Algebraic separation logic

Han Hing Dang, Peter Höfner, Bernhard Möller

Algebraic Separation Logic

H.-H Dang[a], P. Höfner[a,b], B. Möller[a]

[a] *Institut für Informatik, Universität Augsburg, D-86159 Augsburg, Germany*
[b] *National ICT Australia Ltd. (NICTA), Sydney, Australia*

Abstract

We present an algebraic approach to separation logic. In particular, we give an algebraic characterisation for assertions of separation logic, discuss different classes of assertions and prove abstract laws fully algebraically. After that, we use our algebraic framework to give a relational semantics of the commands of a simple programming language associated with separation logic. On this basis we prove the frame rule in an abstract and concise way, parametric in the operator of separating conjunction, of which two particular variants are discussed. In this we also show how to algebraically formulate the requirement that a command preserves certain variables. The algebraic view does not only yield new insights on separation logic but also shortens proofs due to a point free representation. It is largely first-order and hence enables the use of off-the-shelf automated theorem provers for verifying properties at an abstract level.

Keywords: formal semantics, separation logic, frame rule, algebra, semirings, quantales

1. Introduction

Two prominent formal methods for reasoning about the correctness of programs are Hoare logic [20] and Dijkstra's wp-calculus [18]. 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. To overcome this deficiency, Reynolds, O'Hearn and others have developed *separation logic* for reasoning about complex and shared data structures [42, 46]. Their approach extends Hoare logic by a "spatial conjunction" and adds assertions to express separation between memory regions. In particular, for arbitrary assertions p and q the conjunction $p * q$ asserts that p and q both hold, but each for a separate part of the storage. This allows expressing aspects of locality, e.g., that mutation of a single cell in the part that satisfies p will not affect the part that satisfies q. Hence, when reasoning about a program, one may concentrate on the memory locations that are actually touched by its execution and then embed them into a larger memory context.

The basic idea of the spatial conjunction is related to early work of Burstall [7] which was then explicitly described in an intuionistic logic by Reynolds and others. This classical version of separation logic was extended by Reynolds with a command language that allows altering separate ranges and includes pointer arithmetic. O'Hearn extended this language to concurrent programs that work on shared mutable data structures [41].

In this paper we present an abstract algebraic approach to separation logic. Our approach is based on quantales [39] These structures, also called standard Kleene algebras [10], are a special case of the fundamental algebraic structure of idempotent semirings, which have been used in various applications ranging from concurrency control [9, 21, 22] to program analysis [30] and semantics [36]. In particular, there are already algebraic characterisations for Hoare logic [28, 37] and the wp-calculus of Dijkstra [38], which serve as the basis of our current approach.

The algebraic approach achieves several goals. First, the view becomes more abstract, which leads to a considerable reduction of detail and hence allows simpler and more concise proofs. On some occasions also additional precision is gained. Second, the algebraic abstraction places the topic into a more general context and therefore allows re-use of a large body of existing theory. Last but not least, since it is largely formulated in pure first-order logic, the

Email addresses: `h.dang@informatik.uni-augsburg.de` (H.-H Dang), `peter.hoefner@nicta.com.au` (P. Höfner), `moeller@informatik.uni-augsburg.de` (B. Möller)

algebraic view enables the use of off-the-shelf automated theorem provers for verifying properties at the more abstract level.

The paper is organised as follows. In Section 2 we recapitulate syntax and semantics of the expressions and formulas of separation logic. 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 that satisfy an assertion. Abstracting from that set-theoretic view yields an algebraic interpretation of assertions in the setting of idempotent semirings and quantales. In Section 6 we discuss special classes of assertions: intuitionistic assertions that do not specify the heap exactly, pure assertions which do not depend on the heap at all, precise assertions to characterise an unambiguous heap portion, and supported assertions that guarantee a subheap for all heaps that satisfy the assertion. After that, we extend our algebra to cover the command part of separation logic in Section 7 and finally in Section 8 we give an algebraic proof for the frame rule including a simple treatment of its side conditions. 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:

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

where V is the set of program variables, $\uplus$ denotes the disjoint union on sets and $M \leadsto N$ denotes the set of partial functions between M and N. The constant nil is a value for pointers that denotes an improper reference like null in programming languages like Java or C; 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 (or partial function) R by $dom(R)$:

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

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

As in [33] and for later definitions we also need an *update* operator. It is used to model changes in stores and heaps. Let f_1 and f_2 be partial functions. Then we define

$$f_1 \mid f_2 =_{df} f_1 \cup \{(x,y) : (x,y) \in f_2 \wedge x \notin dom(f_1)\}\,. \qquad (1)$$

The function f_1 updates the function f_2 with all possible pairs of f_1 in such a way that $f_1 \mid f_2$ is again a partial function. The domain of the right argument of $\cup$ is disjoint from that of f_1. In particular, $f_1 \mid f_2$ can be seen as an extension of f_1 to $dom(f_1) \cup dom(f_2)$. In later definitions we abbreviate an update $\{(x,y)\} \mid f$ on a single variable or address by omitting the set-braces and simply writing $(x,y) \mid f$ 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

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

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 $dom(s)$). For example,

$$z^S = z \quad \forall z \in Values = \mathbb{Z}\,, \quad \mathsf{true}^S = \mathsf{true} \quad \text{and} \quad \mathsf{false}^S = \mathsf{false}\,.$$

3. Assertions

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

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

In the remainder we consistently write p, q and r for assertions of separation logic. 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 iff 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* iff there exists a state (s, h) which satisfies p. The semantics is defined inductively as follows (e.g. [46]).

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

Here, b is a *bexp*-expression, p, q are assertions and e_1, e_2 are *exp*-expressions. The first four clauses do not consider the heap; they are well known from predicate logic or Hoare logic [20]. 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. Note, that there is no separating operator for stores. Later, in Section 8, we will introduce an operator $*_s$ for splitting stores and heaps simultaneously.

A state (s, h) satisfies the *separating implication* $p \mathrel{-\!\!*} q$ iff, whenever its heap h is extended with a disjoint heap h' satisfying p, the state with identical store s and the combined heap $h \cup h'$ is guaranteed to satisfy q (cf. Figure 1). The diagram for $h \cup h'$ might suggest that the heaps are adjacent after the join. But the intention is only to illustrate that the united heap satisfies q.

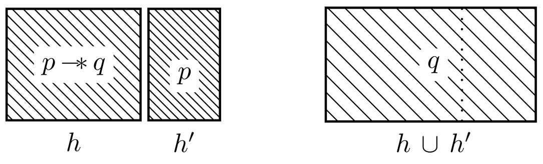

Figure 1: Separating implication

4. Quantales

To present our algebraic semantics of separation logic in the next section, we now prepare the algebraic background by defining the main algebraic structure for assertions.

Definition 4.1.

(a) A *quantale* [39, 47] is a structure $(S, \leq, \cdot, 1)$ where $(S, \leq)$ is a complete lattice, $(S, \cdot, 1)$ is a monoid and multiplication $\cdot$ distributes over arbitrary suprema: for $a \in S$ and $T \subseteq S$,

$$a \cdot (\bigsqcup T) = \bigsqcup\{a \cdot b : b \in T\} \quad \text{and} \quad (\bigsqcup T) \cdot a = \bigsqcup\{b \cdot a : b \in T\}\,, \tag{2}$$

where, for $U \subseteq S$, the element $\bigsqcup U$ is the supremum of U. The least and greatest element of S are denoted by 0 and $\top$, resp. The infimum and supremum of two elements $a, b \in S$ are denoted by $a \sqcap b$ and $a + b$, resp. We assume for the rest of this paper that $\cdot$ binds tighter than $\sqcap$ and $+$. The definition implies that $\cdot$ is strict, i.e., we have $0 \cdot a = 0 = a \cdot 0$ for all $a \in S$. The notion of a quantale is equivalent to that of a *standard Kleene algebra* [10] and a special case of the notion of an idempotent semiring.

(b) A quantale is called *commutative* iff $a \cdot b = b \cdot a$ for all $a, b \in S$.

(c) A quantale is called *Boolean* iff its underlying lattice is distributive and complemented, whence a Boolean algebra. In such a structure, complementation is denoted by $\overline{}$. Moreover, we define the greatest element $\top$ by $\overline{0}$.

An important Boolean quantale is REL, the algebra of binary relations over a set under set inclusion $\subseteq$ and relation composition ; .

A useful property in a Boolean quantale is shunting:

$$a \sqcap b \leq c \ \Leftrightarrow\ b \leq c + \overline{a}\,. \tag{shu}$$

In particular, $a \sqcap b \leq 0 \ \Leftrightarrow\ b \leq \overline{a}$.

Definition 4.2.

(a) In any quantale, the *right residual* $a\backslash b$ [4] exists and is characterised by the Galois connection

$$x \leq a\backslash b \ \Leftrightarrow_{df}\ a \cdot x \leq b\,. \tag{3}$$

This specifies $a\backslash b$ as the greatest solution of the inequation $a \cdot x \leq b$. Therefore, $\backslash$ is a pseudo-inverse to composition, which is useful in many circumstances.

(b) Symmetrically, the *left residual* b/a can be defined. If the underlying quantale is commutative then both residuals coincide, i.e., $a\backslash b = b/a$. In REL, one has $R_1\backslash R_2 = \overline{R_1{}^{\smile} ; \overline{R_2}}$ and $R_1/R_2 = \overline{\overline{R_1} ; R_2{}^{\smile}}$, where $^{\smile}$ denotes relational converse.

(c) 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, $R_1\lfloor R_2 = R_1 ; R_2{}^{\smile}$.

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 [27]

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

The operator $\lfloor$ is isotone in both arguments.

Definition 4.3. In every quantale we define a *test* as an element $t \leq 1$ that has a complement $\neg t$ relative to 1, i.e., $t + \neg t = 1$ and $t \cdot \neg t = 0 = t \cdot \neg t$. The set of all tests of S is denoted by $\mathsf{test}(S)$. It is closed under $+$ and $\cdot$, which coincide with $\sqcup$ and $\sqcap$, resp., and forms a Boolean algebra with 0 and 1 as its least and greatest elements.

In a Boolean quantale, for each a the element $a \sqcap 1$ is a test with complement $\neg(a \sqcap 1) = \overline{a} \sqcap 1$; in particular, every element below 1 is a test. Moreover, according to [35] for a test t and arbitrary elements $a, b \in S$

$$t \cdot (a \sqcap b) = t \cdot a \ \sqcap \ b = t \cdot a \ \sqcap \ t \cdot b. \qquad \text{(testdist)}$$

And as a direct consequence, the equation $t_1 \cdot a \sqcap t_2 \cdot a = t_1 \cdot t_2 \cdot a$ holds for tests t_1, t_2 and an arbitrary element a.

Algebraic structures in general are suitable for automated theorem proving. However, quantales are not easy to encode and perform rather badly [12]. This is mainly due to the two distributivity laws (2). If one only uses the distributivity laws of the form $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(b + c) \cdot a = b \cdot a + c \cdot a$ one can use any first-order automated theorem proving system (ATP system). Examples are Prover9 [32] and Waldmeister [6]. This simpler structure—an idempotent semiring—is particularly suitable for ATP systems [24, 25]. For the purpose of this paper, we have proven most of the theorems using ATP systems. Most of the input files can be found at a web page [23]. However, to demonstrate the simplicity of our algebra, we will also present most of the proofs within the paper.

