfied for the empty heap. Next we show the implication (4) $\Rightarrow$ (5). Instantiating Equation (4) and using the above result immediately imply the claim:

$$
\begin{aligned}
& \forall\, h,h' \in \mathit{Heaps} : (s,h \models p \Leftrightarrow s,h' \models p) \\
\Rightarrow\ & \forall\, h \in \mathit{Heaps} : (s,h \models p \Leftrightarrow s,\emptyset \models p) \\
\Leftrightarrow\ & \forall\, h \in \mathit{Heaps} : (s,h \models p \Leftrightarrow s,h \models (p \wedge \mathsf{emp}) * \mathsf{true})\ .
\end{aligned}
$$

For the converse direction, we take two instances of (5). Then, using again the above result, we get

$$
\begin{aligned}
& \forall\, h \in \mathit{Heaps} : (s,h \models p \Leftrightarrow s,h \models (p \wedge \mathsf{emp}) * \mathsf{true}) \\
& \text{ and } \forall\, h' \in \mathit{Heaps} : (s,h' \models p \Leftrightarrow s,h' \models (p \wedge \mathsf{emp}) * \mathsf{true}) \\
\Rightarrow\ & \forall\, h,h' \in \mathit{Heaps} : (s,h \models p \Leftrightarrow s,h \models (p \wedge \mathsf{emp}) * \mathsf{true} \\
& \text{ and } s,h' \models p \Leftrightarrow s,h' \models (p \wedge \mathsf{emp}) * \mathsf{true}) \\
\Leftrightarrow\ & \forall\, h,h' \in \mathit{Heaps} : (s,h \models p \Leftrightarrow s,\emptyset \models p \text{ and } s,h' \models p \Leftrightarrow s,\emptyset \models p) \\
\Rightarrow\ & \forall\, h,h' \in \mathit{Heaps} : (s,h \models p \Leftrightarrow s,h' \models p)\ .
\end{aligned}
$$

□

The complexity of this proof in predicate-logic illustrates the advantage that is gained by passing to an algebraic treatment. Logic-based formulas (in particular in separation logic) can become long and complicated. Calculating at the abstract level of quantales often shorten the proofs. Moreover the abstraction paves the way to using first-order off-the-shelf theorem provers for verifying properties; whereas a first-order theorem prover for separation logic has yet to be developed and implemented (cf. Section 7).

To conclude the paragraph concerning pure elements we list a couple of properties which can be proved very easily by our algebraic approach.

Lemma 6.8 *Consider a Boolean quantale S, pure elements $p,q \in S$ and arbitrary elements $a,b \in S$ Then*

(a) $p \cdot a = p \sqcap a \cdot \top$;
(b) $(p \sqcap a) \cdot b = p \sqcap a \cdot b$;
(c) $p \cdot q = p \sqcap q$; *in particular* $p \cdot p = p$ *and* $p \cdot \overline{p} = 0$.

Their corresponding counterparts in separation logic and the proofs can again be found in [4].

The following lemma shows a.o. that in the complete lattice of pure elements meet and join coincide with composition and sum, respectively.

As far as we know these closure properties are new and were not shown in separation logic so far.

6.2 Intuitionistic Assertions

Let us now turn to intuitionistic assertions. Following [19], an assertion p is *intuitionistic* iff

$$\forall\, s \in \mathit{Stores}, \forall\, h, h' \in \mathit{Heaps} : (h \subseteq h' \text{ and } s, h \models p) \text{ implies } s, h' \models p\ . \quad (6)$$

This means for a heap that satisfies an intuitionistic assertion p that it can be extended by arbitrary cells and still satisfies p.

Similar calculations as in the proof of Theorem 6.1 yield the equivalence of Equation (6) and

$$\forall\, s \in \mathit{Stores}, \forall\, h \in \mathit{Heaps} : (s, h \models p * \mathsf{true} \Rightarrow s, h \models p)\ . \quad (7)$$

Lifting this to an abstract level motivates the following definition.