5. An Algebraic Model of Assertions

In this section we 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 pointfree 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

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

Sets of states will be the elements of our algebra, which later will be abstracted to an arbitrary Boolean quantale. For the standard Boolean connectives we obtain

$$\begin{array}{rcl}
[\![\, \neg p \,]\!] & = & \{(s,h) : s,h \not\models p\} = \overline{[\![\, p \,]\!]} \,, \\
[\![\, p \vee q \,]\!] & = & [\![\, p \,]\!] \cup [\![\, q \,]\!] \,, \\
[\![\, p \wedge q \,]\!] & = & [\![p]\!] \cap [\![q]\!] \,, \qquad [\![\, p \to q \,]\!] = \overline{[\![p]\!]} \cup [\![q]\!] \,, \\
[\![\, \forall v : p \,]\!] & = & \{(s,h) : \forall x \in \mathbb{Z} : (v,x) \,|\, s, h \models p\} \,, \\
 & = & \bigsqcap_{x \in \mathbb{Z}} \{(s,h) : ((v,x) \,|\, s, h) \in [\![\, p \,]\!]\} \,, \\
[\![\, \exists v : p \,]\!] & = & \overline{[\![\, \forall v : \neg p \,]\!]} = \{(s,h) : \exists x \in \mathbb{Z} : (v,x) \,|\, s,h \models p\} \\
 & = & \bigsqcup_{x \in \mathbb{Z}} \{(s,h) : ((v,x) \,|\, s, h) \in [\![\, p \,]\!]\} \,,
\end{array}$$

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

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

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

Next, we reformulate the separating conjunction $*$ algebraically as

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

This yields an algebraic embedding of separation logic assertions.

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

The proof is by straightforward calculations; it can be found in [11]. 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, the set $[\![\, x \mapsto 1 \,]\!]$ consists of all states (s,h) that have the single-cell heap $\{(s(x),1)\}$. We calculate

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

In the last step, the states (s,h) and (s,h') would have to share that particular heap. Hence the domains of the 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 AS, for assertions p,q,r, we have the inference rules*

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

where an inference rule reads as implication between the premises and the conclusion and $p \Rightarrow q$ stands for $[\![p]\!] \subseteq [\![q]\!]$.

The second property denotes isotony of separating conjunction. More laws and examples can be found in [11].

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

$$\begin{aligned}
[\![\, p \mathrel{-\!\!*} q \,]\!] \quad = \quad &\{(s,h) : \forall\, h' \in Heaps : \ (dom(h) \cap dom(h') = \emptyset \ \land\ (s,h') \in [\![\, p \,]\!]) \\
&\Rightarrow\ (s, h \cup h') \in [\![\, q \,]\!]\}\,.
\end{aligned}$$

This implies that separating implication corresponds to a residual.

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

Proof. By set theory and definition of $\uplus$, we have

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

and therefore, for arbitrary set R of states,

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

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

Now all all laws about $\mathrel{-\!\!*}$ given by Reynolds in [46] follow from the standard theory of residuals (e.g. [5]). Many of these laws are proved algebraically in [11]. For example, the defining Galois connection in Equation (3) for the residual specialises to

$$p * q \le r \ \Leftrightarrow\ p \le (q \mathrel{-\!\!*} r)\,.$$

Since for our assertions the order $\le$ coincides with implication $\Rightarrow$, splitting this equivalence into two implications yields the currying and decurrying rules.

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

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

In Reynolds works it is often only stated that these laws follow directly from the definition although many authors uses these laws as an equivalent definition and refer to Reynolds. In fact the separating implication was first defined in [26] and used there as a particular instantiation of a Boolean BI algebra [45], i.e., a Boolean algebra with an additional residuated commutative monoid structure. Hence Lemma 5.4 expresses exactly the expected equalities.

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

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

Lemma 5.6. *Let S be a quantale. For $a, b \in S$ the inequality $b \cdot (b \backslash a) \leq a$ holds. And therefore also $(a/b) \cdot b \leq a$.*

Proof. By definition of residuals we immediately get

$$b \cdot (b\backslash a) \leq a \;\Leftrightarrow\; b\backslash a \leq b\backslash a \;\Leftrightarrow\; \mathsf{true}\,.$$

□

Calculating at the abstract level of quantales often shortens the proofs. Moreover the abstraction paves the way to use 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 9).

Using the definitions of Section 4 we now give a concrete definition of the right detachment operator in the logic itself. This operation is called *septraction* in the literature (see [49]).

Definition 5.7. $p \mathrel{-\!\circledast} q \;=_{df}\; \neg(q \mathrel{-\!*} (\neg p))$. Abstractly, $p \mathrel{-\!\circledast} q = p \lfloor q$.

Lemma 5.8. $s, h \models p \mathrel{-\!\circledast} q \;\Leftrightarrow\; \exists \hat{h} : \; h \subseteq \hat{h},\; s, \hat{h} \models p,\; s, \hat{h} - h \models q$.

The proof of Lemma 5.8 as well as other proofs are deferred to Appendix A.

In the appendix of [13], a couple of properties for septraction are listed. They can all easily be verified (for example using off-the-shelf ATP systems).

So far, we have derived an algebraic structure for assertions and have presented the correspondence between the operations of separation logic and the algebra. We sum up this correspondence in Table 1.

Name in SL	Symbol in SL	Name in Quantales	Symbol in AS	Symbol in Quantales
disjunction	$\vee$	addition/join	$\cup$	$+$
conjunction	$\wedge$	meet	$\cap$	$\sqcap$
negation	$\neg$	complement	$\overline{}$	$\overline{}$
implication	$\Rightarrow$	natural order	$\subseteq$	$\leq$
separating conjunction	$*$	multiplication	$\dot\cup$	$\cdot$
separating implication	$-\!*$	residual	$/$	$/$
septraction	$-\!\circledast$	detachment	$\lfloor$	$\lfloor$

Table 1: Correspondence between operators of separation logic and algebra

6. Special Classes of Assertions

In separation logic one distinguishes different classes of assertions [46]. We now give algebraic characterisations for the most important classes of assertions, namely *intuitionistic*, *pure*, *precise* and *supported assertions*. 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. This is often the case when pointer references to some portions of a heap are lost. Pure assertions are independent of the heap and therefore only express conditions on store variables. In contrast to intuitionistic assertions, the precise ones point out a unique subheap which is relevant to its predicate. Finally supported assertions ensure for a given set of heaps that there exists a subheap for which the predicate already holds. This class is e.g. used in [46] when reasoning about directed acyclic graphs.

6.1. Intuitionistic Assertions

The first assertion class for which we present a simple algebraic characterisation is the class of intuitionistic assertions. Following [46], an assertion p is *intuitionistic* iff

$$\forall\, s \in \mathit{Stores},\ \forall\, h, h' \in \mathit{Heaps} : (h \subseteq h' \ \wedge\ s, h \models p) \ \Rightarrow\ s, h' \models p\,. \tag{4}$$

Intuitively, if a heap h that satisfies an intuitionistic assertion p then every larger heap, i.e. h extended by arbitrary cells, still satisfies p.

Theorem 6.1. *In* AS *an element* $[\![\, p \,]\!]$ *is intuitionistic iff it satisfies*

$$[\![\, p \,]\!] \mathbin{\dot\cup} [\![\, \mathsf{true} \,]\!] \ \subseteq\ [\![\, p \,]\!]\,.$$

Proof.

$$\begin{array}{ll}
 & \forall\, s, h, h' :\ (h \subseteq h' \ \wedge\ s, h \models p) \ \Rightarrow\ s, h' \models p \\
\Leftrightarrow & \quad \{\!\![\text{ Definition of } \mathsf{true} \ ]\!\!\} \\
 & \forall\, s, h, h' :\ (h \subseteq h' \ \wedge\ s, h \models p \ \wedge\ s, (h' - h) \models \mathsf{true}) \ \Rightarrow\ s, h' \models p \\
\Leftrightarrow & \quad \{\!\![\text{ set theory }]\!\!\} \\
 & \forall\, s, h, h' :\ (s, h \models p \ \wedge\ s, (h' - h) \models \mathsf{true} \ \wedge\ \mathit{dom}(h) \cap \mathit{dom}(h' - h) = \emptyset \ \wedge\ h' = h \cup (h' - h)) \\
 & \qquad\qquad \Rightarrow\ s, h' \models p \\
\Leftrightarrow & \quad \{\!\![\ \Rightarrow : h'' = h' - h,\ \Leftarrow : \mathit{dom}(h) \cap \mathit{dom}(h'') = \emptyset \ \wedge\ h' = h \cup h'' \ \Rightarrow\ h'' = h' - h \ ]\!\!\} \\
 & \forall\, s, h, h' :\ (\exists\, h'' :\ s, h \models p \wedge s, h'' \models \mathsf{true} \wedge \mathit{dom}(h) \cap \mathit{dom}(h'') = \emptyset \wedge h' = h \cup h'') \ \Rightarrow\ s, h' \models p \\
\Leftrightarrow & \quad \{\!\![\text{ logic }]\!\!\} \\
 & \forall\, s, h' :\ (\exists\, h, h'' :\ s, h \models p \wedge s, h'' \models \mathsf{true} \wedge \mathit{dom}(h) \cap \mathit{dom}(h'') = \emptyset \wedge h \cup h'' = h') \ \Rightarrow\ s, h' \models p \\
\Leftrightarrow & \quad \{\!\![\text{ Definition of } * \ ]\!\!\} \\
 & \forall\, s, h' :\ s, h' \models p * \mathsf{true} \ \Rightarrow\ s, h' \models p
\end{array}$$

□

Lifting this to an abstract level motivates the following definition.

Definition 6.2. In an arbitrary Boolean quantale S an element a is called *intuitionistic* iff it satisfies

$$a \cdot \top \leq a\,. \tag{5}$$

This inequation can be strengthened to an equation since its converse holds for arbitrary Boolean quantales. Elements of the form $a \cdot \top$ are also called vectors or ideals. Those elements are well known, and therefore we obtain many properties for free (e.g. [48, 31]). We only list some of them to show again the advantages of the algebra.

In particular, we focus on laws that describe the interaction of $\cdot$ and $\sqcap$ using intuitionistic assertions.

Lemma 6.3. *Consider a commutative Boolean quantale* S*, intuitionistic elements* $a, a' \in S$ *and arbitrary elements* $b, c \in S$ *Then*

(a) $(a \sqcap b) \cdot \top \leq a$;

(b) $a \cdot b \leq a \sqcap (b \cdot \top)$;

(c) $(a \sqcap b) \cdot c \leq a \sqcap (b \cdot c)$;

(d) $a \cdot a' \leq a \sqcap a'$.

Proof. To show Part (a) we calculate $(a \sqcap b) \cdot \top \leq a \cdot \top \leq a$. For a proof of Part (b) we know $a \cdot b \leq a \cdot \top \leq a$ and $a \cdot b \leq b \cdot \top$ by isotony of $\cdot$ and the assumption. The Laws (c) and (d) can be proved analogously. $\square$

Using the quantale AS, it is easy to see that none of these inequations can be strengthened to an equation. In particular, multiplication (separation conjunction) and meet need not coincide.

Example 6.4. Consider $a =_{df} a' =_{df} [\![x \mapsto 1 * \mathsf{true}]\!] = [\![x \mapsto 1]\!] \mathbin{\cup\!\!\!\!\cdot\,} [\![\mathsf{true}]\!]$. By this definition it is obvious that a and a' are intuitionistic. The definitions of Section 3 then immediately imply

$$\begin{array}{rcl} a \cap a' & = & [\![x \mapsto 1]\!] \mathbin{\dot\cup} [\![\mathsf{true}]\!] \\ a \mathbin{\dot\cup} a' & = & [\![x \mapsto 1]\!] \mathbin{\dot\cup} [\![\mathsf{true}]\!] \mathbin{\dot\cup} [\![x \mapsto 1]\!] \mathbin{\dot\cup} [\![\mathsf{true}]\!] = \emptyset \,. \end{array}$$

The last step follows from Example 5.2. $\square$

The next class will be a proper subset of intuitionistic assertions where these operations indeed coincide.

6.2. Pure Assertions

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

$$p \text{ is pure} \quad \Leftrightarrow_{df} \quad (\forall\, s \in \mathit{Stores} : \forall\, h, h' \in \mathit{Heaps} : s, h \models p \Leftrightarrow s, h' \models p) \,. \tag{6}$$

Examples for such assertions are $x = 2$, $x = x + 2$, false or true .

Theorem 6.5. *In* AS *an element* $[\![p]\!]$ *is pure iff it satisfies, for all* $[\![q]\!]$ *and* $[\![r]\!]$,

$$[\![p]\!] \mathbin{\dot\cup} [\![\mathsf{true}]\!] \subseteq [\![p]\!] \;\; and \;\; [\![p]\!] \cap ([\![q]\!] \mathbin{\dot\cup} [\![r]\!]) \subseteq ([\![p]\!] \cap [\![q]\!]) \mathbin{\dot\cup} ([\![p]\!] \cap [\![r]\!]) \,.$$

For the proof—which is presented later—we need a number of auxiliary laws. The above theorem motivates the following definition.

Definition 6.6. In an arbitrary Boolean quantale S an element a is called *pure* iff it satisfies, for all $b, c \in S$,

$$a \cdot \top \leq a \,, \tag{7}$$

$$a \sqcap (b \cdot c) \leq (a \sqcap b) \cdot (a \sqcap c) \,. \tag{8}$$

Corollary 6.7. $\top$ *is pure.*

The second equation states that pure elements distribute over meet. By this characterisation we can immediately conclude

Corollary 6.8. *Every pure element of a Boolean quantale is also intuitionistic.*

Lemma 6.9. *In a commutative Boolean quantale, Property* (8) *is equivalent to* $a \lfloor \top \leq a$, *where* $a \lfloor \top$ *forms the downward closure of* a.

Lemma 6.10. *In a commutative Boolean quantale, an element a is pure iff one of the following equivalent properties is satisfied.*

(a) $a \cdot \top \leq a$ *and* $\overline{a} \cdot \top \leq \overline{a}$.

(b) $a = (a \sqcap 1) \cdot \top$.

(c) $(a \sqcap b) \cdot c = a \sqcap b \cdot c$.

A proof can be found in Appendix A. Part (a) also holds for non-commutative quantales; Part (b) characterises pure elements as fixed points. Part (c) is according to [3]. Since the underlying quantale is commutative, there is also the dual of Part (c), namely $b \cdot (a \sqcap c) = a \sqcap b \cdot c$.

With these characterisations we can now prove the equivalence between the formulation in separation logic and the algebraic one.

Proof of Theorem 6.5. By Lemma 6.10(b) and definition of the elements of AS it is sufficient to show that the following formulas are equivalent in separation logic

$$\forall\, s \in \mathit{Stores},\ \forall\, h, h' \in \mathit{Heaps} : (s, h \models p \Leftrightarrow s, h' \models p)\,, \tag{9}$$

$$\forall\, s \in \mathit{Stores},\ \forall\, h \in \mathit{Heaps} : (s, h \models p \Leftrightarrow s, h \models (p \wedge \mathsf{emp}) * \mathsf{true})\,. \tag{10}$$

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 $h \in \mathit{Heaps}$

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

Instantiating Equation (9) and using this result we obtain

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

For the converse direction, we have, for arbitrary s, h, h', that $s, h \models p \Leftrightarrow s, \emptyset \models p \Leftrightarrow s, h' \models p$. □

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

Corollary 6.11. *Pure elements form a Boolean lattice, i.e., they are closed under $+$, $\sqcap$ and $\overline{}$. Moreover the lattice is complete.*

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

Lemma 6.12. *Consider a commutative Boolean quantale S, pure elements $a, a' \in S$ and arbitrary elements $b, c \in S$ Then*

(a) $a \cdot b = a \sqcap b \cdot \top$;

(b) $a \cdot a' = a \sqcap a'$; *in particular* $a \cdot a = a$ *and* $a \cdot \overline{a} = 0$.

Proof. For a proof of Part (a) we calculate, using Lemma 6.10(c), $a \sqcap b \cdot \top = a \sqcap \top \cdot b = (a \sqcap \top) \cdot b = a \cdot b$. To show Part (b), we use again Lemma 6.10(c) and neutrality of 1 w.r.t. $\cdot$ to obtain $a \cdot a' = a \sqcap a' \cdot 1 = a \sqcap a'$. □

Many further properties, in particular, for the interaction of pure assertions with residuals and detachments, can be found in in the appendix of [13]. In some cases we have analogous situations as for the $*$ - operator in Lemma 6.10(c) where pure assertions can be pulled out of each argument of residuals and detachments.

6.3. Precise Assertions

Next we focus on the class of *precise* assertions. They play a major rôle in characterising best local-actions in [8]. Intuitively they point out precise portions of heaps that satisfy the predicate. An assertion p is called *precise* iff for all states (s, h), there is at most one subheap h' of h for which $(s, h') \models p$, i.e.,

$$\forall\, s, h, h_1, h_2 : (s, h_1 \models p \,\wedge\, s, h_2 \models p \,\wedge\, h_1 \subseteq h \,\wedge\, h_2 \subseteq h) \Rightarrow h_1 = h_2$$

According to [43], this definition is equivalent to distributivity of $*$ over $\wedge$.

Theorem 6.13. *In* AS *an element* $[\![p]\!]$ *is precise iff it satisfies, for all* $[\![q]\!]$ *and* $[\![r]\!]$,

$$([\![p]\!] \cup\!\!\!\cdot\; [\![q]\!]) \cap ([\![p]\!] \cup\!\!\!\cdot\; [\![r]\!]) \subseteq [\![p]\!] \cup\!\!\!\cdot\; ([\![q]\!] \cap [\![r]\!]) \,.$$

Hence in our setting, we can algebraically characterise precise assertions as follows.

Definition 6.14. In an arbitrary Boolean quantale S an element a is called *precise* iff for all $b, c \in S$

$$(a \cdot b) \sqcap (a \cdot c) \le a \cdot (b \sqcap c) \,. \tag{11}$$

Equation (11) can be strengthened to an equation since $a \cdot (b \sqcap c) \le (a \cdot b) \sqcap (a \cdot c)$ holds by isotony. Next we give some closure properties for this assertion class which again can be proved fully algebraically.

Lemma 6.15. *If a and a' are precise then so is $a \cdot a'$, i.e., precise assertions are closed under multiplication.*

Proof. The proof is by straightforward calculations. For arbitrary elements b, c and precise elements a, a', we have

$$(a \cdot a') \cdot b \sqcap (a \cdot a') \cdot c = a \cdot (a' \cdot b) \sqcap a \cdot (a' \cdot c) \le a \cdot (a' \cdot b \sqcap a' \cdot c) \le a \cdot a' \cdot (b \sqcap c) \,.$$

□

Lemma 6.16. *If a is precise and $a' \le a$ then a' is precise, i.e., precise assertions are downward closed.*

A proof can be found in [16].

Corollary 6.17. *For an arbitrary assertion b and precise a, also $a \sqcap b$ is precise.*

Further useful properties are again listed in the appendix of [13].

6.4. Fully-allocated assertions

After giving algebraic characterisations for precise and intuitionistic elements we turn to the question whether there exists a class of assertions that can fulfil both properties. At first sight trying to find such assertions does not seem to be sensible, since an assertion p that holds for every larger heap cannot unambiguously point out an exact heap portion. This is stated in [46]. But heaps that are completely allocated, fulfil preciseness *and* intuitionisticness. As a consequence this might disable any allocation of further heap cells in an execution of a program. The assertions in this class are called *fully allocated* and characterised by

$$p \text{ is } \textit{fully allocated} \quad \Leftrightarrow_{df} \quad (\forall\, s, h :\; s, h \models p \Rightarrow \mathit{dom}(h) = \mathit{Addresses}) \,. \tag{12}$$

Theorem 6.18. *In* AS *an element* $[\![p]\!]$ *is fully allocated iff it satisfies*

$$[\![p]\!] \cup\!\!\!\cdot\; \overline{[\![\mathsf{emp}]\!]} \subseteq \emptyset$$

Proof.

$$\begin{array}{l}
\ \forall\, s,h :\ s,h \models p \Rightarrow dom(h) = Addresses \\
\Leftrightarrow \forall\, s,h :\ (s,h \models p \Rightarrow (\forall h'.\ h \subseteq h' \Rightarrow h' \subseteq h)) \\
\Leftrightarrow \forall\, s,h,h' :\ (s,h \models p \Rightarrow (h \subseteq h' \Rightarrow h' \subseteq h)) \\
\Leftrightarrow \forall\, s,h,h' :\ (s,h \models p \Rightarrow \neg(h \subseteq h' \wedge h' - h \neq \emptyset)) \\
\Leftrightarrow \forall\, s,h' :\ \neg(\exists\, h :\ s,h \models p \wedge h \subseteq h' \wedge h' - h \neq \emptyset) \\
\Leftrightarrow \forall\, s,h' :\ \neg(\exists\, h,h'' :\ s,h \models p \wedge h'' \neq \emptyset \wedge dom(h) \cap dom(h'') = \emptyset \wedge h' = h \cup h'') \\
\Leftrightarrow \forall\, s,h' :\ s,h' \models p * \neg\mathsf{emp} \Rightarrow \mathsf{false}
\end{array}$$

□

Consequently in our algebraic setting we characterise this class as follows.

Definition 6.19. In an arbitrary Boolean quantale S an element a is called *fully allocated* iff

$$a \cdot \overline{1} \leq 0\,. \tag{13}$$

Lemma 6.20. *Every fully allocated element is also intuitionistic.*

Proof. Let a be fully allocated, then we $a \cdot \top = a \cdot (1 + \overline{1}) = a \cdot 1 + a \cdot \overline{1} \leq a \cdot 1 = a$. These (in)equations hold by Boolean algebra, distributivity, p being fully allocated and neutrality of 1 w.r.t. $\cdot$. □

Lemma 6.21. *If a is fully allocated then a is also precise.*

For the proof we need an auxiliary property.