Definition 6.9 In an arbitrary Boolean quantale S an element i is called *intuitionistic* iff it satisfies

$$i \cdot \top \leq i\ . \quad (8)$$

Elements of the form $i \cdot \top$ are also called vectors or ideals.

Corollary 6.10 *Every pure element of a Boolean quantale is intuitionistic.*

As before we just give a couple of properties. The proofs are again straightforward at the algebraic level.

Lemma 6.11 *Consider a Boolean quantale S, intuitionistic elements $i, j \in S$ and arbitrary elements $a, b \in S$ Then*

(a) $(i \sqcap 1) \cdot \top \leq i$;
(b) $i \cdot a \leq i \sqcap (a \cdot \top)$;
(c) $(i \sqcap a) \cdot b \leq i \sqcap (a \cdot b)$;
(d) $i \cdot j \leq i \sqcap j$.

Using the quantale AS, it is easy to see that none of these inequations can be strengthened to an equation. In particular, unlike as for pure assertions, multiplication and meet need not coincide.

Example 6.12 Consider $i =_{df} j =_{df} [\![x \mapsto 1 * \mathsf{true}]\!] = [\![x \mapsto 1]\!] \cup [\![\mathsf{true}]\!]$. By this definition it is obvious that i and j are intuitionistic. The definitions of Section 3 then immediately imply

$$i \cap j = [\![x \mapsto 1]\!] \cup [\![\mathsf{true}]\!]$$
$$i \cup j = [\![x \mapsto 1]\!] \cup [\![\mathsf{true}]\!] \cup [\![x \mapsto 1]\!] \cup [\![\mathsf{true}]\!] = \emptyset\ .$$

The last step follows from Example 5.2. □

Other classes of assertions for separation logic are given in [19] and most of their algebraic counterparts in [4].

6.3 Precise Assertions

An assertion p is called *precise* if and only if for all states (s, h), there is at most one subheap h' of h for which $(s, h') \models p$. According to [18], this definition is equivalent to distributivity of $*$ over $\wedge$. Hence, using isotony of $*$ we can algebraically characterise precise assertions as follows.

Definition 6.13 In an arbitrary Boolean quantale S an element r is called *precise* iff for all p, q

$$(r * p) \sqcap (r * q) \leq r * (p \sqcap q) \ . \tag{9}$$

Next we give some closure properties for this assertion class.

Lemma 6.14 *If p and q are precise then also $p * q$ is precise.*

Proof. For arbitrary r_1 and r_2 we calculate

$$p * q * r_1 \sqcap p * q * r_2 \leq p * (q * r_1 \sqcap q * r_2) \leq (p * q) * (r_1 \sqcap r_2)$$

assuming p and q are precise. □

Lemma 6.15 *If p is precise and $q \leq p$ then q is precise, i.e., precise assertions are downward closed.*

A proof can be found in [6].

Corollary 6.16 *For an arbitrary assertion q and precise p, also $p \sqcap q$ is precise.*

7 Conclusion and Outlook

We have presented a treatment towards an algebra of separation logic. For assertions we have introduced a model based on sets of states. By this, separating implication coincides with a residual and most of the inference rules of [19] are simple consequences of standard residual laws. For pure, intuitionistic and precise assertions we have given algebraic axiomatisations.

The next step will be to embed the command language of separation logic into a relational algebraic structure. A first attempt is given in [5] where we have defined a relational semantics for the heap-dependent commands and lifted the set-based semantics of assertions to relations. There, we are able to characterise the frame rule

$$\frac{\{p\}\, c\, \{q\}}{\{p * r\}\, c\, \{q * r\}} \ ,$$

where p, q and r are arbitrary assertions and c is a command. The rule assumes that no free variable of r is modified by c. However, a complete algebraic proof of the frame rule is still missing, since we do not yet know how to characterise the conditions controlling the modification of free variables.