Theorem 6.22. *If a is fully allocated then* $a \cdot b = a \cdot (b \sqcap 1)$ *holds.*

Proof. We calculate

$$a \cdot b = a \cdot ((b \sqcap 1) + (b \sqcap \overline{1})) = a \cdot (b \sqcap 1) + a \cdot (b \sqcap \overline{1}) = a \cdot (b \sqcap 1)\,,$$

since $a \cdot (b \sqcap \overline{1}) \leq a \cdot \overline{1} \leq 0$ by isotony of $\cdot$ and the assumption. □

Intuitively, this theorem says that adding storage is only possible if the extra portion is empty. In particular, only the store component can then be changed.

Now we are able to show Lemma 6.21.

Proof of 6.21. For a fully allocated element a and arbitrary b and c, we then get

$$(a \cdot b) \sqcap (a \cdot c) \;=\; (a \cdot (b \sqcap 1)) \sqcap (a \cdot (c \sqcap 1)) \;=\; a \cdot (b \sqcap 1) \cdot (c \sqcap 1) \;=\; a \cdot ((b \sqcap 1) \cdot (c \sqcap 1)) \leq a \cdot (b \sqcap c)\,.$$

Again this holds by Theorem 6.22, (testdist), isotony of $\cdot$, $\sqcap$ and the fact that $\cdot$ coincides with $\sqcap$ on test elements. □

6.5. Supported Assertions

The last assertion class we are considering in this section are *supported* assertions. These assertions characterise pairs of heaps for which joint satisfaction of the assertion can be traced down to a common subheap. These assertions are for example used in [46] for reasoning about directed acyclic graphs.

An assertion p is called *supported* iff

$$\begin{array}{rl}
\forall\, s,h_1,h_2 :\ h_1,h_2 \text{ are compatible} \wedge s,h_1 \models p \wedge s,h_2 \models p & \\
\Rightarrow & \exists\, h' :\ h' \subseteq h_1 \wedge h' \subseteq h_2 \wedge s,h' \models p
\end{array}$$

By the assumption h_1 and h_2 are compatible, it is meant that they agree on their intersection, i.e., $h_1 \cup h_2$ is a function again. However, the store s is fixed in the definition. Later on, we will introduce an operator that also allows decomposition of the store.

The following characterisation of supported elements is novel.

Theorem 6.23. *In* AS *an element* $[\![p]\!]$ *is supported iff it satisfies, for all* $[\![q]\!]$ *and* $[\![r]\!]$,

$$([\![p]\!] \mathbin{\dot\cup} [\![q]\!]) \cap ([\![p]\!] \mathbin{\dot\cup} [\![r]\!]) \subseteq [\![p]\!] \mathbin{\dot\cup} ([\![q]\!] \mathbin{\dot\cup} [\![\mathsf{true}]\!] \cap [\![r]\!] \mathbin{\dot\cup} [\![\mathsf{true}]\!]) .$$

The key idea to prove Theorem 6.23 is to use predicates p which precisely describe one heap and one store, i.e., the set $[\![p]\!]$ is a singleton set that contains only a single state. Therefore we state directly $[\![s, h]\!] = \{(s, h)\}$. Before showing the proof we need two auxiliary lemmas. The first is simple set theory — therefore we skip the proof.

Lemma 6.24. *We assume three arbitrary sets A, B and C. If* $B \subseteq C$ *then we have* $C - A \subseteq C - B \Leftrightarrow B \subseteq A$. *Hence, if* $A \subseteq C$ *and* $B \subseteq C$ *then* $C - A = C - B \Leftrightarrow A = B$.

Next we give some simple properties of $[\![s, h]\!]$.

Lemma 6.25. *For arbitrary heaps* h, h'*, store s and assertion p we have*

(a) $s, h' \models [\![s, h]\!] \Leftrightarrow h = h'$. *In particular,* $s, h \models [\![s, h]\!]$.

(b) *If* $h \subseteq h'$ *then* $s, h \models p \Leftrightarrow s, h' \models p * [\![s, h' - h]\!]$.

(c) $s, h' \models [\![s, h]\!] * \mathsf{true} \Leftrightarrow h \subseteq h'$.

The lengthy, but straightforward proof can be found in Appendix A. Now we give a proof of the above theorem.

Proof of Theorem 6.23. For the $\Rightarrow$-direction we assume p is supported.
Let $s, h \models p * q \wedge p * r$. Then

$$\begin{array}{ll}
 & s, h \models p * q \wedge p * r \\
\Leftrightarrow & \{\!\![\text{ definition of } * \,]\!\!\} \\
 & \exists h_1, h_2 : h_1 \subseteq h \wedge h_2 \subseteq h \wedge s, h_1 \models p \wedge s, h - h_1 \models q \\
 & \wedge\, s, h_2 \models p \wedge s, h - h_2 \models r \\
\Rightarrow & \{\!\![\; p \text{ supported}, h_1 \cup h_2 \subseteq h \text{ is a function }]\!\!\} \\
 & \exists h' : s, h' \models p \wedge h' \subseteq h_1 \wedge h' \subseteq h_2 \wedge \\
 & s, h - h_1 \models q \wedge s, h - h_2 \models r \\
\Rightarrow & \{\!\![\; h - h_1 \subseteq h - h', s, h_1 - h' \models \mathsf{true}, \text{ analogously } h_2 \,]\!\!\} \\
 & \exists h' : s, h' \models p \wedge s, h - h' \models q * \mathsf{true} \wedge s, h - h' \models r * \mathsf{true} \\
\Leftrightarrow & \{\!\![\text{ definition of } * \,]\!\!\} \\
 & s, h \models p * (q * \mathsf{true} \wedge r * \mathsf{true})
\end{array}$$

Next we show the other direction. For that we assume, for all q,r,

$$s, h \models p * q \wedge p * r \Rightarrow s, h \models p * (q * \mathsf{true} \wedge r * \mathsf{true}) . \qquad (14)$$

as well as $s, h_1 \models p$ and $s, h_2 \models p$ and $h_1 \cup h_2$ is a function.
From this we calculate

$$\begin{array}{ll}
 & s, h_1 \models p \wedge s, h_2 \models p \\
\Leftrightarrow & \{\!\![\; h_1 \subseteq h_1 \cup h_2 \text{ and } h_2 \subseteq h_1 \cup h_2 \text{ and Lemma 6.25(b) }]\!\!\} \\
 & s, h_1 \cup h_2 \models p * [\![s, (h_1 \cup h_2) - h_1]\!] \wedge \\
 & s, h_1 \cup h_2 \models p * [\![s, (h_1 \cup h_2) - h_2]\!] \\
\Rightarrow & \{\!\![\text{ Assumption (14) }]\!\!\} \\
 & s, h_1 \cup h_2 \models p * ([\![s, (h_1 \cup h_2) - h_1]\!] * \mathsf{true} \wedge [\![s, (h_1 \cup h_2) - h_2]\!] * \mathsf{true}) \\
\Leftrightarrow & \{\!\![\text{ definition of } * \,]\!\!\} \\
 & \exists h' : h' \subseteq h_1 \cup h_2 \wedge s, h' \models p \wedge \\
 & s, (h_1 \cup h_2) - h' \models [\![s, (h_1 \cup h_2) - h_1]\!] * \mathsf{true} \wedge [\![s, (h_1 \cup h_2) - h_2]\!] * \mathsf{true}
\end{array}$$

$\Leftrightarrow$ $\{\!|$ definition of $\wedge$ $|\!\}$
$\exists h' : h' \subseteq h_1 \cup h_2 \;\wedge\; s, h' \models p \;\wedge$
$s, (h_1 \cup h_2) - h' \models [\![s, (h_1 \cup h_2) - h_1]\!] * \mathsf{true} \;\wedge$
$s, (h_1 \cup h_2) - h' \models [\![s, (h_1 \cup h_2) - h_2]\!] * \mathsf{true}$
$\Leftrightarrow$ $\{\!|$ Lemma 6.25(c) (twice) $|\!\}$
$\exists h' : h' \subseteq h_1 \cup h_2 \;\wedge\; s, h' \models p \;\wedge$
$(h_1 \cup h_2) - h_1 \subseteq (h_1 \cup h_2) - h' \;\wedge$
$(h_1 \cup h_2) - h_2 \subseteq (h_1 \cup h_2) - h'$
$\Leftrightarrow$ $\{\!|$ Lemma 6.24 $|\!\}$
$\exists h' : h' \subseteq h_1 \cup h_2 \;\wedge\; s, h' \models p \;\wedge\; h' \subseteq h_1 \;\wedge\; h' \subseteq h_2$
$\Leftrightarrow$ $\{\!|$ set theory $|\!\}$
$\exists h' : s, h' \models p \;\wedge\; h' \subseteq h_1 \;\wedge\; h' \subseteq h_2$

□

As before, this can be lifted to the abstract level of quantales.

Definition 6.26. In an arbitrary Boolean quantale S an element a is *supported* iff it satisfies for arbitrary b,c

$$a \cdot b \sqcap a \cdot c \leq a \cdot (b \cdot \top \sqcap c \cdot \top) .$$

Following this characterisation of supported assertions we now give properties for this class of assertion which can be completely derived algebraically. As before, we refer to the appendix of [13] for further properties.

Lemma 6.27. *If a is pure then it is also supported.*

Proof. By Lemma 6.12(a), associativity, commutativity and idempotence of $\sqcap$, isotony, and Lemma 6.12(a) again:

$$\begin{array}{ll} & a \cdot b \sqcap a \cdot c \\ = & (a \sqcap b \cdot \top) \sqcap (a \sqcap c \cdot \top) \\ = & a \sqcap (b \cdot \top \sqcap c \cdot \top) \\ \leq & a \sqcap (b \cdot \top \sqcap c \cdot \top) \cdot \top \\ = & a \cdot (b \cdot \top \sqcap c \cdot \top) \end{array}$$

□

Lemma 6.28. *a is precise implies a is supported.*

Proof. $a \cdot b \sqcap a \cdot c \leq a \cdot (b \sqcap c) \leq a \cdot (b \cdot \top \sqcap c \cdot \top)$. This holds by the definition of precise elements and isotony. □

Lemma 6.29. *Supported elements are closed under $\cdot$.*

Proof. For supported elements a and a' we calculate, using the definition of supported elements and isotony,

$$\begin{array}{rcl} a \cdot a' \cdot b \sqcap a \cdot a' \cdot c & \leq & a \cdot (a' \cdot b \cdot \top \sqcap a' \cdot c \cdot \top) \\ & \leq & a \cdot a \;\cdot (b \cdot \top \cdot \top \sqcap c \cdot \top \cdot \top) \\ & \leq & a \cdot a' \cdot (b \cdot \top \sqcap c \cdot \top) \end{array}$$

□

Corollary 6.30. *If a is supported and b is precise or pure then $a \cdot b$ is supported.*

Using this and Corollary 6.7, we obtain

Corollary 6.31. *a is precise implies $a \cdot \top$ is supported.*

7. Commands

After dealing with the assertions of separation logic, we now turn to the commands in the simple imperative language associated with it.

7.1. Commands as Relations

Semantically, we use the common technique of modelling commands as relations between states.

Definition 7.1. A *command* is a relation $C \in \mathit{Cmds} =_{df} \mathcal{P}(\mathit{States} \times \mathit{States})$.

Therefore, all relational operations, including sequential composition ; and reflexive-transitive closure *, are available for commands. Moreover, the structure $(\mathit{Cmds}, \subseteq, ;, I)$, where I is the identity relation, forms a Boolean quantale. The greatest element $\top$ is the universal relation.

In using relations for program semantics, some care has to be taken to correctly treat the possibility of errors in program execution. In the standard literature on separation logic this is done by introducing a special state *fault* which is the "result" of such an error, e.g., access to an unallocated or uninitialised memory cell or to a variable without value.

The precise treatment of the *fault* state leads, however, to a lot of detail that is not really essential to the semantics proper. Therefore, for the purposes of this paper, we ignore these phenomena and deal only with the partial correctness semantics of program constructs.

A detailed treatment dealing with the possibility of program faults is provided by the well known approach of *demonic relational semantics* (see e.g. [2, 14, 15, 40] and [17] for a more abstract algebraic treatment in terms of idempotent semirings and Kleene algebras). There a total correctness view is taken: a state belongs to the domain of a command relation iff no execution starting from it may lead to an error. Hence there is no need to include *fault* into the set of states. As a price one has to pay, the relation composition operators become more complex: instead of the usual (angelic) operators of union and sequential composition one has to use their demonic variants. As we will see, our techniques for the partial correctness semantics can be adapted to the demonic setting; spelling out the details would fill a paper of its own, though. An alternative would be to use monotonic predicate transformers [44]; however, this would step outside the current relational framework.

7.2. Tests and Hoare Triples

In partial correctness semantics one uses Hoare triples $\{p\}\,C\,\{q\}$ where p and q are predicates about states and C is a command. In the semiring and quantale setting the predicates for pre- and postconditions of such triples can be modelled by tests (cf. Definition 4.3).

In the command quantale tests are given by the partial identity relations of the form $\widehat{P} = \{(\sigma, \sigma) \mid \sigma \in P\}$ for some set P of states. It is clear that these subidentities, sets of states and predicates characterising states are in one-to-one correspondence. The set P can be retrieved from $\widehat{P}$ as $P = \mathit{dom}(\widehat{P})$. Because of this isomorphism, for a command C we simply write $\mathit{dom}(C)$ instead of $\widehat{\mathit{dom}(C)}$.

Definition 7.2. Employing tests, we can give the following equivalent definitions of the meaning of *Hoare triples* (e.g. [29]):

$$\{p\}\,C\,\{q\} \Leftrightarrow p\,;C\,;\neg q \subseteq \emptyset \Leftrightarrow p\,;C \subseteq C\,;q \Leftrightarrow p\,;C = p\,;C\,;q\,.$$

To use this general approach to Hoare logic we simply need to embed the quantale of assertions from the previous sections into the set of tests of the relational quantale by the above correspondence between sets of states and partial identity relations. The operation $\cup\!\!\!\cdot$ on assertions is lifted to tests by

$$\widehat{P} \cup\!\!\!\cdot\, \widehat{Q} =_{df} \widehat{P \cup\!\!\!\cdot\, Q}\,.$$

Since we are working in a semiring with greatest element $\top$ we can give the following further useful equivalent formulations of Hoare triples.

Lemma 7.3. $\{p\}\,C\,\{q\} \Leftrightarrow \top\,;p\,;C \subseteq \top\,;q \Leftrightarrow p\,;C \subseteq \top\,;q$.

The proof can be found in the Appendix.

7.3. Syntax and Semantics of a Simple Programming Language

We now introduce the program constructs associated with separation logic [46]. Syntactically, they are given by

$$
\begin{array}{rcl}
\mathit{comm} & ::= & \mathit{var} := \mathit{exp} \mid \mathsf{skip} \mid \mathit{comm}\,;\mathit{comm} \\
& & \mid \mathsf{if}\,\mathit{bexp}\ \mathsf{then}\,\mathit{comm}\ \mathsf{else}\,\mathit{comm} \mid \mathsf{while}\,\mathit{bexp}\ \mathsf{do}\,\mathit{comm} \\
& & \mid \mathsf{newvar}\,\mathit{var}\ \mathsf{in}\,\mathit{comm} \mid \mathsf{newvar}\,\mathit{var} := \mathit{exp}\ \mathsf{in}\,\mathit{comm} \\
& & \mid \mathit{var} := \mathsf{cons}\,(\mathit{exp},\ldots,\mathit{exp}) \\
& & \mid \mathit{var} := [\mathit{exp}] \mid [\mathit{exp}] := \mathit{exp} \\
& & \mid \mathsf{dispose}\,\mathit{exp}
\end{array}
$$

Let us explain these constructs informally, where we skip the well-known first three lines. For a (boolean) expression e we denote its value w.r.t. a store s by e^S.

In a given state s the command $v := \mathsf{cons}\,(e_1,...,e_n)$ allocates n cells with e_i^S as the contents of the i-th cell. The cells have to form an unused contiguous region somewhere on the heap; the concrete allocation is chosen non-deterministically [50].

The address of the first cell is stored in v while the rest of the cells can be addressed indirectly via the start address.

A dereferencing assignment $v := [e]$ assumes that e is an *exp*-expression and the value e^S (corresponding to `*e` in `C`) is an allocated address on the heap, i.e., $e^S \in \mathit{dom}(h)$ for the current heap h. In particular, after its execution, the value of v is the contents of a dereferenced heap cell.

Conversely, an execution of $[e_1] := e_2$ assigns the value of the expression on the right hand side to the cell whose address is the value of the left hand side.

Finally, the command $\mathsf{dispose}\,e$ is used for deallocating the heap cell with address e^S. After its execution the disposed cell is not valid any more, i.e., dereferencing that cell would cause a fault in the program execution.

We now define a formal semantics for this language by inductively assigning to every program P formed according to the above grammar a command $[\![P]\!]_c \in \mathit{Cmds}$ in the sense of the previous section. In particular, $[\![\mathsf{skip}]\!]_c =_{df} I$. For all language constructs we also specify the correctness conditions, like that the free variables of all program parts are within the domains of the stores involved and that only allocated heap cells may be accessed.

As auxiliaries we define the functions FV and MV that assign to every program P the set of its free and modified variables, resp., where in the case of FV we assume that for (boolean) expressions a corresponding function with the same name is predefined.

$\mathrm{FV}(v := e)$	$=_{df} \{v\} \cup \mathrm{FV}(e)$	$\mathrm{MV}(v := e)$	$=_{df} \{v\}$
$\mathrm{FV}(\mathsf{skip})$	$=_{df} \emptyset$	$\mathrm{MV}(\mathsf{skip})$	$=_{df} \emptyset$
$\mathrm{FV}(P\,;Q)$	$=_{df} \mathrm{FV}(P) \cup \mathrm{FV}(Q)$	$\mathrm{MV}(P\,;Q)$	$=_{df} \mathrm{MV}(P) \cup \mathrm{MV}(Q)$
$\mathrm{FV}(\mathsf{if}\,b\,\mathsf{then}\ P\ \mathsf{else}\ Q)$	$=_{df} \mathrm{FV}(b) \cup \mathrm{FV}(P) \cup \mathrm{FV}(Q)$	$\mathrm{MV}(\mathsf{if}\,b\,\mathsf{then}\ P\ \mathsf{else}\ Q)$	$=_{df} \mathrm{MV}(P) \cup \mathrm{MV}(Q)$
$\mathrm{FV}(\mathsf{while}\,b\,\mathsf{do}\ P)$	$=_{df} \mathrm{FV}(b) \cup \mathrm{FV}(P)$	$\mathrm{MV}(\mathsf{while}\,b\,\mathsf{do}\ P)$	$=_{df} \mathrm{MV}(P)$
$\mathrm{FV}(\mathsf{newvar}\,v\,\mathsf{in}\,P)$	$=_{df} \mathrm{FV}(P) - \{v\}$	$\mathrm{MV}(\mathsf{newvar}\,v\,\mathsf{in}\,P)$	$=_{df} \mathrm{MV}(P) - \{v\}$
$\mathrm{FV}(\mathsf{newvar}\,v := e\,\mathsf{in}\,P)$	$=_{df} (\mathrm{FV}(e) \cup \mathrm{FV}(P)) - \{v\}$	$\mathrm{MV}(\mathsf{newvar}\,v := e\,\mathsf{in}\,P)$	$=_{df} \mathrm{MV}(P) - \{v\}$
$\mathrm{FV}(v := \mathsf{cons}\,(e_1,...,e_n))$	$=_{df} \{v\} \cup \bigcup_{i=1}^{n} \mathrm{FV}(e_i)$	$\mathrm{MV}(v := \mathsf{cons}\,(e_1,...,e_n))$	$=_{df} \{v\}$
$\mathrm{FV}(v := [e])$	$=_{df} \{v\} \cup \mathrm{FV}(e)$	$\mathrm{MV}(v := [e])$	$=_{df} \{v\}$
$\mathrm{FV}([e_1] := e_2)$	$=_{df} \mathrm{FV}(e_1) \cup \mathrm{FV}(e_2)$	$\mathrm{MV}([e_1] := e_2)$	$=_{df} \emptyset$
$\mathrm{FV}(\mathsf{dispose}\ e)$	$=_{df} \mathrm{FV}(e)$	$\mathrm{MV}(\mathsf{dispose}\ e)$	$=_{df} \emptyset$

To abbreviate the semantic definition, we use a convention similar to that of the refinement calculus (e.g. [1]). We characterise relations by formulas linking the input states (s,h) and output states (s',h'). If F is such a formula then $R \mathrel{\widehat{=}} F$ abbreviates the clause $(s,h)\,R\,(s',h') \Leftrightarrow_{df} F$. For use in the definition of if and while we also assign a semantics to boolean expressions b viewed as assertion commands.