To underpin our approach we have algebraically verified one of the standard examples — an in-place list reversal algorithm. The details can be found in [4]. The term *in-place* means that there is no copying of whole structures, i.e., the reversal is done by simple pointer modifications.

So far we have not analysed situations where data structures share parts of their cells (cf. Figure 2). First steps towards an algebraic handling of such

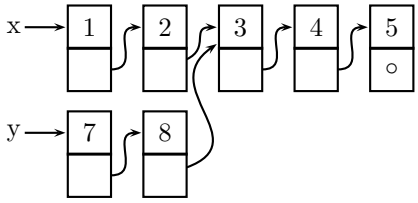

Fig. 2. Two lists with shared cells.

situations are given in [15, 8]. In future work, we will adapt these approaches for our algebra of separation logic.

Our algebraic approach to separation logic also paves the way to verifying-properties with off-the-shelf theorem provers. Boolean semirings and quantales have proved to be reasonably well suitable for automated theorem provers [10]. Hence one of the next plans for future work is to analyse the power of such systems for reasoning with separation logic. A long-term perspective is to incorporate reasoning about concurrent programs with shared linked data structures along the lines of [16].

Acknowledgements: We are most grateful to the anonymous referees for their many valuable remarks.

References

1. G. Birkhoff. *Lattice Theory*, volume XXV of *Colloquium Publications*. American Mathematical Society, 3rd edition, 1967.
2. T. Blyth and M. Janowitz. *Residuation Theory*. Pergamon Press, 1972.
3. J. H. Conway. *Regular Algebra and Finite Machines*. Chapman & Hall, 1971.
4. H.-H. Dang. Algebraic aspects of separation logic. Technical Report 2009-01, Institut für Informatik,, 2009.
5. H.-H. Dang, P. Höfner, and B. Möller. Towards algebraic separation logic. Technical Report 2009-12, Institut für Informatik, Universität Augsburg, 2009.
6. J. Desharnais and B. Möller. Characterizing Determinacy in Kleene Algebras. *Information Sciences*, 139:253–273, 2001.
7. E. Dijkstra. *A discipline of programming*. Prentice Hall, 1976.
8. T. Ehm. Pointer Kleene algebra. In R. Berghammer, B. Möller, and G. Struth, editors, *Relational and Kleene-Algebraic Methods in Computer Science*, volume 3051 of *LNCS*, pages 99–111. Springer, 2004.

9. C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.
10. P. Höfner and G. Struth. Automated reasoning in Kleene algebra. In F. Pfennig, editor, *Automated Deduction*, volume 4603 of *LNAI*, pages 279–294. Springer, 2007.
11. E. V. Huntington. Boolean algebra. A correction. *Transaction of AMS*, 35:557–558, 1933.
12. E. V. Huntington. New sets of independent postulates for the algebra of logic. *Transaction of AMS*, 35:274–304, 1933.
13. B. Jónsson and A. Tarski. Boolean algebras with operators, Part I. *American Journal of Mathematics*, 73, 1951.
14. B. Möller. Towards pointer algebra. *Science of Computer Prog.*, 21(1):57–90, 1993.
15. B. Möller. Calculating with acyclic and cyclic lists. *Information Sciences*, 119(3-4):135–154, 1999.
16. P. O'Hearn. Resources, concurrency, and local reasoning. *Theoretical Computer Science*, 375:271–307, 2007.
17. P. W. O'Hearn, J. C. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In L. Fribourg, editor, *CSL '01: 15th International Workshop on Computer Science Logic*, volume 2142 of *LNCS*, pages 1–19. Springer, 2001.
18. P. W. O'Hearn, J. C. Reynolds, and H. Yang. Separation and information hiding. *ACM Trans. Program. Lang. Syst.*, 31(3):1–50, 2009.
19. J. C. Reynolds. An introduction to separation logic. Proceedings Marktoberdorf Summer School 2008 (forthcoming).
20. K. Rosenthal. Quantales and their applications. *Pitman Research Notes in Mathematics Series*, 234, 1990.