$$\begin{aligned}
[\![v := e]\!]_c &\mathrel{\widehat{=}} \{v\} \cup \mathrm{FV}(e) \subseteq dom(s) \ \wedge\ s' = (v, e^S) \,|\, s \ \wedge\ h' = h\,,\\
[\![b]\!]_c &\mathrel{\widehat{=}} \mathrm{FV}(b) \subseteq dom(s) \ \wedge\ b^S = \mathsf{true} \ \wedge\ s' = s\,,\\
[\![P\,;Q]\!]_c &\mathrel{\widehat{=}} \mathrm{FV}(P) \ \cup\ \mathrm{FV}(Q) \subseteq dom(s) \ \wedge\ s\,S\,s'\\
&\qquad \text{where } S \ = \ [\![P]\!]_c\,;[\![Q]\!]_c\,,\\
[\![\,\mathsf{if}\, b\, \mathsf{then}\, P\ \mathsf{else}\ Q]\!]_c &\mathrel{\widehat{=}} \mathrm{FV}(b) \cup \mathrm{FV}(P) \ \cup\ \mathrm{FV}(Q) \subseteq dom(s) \ \wedge\ s\,S\,s'\\
&\qquad \text{where } S \ = \ [\![b]\!]_c\,;[\![P]\!]_c \cup \neg[\![b]\!]_c\,;[\![Q]\!]_c\,,\\
[\![\,\mathsf{while}\, b\, \mathsf{do}\, P]\!]_c &\mathrel{\widehat{=}} \mathrm{FV}(b) \cup \mathrm{FV}(P) \subseteq dom(s) \ \wedge\ s\,S\,s'\\
&\qquad \text{where } S \ = \ ([\![b]\!]_c\,;[\![P]\!]_c)^*\,;\neg[\![b]\!]_c\,,\\
[\![\,\mathsf{newvar}\, v\, \mathsf{in}\, P]\!]_c &\mathrel{\widehat{=}} v \notin dom(s) \ \wedge\ \exists\, i \in Values : ((v,i) \,|\, s, h)\, [\![P]\!]_c\, (s', h')\,,\\
[\![\,\mathsf{newvar}\, v := e\, \mathsf{in}\, P]\!]_c &\mathrel{\widehat{=}} v \notin (\mathrm{FV}(e) \cup dom(s)) \ \wedge\ \mathrm{FV}(e) \subseteq dom(s) \ \wedge\ ((v, e^S) \,|\, s, h)\, [\![P]\!]_c\, (s', h')\,,\\
[\![v := \ \mathsf{cons}\,(e_1, ..., e_n)]\!]_c &\mathrel{\widehat{=}} \exists\, a \in Addresses : s' = (v, a) \,|\, s \ \wedge\\
&\qquad a, \ldots, a+n-1 \notin dom(h) \wedge\\
&\qquad h' = \{(a, e_1^S), \ldots, (a+n-1, e_n^S)\} \,|\, h\,,\\
[\![v := [e]]\!]_c &\mathrel{\widehat{=}} \{v\} \cup \mathrm{FV}(e) \subseteq dom(s) \ \wedge\ s' = (v, h(e^S)) \,|\, s \ \ \wedge\ \ h' = h\,,\\
[\![[e_1] := e_2]\!]_c &\mathrel{\widehat{=}} \mathrm{FV}(e_1) \cup \mathrm{FV}(e_2) \subseteq dom(s) \ \wedge\ s' = s \ \wedge\ h' = (e_1^S, e_2^S) \,|\, h \ \ \wedge\ e_1^S \in dom(h)\,,\\
[\![\,\mathsf{dispose}\ e]\!]_c &\mathrel{\widehat{=}} \mathrm{FV}(e) \subseteq dom(s) \ \wedge\ s' = s \ \wedge\ e^S \in dom(h) \ \wedge\ h' = h - \{(e^S, h(e^S))\}\,.
\end{aligned}$$

We now discuss inference rules for commands. By the above abstraction these rules become simple consequences of the well established general relational view of commands. We only sketch the rule for *mutation*.

Theorem 7.4. *Let p be an assertion and e, e' be exp-expressions. Then valid triples for mutation commands are*

$$\begin{aligned}
\{(e_1 \ \mapsto\ -)\}\ \ [e_1] := e_2\ \{(e_1 \ \mapsto\ e_2)\}\,, &\qquad \text{(local)}\\
\{(e_1 \ \mapsto\ -) * p\}\ \ [e_1] := e_2\ \{(e_1 \ \mapsto\ e_2) * p\}\,, &\qquad \text{(global)}\\
\{(e_1 \ \mapsto\ -) * ((e_1 \ \mapsto\ e_2) -\!\!* p)\}\ \ [e_1] := e_2\ \{p\} &\qquad \text{(backward reasoning)}
\end{aligned}$$

where $(e_1 \ \mapsto\ -)$ abbreviates $\exists\, v.\, e_1 \ \mapsto\ v$.

The local mutation law ensures that the cell at address e_1^S has the value e_2^S after $[e_1] := e_2$ has been executed; provided there is an allocated cell at e_1^S. In the relational semantics the precondition is satisfied by $e_1^S \in dom(h)$. The global mutation law follows from the local one as an instantiation of the frame rule. It implies that reasoning can be modularised. Conversely, the local rule can be derived from the global one by setting $p = \ \mathsf{emp}$. The backward reasoning law is used to determine a precondition that ensures the postcondition p. The global and the backward reasoning rules are again interderivable (see [46, 11]). Further inference rules, like for disposal, overwriting and non-overwriting allocation as well as lookups are presented and explained in [46]; the algebraic treatment is similar to the one of mutation and straightforward.

The most central rule of separation logic, however, will be treated in detail in the following section.

8. The Frame Rule

An important ingredient of the separation calculus is the *frame rule* [42]. It describes the use of the separating conjunction in pre- and postconditions for commands and allows local reasoning. For assertions p, q and r and command C it reads

$$\frac{\{p\}\,C\,\{q\}}{\{p * r\}\,C\,\{q * r\}}\,.$$

The premise ensures that starting the execution of C in a state satisfying p ends in a state satisfying q. Furthermore the conclusion says that extending the initial and final heaps consistently with disjoint heaps will not invalidate the triple in the premise. Hence a "local" proof of $\{p\}\,C\,\{q\}$ will extend to a "more global" one in the additional heap context r. A standard proof [50] uses three assumptions: A side condition stating that no modified variable is changed by r, *safety monotonicity* and the *frame property*. Safety monotonicity guarantees that if C is executable from a state, it can also run on a state with a larger heap; the frame property states that every execution of C can be tracked back to an execution of C running on states with a possibly smaller heap.

We will present a modified, more generally applicable rule using algebraic counterparts for the conditions involved.

8.1. Separation Algebras

Most of the calculations in the soundness proof of the frame rule can be done at an abstract level. Therefore in this section we abstract from the concrete separation logic instance of *States* and the separating conjunction. We will also model the required conditions for commands that satisfy the frame rule at an abstract level. This captures a wider range of operations with behaviour similar to the separating conjunction. The following definition and its notation are inspired by [8].

Definition 8.1. A *separation algebra* is a partial commutative monoid $(\Sigma, \bullet, u)$. A partial commutative monoid is given by a partial binary operation where the commutativity, associativity and unit laws hold for the equality that means both sides are defined and equal, or both are undefined. The induced *combinability* relation # is given by

$$\sigma_0 \,\#\, \sigma_1 \;\Leftrightarrow_{df}\; \sigma_0 \bullet \sigma_1 \text{ is defined}$$

As before, a *command* is a relation $C \subseteq \Sigma \times \Sigma$.

An example for a separation algebra is $(States, *, [\![\mathsf{emp}]\!]_c)$. Later we will use this particular algebra to illustrate the frame rule.

Over every separation algebra we can define the split and join relations:

Definition 8.2.

(a) The *split* relation $\lhd \subseteq \Sigma \times (\Sigma \times \Sigma)$ w.r.t. $\bullet$ is given by

$$\sigma \lhd (\sigma_1, \sigma_2) \;\Leftrightarrow_{df}\; \sigma_1 \,\#\, \sigma_2 \;\wedge\; \sigma = \sigma_1 \bullet \sigma_2$$

(b) The *join* relation $\rhd$ is the converse of split, i.e.,

$$(\sigma_1, \sigma_2) \rhd \sigma \;\Leftrightarrow_{df}\; \sigma_1 \,\#\, \sigma_2 \;\wedge\; \sigma = \sigma_1 \bullet \sigma_2$$

We introduce a special symbol for it, rather than writing $\lhd^{\smile}$, to ease reading.

(c) The *Cartesian product* $R_1 \times R_2 \subseteq (\Sigma \times \Sigma) \times (\Sigma \times \Sigma)$ of two commands R_1, R_2 is defined by

$$(\sigma_1, \sigma_2)\,(R_1 \times R_2)\,(\tau_1, \tau_2) \;\Leftrightarrow_{df}\; \sigma_1\, R_1\, \tau_1 \;\wedge\; \sigma_2\, R_2\, \tau_2\;.$$

Sequential composition on Cartesian products is defined componentwise.

Next we lift the operations $\bullet$ and # to commands.

(d) The $\bullet$ *composition* $R_1 \bullet R_2$ of commands $R_1, R_2 \subseteq \Sigma \times \Sigma$ is again a command defined by

$$R_1 \bullet R_2 \;=_{df}\; \lhd\; ;\; (R_1 \times R_2)\; ;\; \rhd$$

By associativity and commutativity of $\bullet$ on Σ, the lifted operation is associative and commutative, too. For tests p, q the $\bullet$ composition $p \bullet q$ is a test again, irrespective of the underlying separation algebra.

(e) The partial identity $\# \subseteq (\Sigma \times \Sigma) \times (\Sigma \times \Sigma)$ characterises those pairs of states which are combinable:

$$(\sigma_1, \sigma_2)\,\#\,(\tau_1, \tau_2) \;\Leftrightarrow_{df}\; \sigma_1 \,\#\, \sigma_2 \;\wedge\; \sigma_1 = \tau_1 \;\wedge\; \sigma_2 = \tau_2.$$

It is well known that $\times$ and ; satisfy an exchange law:

$$(R_1 \times R_2)\,;(S_1 \times S_2) \;=\; (R_1\,; S_1) \times (R_2\,; S_2)\,, \tag{15}$$

We have the following properties of split, join and compatibility:

$$\lhd = \lhd\,;\#\,, \qquad \rhd = \#\,;\rhd\,. \tag{16}$$

Moreover, from the definition and the unit property of u it is clear that $\lhd\,;\rhd = I$.

In the concrete separation algebra $(States, *, [\![\mathsf{emp}]\!]_c)$ the split relation $\lhd$ means that the a state (s,h) is split into two states (s,h_1) and (s,h_2) with $h = h_1 \uplus h_2$; more precisely, only the heap is split into disjoint parts. By this, the separating conjunction $p \uplus q$ of tests p, q in the command quantale over *States* coincides with $p * q = \lhd\,;(p \times q)\,;\rhd$.

The following properties will be used later on.

Lemma 8.3.

(a) $\top\,;\lhd = \lhd\,;(\top \times \top)\,;\#$.

(b) For tests p, q we have $\top\,;\lhd\,;(p \times q)\,;\# \;=\; \lhd\,;((\top\,;p) \times (\top\,;q))\,;\#$.
Therefore, $\top\,;(p \bullet q) = (\top\,;p) \bullet (\top\,;q)$.

Proof.

(a) First, $\sigma\,(\top\,;\lhd)\,(\rho_1,\rho_2) \;\Leftrightarrow\; \exists\rho : \sigma \top \rho \;\wedge\; \rho_1 \# \rho_2 \;\wedge\; \rho = \rho_1 \bullet \rho_2 \;\Leftrightarrow\; \rho_1 \# \rho_2 \;\wedge\; \exists\rho : \rho = \rho_1 \bullet \rho_2 \;\Leftrightarrow\; \rho_1 \# \rho_2$.
Second,

$$\begin{aligned}
&\sigma\,(\lhd\,;(\top \times \top)\,;\#)\,(\rho_1,\rho_2)\\
&\quad\Leftrightarrow\; \exists\sigma_1,\sigma_2 : \sigma_1 \# \sigma_2 \;\wedge\; \sigma = \sigma_1 \bullet \sigma_2 \;\wedge\; \sigma_1 \top \rho_1 \;\wedge\; \sigma_2 \top \rho_2 \;\wedge\; \rho_1 \# \rho_2\\
&\quad\Leftrightarrow\; \exists\sigma_1,\sigma_2 : \sigma_1 \# \sigma_2 \;\wedge\; \sigma = \sigma_1 \bullet \sigma_2 \;\wedge\; \rho_1 \# \rho_2\\
&\quad\Leftrightarrow\; \rho_1 \# \rho_2\,,
\end{aligned}$$

since we can choose $\sigma_1 = \sigma$ and $\sigma_2 = u$.

(b) Straightforward from Part (a), exchange (15) and the definition of $\bullet$ on relations. □

8.2. The Frame Property

We now turn to an algebraic characterisation of the above-mentioned frame property. In this and the following subsection we are working within the concrete separation algebra $(States, *, [\![\mathsf{emp}]\!]_c)$ for better motivation of the development; afterwards we will generalise. Intuitively, the frame property expresses that an execution of a command C can be tracked back to a possibly smaller heap portion that is sufficient for its execution. This means that certain variables may change while certain parts of the heap are preserved. This means that care has to be taken when reassembling the overall result state from that resulting from the computation on the smaller portion.

As an example, take the command $C = (x := 1)$ and a starting state $\sigma = (s,h)$ with $s(x) = 2$. Suppose that σ splits into states $\sigma_1 = (s_1,h_1), \sigma_2 = (s_2,h_2)$ which then satisfy $s_1(x) = s_2(x) = 2$. Clearly, C can act on the smaller state σ_1, delivering some result state $\tau_1 = (t_1,k_1)$ with $t_1(x) = 1$. However, this cannot be combined with the "remainder" σ_2 of the original state σ.

To compensate for this aspect we introduce a special command.

Definition 8.4. Let H be the command that preserves all heaps while being liberal about the stores:

$$(s,h)\,H\,(s',h') \;\Leftrightarrow_{df}\; h = h'\,.$$

It is clear that H is an equivalence relation.

Using H, we can formulate the frame property for the $*$ case algebraically.

Definition 8.5. Command C has the $*$*-frame-property* iff

$$(dom(C) \times I)\,;\, \mathrel{\rhd\!\!\!\!*}\,;\; C \;\subseteq\; (C \times H)\,;\, \mathrel{\rhd\!\!\!\!*}\,.$$

Prefixing the join operator with $dom(C) \times I$ on the left-hand side ensures that the property need only hold in those cases where C can indeed act on a smaller state. Pointwise, for arbitrary $s, s' \in Stores$ and $h_0, h_1, h' \in Heaps$, this definition spells out as

$$\begin{array}{ll} & (s, h_0) \in dom(C) \;\wedge\; h_0 \cap h_1 = \emptyset \;\wedge\; (s, h_0 \cup h_1)\, C\, (s', h') \\ \Rightarrow & \exists\, h_0', h_1' : \; dom(h_0') \cap dom(h_1') = \emptyset \;\wedge\; h' = h_0' \cup h_1' \;\wedge\; (s, h_1)\, H\, (s', h_1') \;\wedge\; (s, h_0 \cup h_1)\, C\, (s', h')\,. \end{array}$$

By the definition of H the conclusion simplifies to

$$\exists\, h_0' : \; dom(h_0') \cap dom(h_1) = \emptyset \;\wedge\; (s, h_0 \cup h_1)\, C\, (s', h_0' \cup h_1)\,.$$

Informally, this describes that the heap storage not being used by C will remain unchanged while C may change certain variables of the starting state (s, h_1).

As a check for adequacy we show exemplarily that the mutation command $[e_1] := e_2$ satisfies the frame property. To ease reading, we denote the heap of a state σ by h_σ. Moreover, none of the mutation commands change the stores and $\mathrel{\rhd\!\!\!\!*}$ assumes the stores of the considered states to be equal, hence all stores of all states that occur in the proof are the same, denoted by s. Since $I \subseteq H$, it suffices to show the frame property with $([\![[e_1] := e_2]\!]_c \times I)\,;\mathrel{\rhd\!\!\!\!*}$ instead of $([\![[e_1] := e_2]\!]_c \times H)\,;\mathrel{\rhd\!\!\!\!*}$ on the right hand side.

Moreover by the definition of $[\![[e_1] := e_2]\!]_c$, we immediately have

$$\sigma = (s, h) \in dom([\![[e_1] := e_2]\!]_c) \;\Leftrightarrow\; e_1^s \in dom(h) \tag{17}$$

From this and setting $s = s_{\sigma_1 * \sigma_2} = s_{\sigma_1}$, we get

$$\begin{array}{ll}
& (\sigma_1, \sigma_2)\ (dom([\![[e_1] := e_2]\!]_c) \times I)\,;\mathrel{\rhd\!\!\!\!*}\,;\, [\![[e_1] := e_2]\!]_c\ \tau \\
\Leftrightarrow & \{\!|\ \text{definition of } \times \text{ and } I\ |\!\} \\
& \sigma_1 \in dom([\![[e_1] := e_2]\!]_c) \;\wedge\; (\sigma_1, \sigma_2)\ \mathrel{\rhd\!\!\!\!*}\,; [\![[e_1] := e_2]\!]_c\ \tau \\
\Leftrightarrow & \{\!|\ \text{definition of } \mathrel{\rhd\!\!\!\!*} \text{ and (17)}\ |\!\} \\
& (\sigma_1 * \sigma_2)\ [\![[e_1] := e_2]\!]_c\ \tau \;\wedge\; \sigma_1 \,\#\, \sigma_2 \;\wedge\; e_1^s \in dom(h_{\sigma_1}) \\
\Leftrightarrow & \{\!|\ \text{definition of } [\![[e_1] := e_2]\!]_c\ |\!\} \\
& h_\tau = (e_1^s, e_2^s) \,|\, h_{(\sigma_1 * \sigma_2)} \;\wedge\; e_1^s \in dom(h_{(\sigma_1 * \sigma_2)}) \;\wedge\; \sigma_1 \,\#\, \sigma_2 \;\wedge\; e_1^s \in dom(h_{\sigma_1}) \\
\Leftrightarrow & \{\!|\ \text{definition of } *\ |\!\} \\
& h_\tau = (e_1^s, e_2^s) \,|\, (h_{\sigma_1} \,\cup\, h_{\sigma_2}) \;\wedge\; dom(h_{\sigma_1}) \,\cap\, dom(h_{\sigma_2}) = \emptyset \,\wedge \\
& e_1^s \in dom(h_{\sigma_1}) \cup dom(h_{\sigma_2}) \;\wedge\; \sigma_1 \,\#\, \sigma_2 \;\wedge\; e_1^s \in dom(h_{\sigma_1}) \\
\Leftrightarrow & \{\!|\ \text{localisation property for } | \text{ (see below), since } e_1^s \notin dom(h_{\sigma_2}), \text{ isotony of } dom\ |\!\} \\
& h_\tau = ((e_1^s, e_2^s) \,|\, h_{\sigma_1}) \,\cup\, h_{\sigma_2} \;\wedge\; dom(h_{\sigma_1}) \,\cap\, dom(h_{\sigma_2}) = \emptyset \;\wedge\; \sigma_1 \,\#\, \sigma_2 \;\wedge\; e_1^s \in dom(h_{\sigma_1}) \\
\Rightarrow & \{\!|\ \text{since } e_1^s \notin dom(h_{\sigma_2})\ |\!\} \\
& h_\tau = ((e_1^s, e_2^s) \,|\, h_{\sigma_1}) \,\cup\, h_{\sigma_2} \;\wedge\; dom((e_1^s, e_2^s) \,|\, h_{\sigma_1}) \,\cap\, dom(h_{\sigma_2}) = \emptyset \;\wedge\; \sigma_1 \,\#\, \sigma_2 \;\wedge\; e_1^s \in dom(h_{\sigma_1}) \\
\Rightarrow & \{\!|\ \text{logic, definition of } [\![[e_1] := e_2]\!]_c \text{ and } *\ |\!\} \\
& \sigma_1\ [\![[e_1] := e_2]\!]_c\ (s, (e_1^s, e_2^s) \,|\, h_{\sigma_1}) \;\wedge\; \tau = (s, (e_1^s, e_2^s) \,|\, h_{\sigma_1}) * \sigma_2 \;\wedge\; (s, (e_1^s, e_2^s) \,|\, h_{\sigma_1}) \,\#\, \sigma_2 \\
\Leftrightarrow & \{\!|\ \text{definitions}\ |\!\} \\
& (\sigma_1, \sigma_2)\ ([\![[e_1] := e_2]\!]_c \times I)\,;\mathrel{\rhd\!\!\!\!*}\, \tau\,.
\end{array}$$

The localisation property states that for partial functions $f_1 \,|\, (f_2 \cup f_3) = (f_1 \,|\, f_2) \cup f_3$ provided $dom(f_1) \cap dom(f_3) = \emptyset$. A proof can be found in [33].

8.3. Preservation of Variables

It remains to express algebraically the requirement that a command preserves certain variables. In this, we would like to avoid an explicit mention of syntax and free variables and find a suitable purely algebraic condition instead. In the proof of the frame rule it will be applied after the command under consideration has been placed in parallel composition with H after application of the frame property. This "context" is taken care of by the following definition, where $\top$ is again the universal relation.

Definition 8.6. Command C *$*$-preserves* test r iff

$$\lhd\!\!\ast \,;\, (C \times (r \,;\, H)) \,;\, \# \;\subseteq\; \top \,;\, \lhd\!\!\ast \,;\, (I \times r)\,.$$

The informal explanation is straightforward: when C is run in parallel with something starting in r, every re-assembled result state must contain an r-part, too. Equivalently, we can replace the right-hand side by $\top \,;\, \lhd\!\!\ast \,;\, (I \times r) \,;\, \#$. Moreover the definition of preservation is also equivalent to

$$\# \,;\, (C \times (r \,;\, H)) \,;\, \# \;\subseteq\; \# \,;\, (\top \times (\top \,;\, r)) \,;\, \#\ . \tag{18}$$

This allows simplifying proofs of preservation for concrete commands.

We now show that our notion of preservation fits well with the original side condition of the frame rule. That condition is $\mathrm{MV}(C) \cap \mathrm{FV}(r) = \emptyset$. It is trivially satisfied for all r if $\mathrm{MV}(C) = \emptyset$. Commands C with this property are, e.g., the mutation command $[e_1] := e_2$ and `dispose`. We show that this condition implies our notion of preservation, so that it is adequate, but also more liberal than the original one.

Lemma 8.7. *If* $\mathrm{MV}(C) = \emptyset$ *then C $*$-preserves all tests r.*

Proof. The assumption implies that C cannot change the store part of any state, but only heap parts. Formally,

$$\forall\, \sigma, \tau : \sigma\, C\, \tau \;\Rightarrow\; s_\sigma = s_\tau\ . \tag{19}$$

Now we calculate

$$\begin{array}{ll}
 & (\sigma_1, \sigma_2)\ \ \# \,;\, (C \times (r \,;\, H)) \,;\, \#\ (\tau_1, \tau_2) \\
\Leftrightarrow & \quad \{\!\![\ \text{definitions}\ ]\!\!\} \\
 & \sigma_1 \# \sigma_2 \;\wedge\; \sigma_1\, C\, \tau_1 \;\wedge\; \sigma_2\, (r \,;\, H)\, \tau_2 \;\wedge\; \tau_1 \# \tau_2 \\
\Leftrightarrow & \quad \{\!\![\ \text{definition of ; and } r \text{ is subidentity}\ ]\!\!\} \\
 & \sigma_1 \# \sigma_2 \;\wedge\; \sigma_1\, C\, \tau_1 \;\wedge\; \sigma_2 \in r \;\wedge\; \sigma_2\, H\, \tau_2 \;\wedge\; \tau_1 \# \tau_2 \\
\Leftrightarrow & \quad \{\!\![\ \text{by } \tau_1 \# \tau_2 \wedge \sigma_1 \# \sigma_2\text{, assumption (19) implies } s_{\sigma_2} = s_{\tau_2}\ ]\!\!\} \\
 & \sigma_1 \# \sigma_2 \;\wedge\; \sigma_1\, C\, \tau_1 \;\wedge\; \sigma_2 \in r \;\wedge\; \sigma_2\, H\, \tau_2 \;\wedge\; s_{\sigma_2} = s_{\tau_2} \;\wedge\; \tau_1 \# \tau_2 \\
\Rightarrow & \quad \{\!\![\ \text{the definition of } H \text{ and } s_{\sigma_2} = s_{\tau_2} \text{ imply } \sigma_2 = \tau_2\ ]\!\!\} \\
 & \sigma_1 \# \sigma_2 \;\wedge\; \sigma_1\, C\, \tau_1 \;\wedge\; \sigma_2 \in r \;\wedge\; \sigma_2 = \tau_2 \;\wedge\; \tau_1 \# \tau_2 \\
\Rightarrow & \quad \{\!\![\ \text{logic and omitting conjuncts}\ ]\!\!\} \\
 & \sigma_1 \# \sigma_2 \;\wedge\; \tau_2 \in r \;\wedge\; \tau_1 \# \tau_2 \\
\Leftrightarrow & \quad \{\!\![\ \text{definition of } \top\ ]\!\!\} \\
 & \sigma_1 \# \sigma_2 \;\wedge\; \sigma_1 \top \tau_1 \;\wedge\; \sigma_2 \top \tau_2 \;\wedge\; \tau_2 \in r \;\wedge\; \tau_1 \# \tau_2 \\
\Leftrightarrow & \quad \{\!\![\ \text{definitions}\ ]\!\!\} \\
 & (\sigma_1, \sigma_2)\ \ \# \,;\, (\top \times (\top \,;\, r)) \,;\, \#\ (\tau_1, \tau_2)\ .
\end{array}$$

□

A similar treatment of the general condition $\mathrm{MV}(C) \cap \mathrm{FV}(r) = \emptyset$ is possible; we omit the technicalities.

8.4. A General Frame Rule

In this section we give a completely algebraic soundness proof of the standard frame rule, generalised to separation algebras. In particular, the relation H will be replaced by a generic one that again compensates for incongruities, possibly both on store and heap.

Definition 8.8. Given a separation algebra $(\Sigma, \bullet, u)$, a relation $K \subseteq \Sigma \times \Sigma$ is called a *compensator* iff it satisfies the following properties:

(a) $cod(K) \subseteq dom(K) \subseteq K$,

(b) $\# ; (I \times K) ; \# = \#$,

(c) $K ; K = K$.

Requirement (a) ensures that arbitrarily long sequences of commands can be "accompanied" by equally long compensator sequences. Requirement (b) enforces that K does not deviate too far from simple equality. This and Requirement (c) r will be used right below and in Section 8.5 for establishing the frame property and preservation for some concrete (classes of) commands.

The command H is a compensator for the separation algebra $(States, *, [\![\mathsf{emp}]\!]_c)$.

We now can give generalised definitions of the frame and preservation properties.

Definition 8.9. Assume a compensator K.

(a) Command C has the *K-frame-property* iff

$$(dom(C) \times dom(K)) ; \rhd ; C \subseteq (C \times K) ; \rhd .$$

(b) Command C *K-preserves* test r iff

$$\lhd ; (C \times (r ; K)) ; \# \subseteq \top ; \lhd ; (I \times r) .$$

Again, preservation is equivalent to

$$\# ; (C \times (r ; K)) ; \# \subseteq \# ; (\top \times (\top ; r)) ; \# . \tag{20}$$

As an indication that this definition is reasonable, we show some properties about I, both as a command and a test.

Corollary 8.10. *I has the frame property and preserves every test. Moreover, every command preserves I.*

Proof. First, by the definitions, $(dom(I) \times dom(K)) ; \rhd ; I = (I \times dom(K)) ; \rhd \subseteq (I \times K) ; \rhd$, so that I satisfies the K-frame property.
Second,

$$\begin{array}{ll}
 & \# ; (I \times (r ; K)) ; \# \\
= & \quad \{\!\!\{ \text{ neutrality of } I \text{ and exchange (15) } \}\!\!\} \\
 & \# ; (I \times r) ; (I \times K) ; \# \\
= & \quad \{\!\!\{ \ \# \text{ and } (I \times r) \text{ are partial identities and hence are idempotent and commute } \}\!\!\} \\
 & \# ; (I \times r) ; \# ; (I \times K) ; \# \\
\Leftrightarrow & \quad \{\!\!\{ \text{ definition of compensators (Def. 8.8(b)) } \}\!\!\} \\
 & \# ; (I \times r) ; \# \\
\subseteq & \quad \{\!\!\{ \text{ isotony } \}\!\!\} \\
 & \# ; (\top \times (\top ; r)) ; \#
\end{array}$$

Third, that every command preserves I is immediate from (20) and neutrality of I. □

We derive some further consequences from the definitions.

Lemma 8.11.

(a) Suppose command C has the K-frame-property. Then for all tests p, r we have

$$p \bullet r \subseteq (p\,;dom(C)) \bullet (r\,;dom(K)) \quad\Rightarrow\quad (p \bullet r)\,; C \;\subseteq\; (p\,;C) \bullet (r\,;K).$$

(b) Suppose command C K-preserves test r. Then for all tests q we have

$$(C\,;q) \bullet (r\,;K) \;\subseteq\; \top\,;(q \bullet r)\,.$$

The proof can be found in the Appendix.

The generalised frame rule now reads as follows.

Theorem 8.12. *Assume a compensator K, a command C and a test r such that C has the K-frame-property and K-preserves r. Then the following inference rule is valid:*

$$\frac{p \bullet r \subseteq (p\,;dom(C)) \bullet (r\,;dom(K)) \qquad \{p\}\,C\,\{q\}}{\{p \bullet r\}\,C\,\{q \bullet r\}}$$

Proof. We calculate, assuming $p \bullet r \subseteq (p\,;dom(C)) \bullet (r\,;dom(K))$,

$$\begin{array}{ll}
& (p \bullet r)\,;C \\
\subseteq & \quad [\![\text{ by Lemma 8.11(a) }]\!] \\
& (p\,;C) \bullet (r\,;K) \\
\subseteq & \quad [\![\text{ by } \{p\}\,C\,\{q\}\text{, i.e., } p\,;C \;\subseteq\; C\,;q\text{, and isotony }]\!] \\
& (C\,;q) \bullet (r\,;K) \\
\subseteq & \quad [\![\text{ by Lemma 8.11(b) }]\!] \\
& \top\,;(q \bullet r)
\end{array}$$

By the second characterisation of the Hoare triple presented in Lemma 7.3 we have therefore established $\{p\bullet r\}\,C\,\{q\bullet r\}$.
□

For the special separation algebra $(States, *, [\![\mathsf{emp}]\!]_c)$ we obtain the $*$ frame rule. Notice that $dom(H) = I$.

Corollary 8.13. *Consider a command C and a test r such that C has the $*$-frame-property and $*$-preserves r. Then the following inference rule is valid:*

$$\frac{p * r \subseteq (p\,;dom(C)) * r \qquad \{p\}\,C\,\{q\}}{\{p * r\}\,C\,\{q * r\}}$$

The condition $p*r \subseteq (p\,;dom(C))*r$ is a relaxation of the implicit condition $p \subseteq dom(C)$ that is part of Reynolds's definition of validity of a Hoare triple (which differs from the standard one). For that reason also no notion of safety or safety-monotonicity of a command enters our treatment.

8.5. Closure Properties

Since more complex commands are built up from simpler ones using the $\cup$ and ; operators, we show that, subject to suitable conditions, our frame and preservation properties are closed under them.

Lemma 8.14. *Assume a compensator K.*

(a) The K-frame property is closed under union and pre-composition with a test. If C and D satisfy $cod(C) \subseteq dom(D)$ then the K-frame-property propagates from C and D to $C\,;D$.

(b) K-preservation of a test r is closed under union and pre-composition with a test. If

$$\#\,;((C\,;D) \times K)\,;\# \;=\; \#\,;(C \times K)\,;\#\,;(D \times K)\,;\# \tag{21}$$

then K-preservation of r propagates from C and D to $C\,;D$.

The proof can be found in the Appendix. The condition $\# ; ((C ; D) \times K) ; \# = \# ; (C \times K) ; \# ; (D \times K) ; \#$ means that the "local" intermediate state of the composition $C ; D$ induces a "global" intermediate state; it means a "modular" way of composition.

Let us briefly comment on the assumption $cod(C) \subseteq dom(D)$ in this lemma. This is needed since we want to argue about compositions in isolation. Contrarily, the related paper [50] talks about complete, non-blocking runs of programs; therefore for all compositions that make up such a run the above assumption about codomain and domain of successive commands is automatically always satisfied.

The second condition that enters closure of preservation under composition reflects the following observations in algebraic terms. If one wants to prove a frame law $\{p \bullet r\}\, C ; D\, \{s \bullet r\}$ for a composition $C ; D$ one may proceed in the Hoare logic style as follows. First one shows, with a suitable intermediate assertion q, that $\{p\}\, C\, \{q\}$ and $\{q\}\, D\, \{s\}$ hold. Then, assuming the frame property and r-preservation for C and D, one infers $\{p \bullet r\}\, C\, \{q \bullet r\}$ and $\{q \bullet r\}\, D\, \{s \bullet r\}$, from which the claim follows by the sequencing rule. The existence of q corresponds to the insertion of an intermediate $\#$ relation in (21).

Corollary 8.15. *The K-frame property and K-preservation propagate from commands C and D to* if p then C else D *for arbitrary test p.*

8.6. Another Separation Algebra

To present the advantage by abstracting from the concrete structure $(States, *, [\![\mathsf{emp}]\!]_c)$ in Section 8.4 we now give a second separation algebra with an adequate compensator that also satisfies the frame rule. Contrarily to the structure $(States, *, [\![\mathsf{emp}]\!]_c)$ it splits both store and heap into disjoint portions.

More precisely, we define a relation $\#_S$ on states by $(s_1, h_1) \;\#_S\; (s_2, h_2) \;\Leftrightarrow_{df}\; s_1 \cap s_2 = \emptyset \wedge h_1 \cap h_2 = \emptyset$. Now assume for states (s_1, h_1) and (s_2, h_2) that $(s_1, h_1) \;\#_S\; (s_2, h_2)$ holds. Then the operation $*_S$ is defined by $(s_1, h_1) *_S (s_2, h_2) =_{df} (s_1 \cup s_2, h_1 \cup h_2)$.

The following result is easily verified.

Corollary 8.16. *The structure $(States, *_S, \{(\emptyset, \emptyset)\})$ is a separation algebra and I is a compensator. Therefore the generalised frame rule is valid for it.*

9. Conclusion and Outlook

We have presented an algebraic treatment 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 [46] are simple consequences of standard residual laws. For intuitionistic, pure, precise and supported assertions we have given algebraic characterisations. Furthermore we have defined a class of assertions which are both intuitionistic and precise.

As a next step we embedded the command part of separation logic into a relational algebraic structure. There, we have derived algebraic characterisations of properties on which the frame rule relies. In particular, we have expressed the side condition that certain variables must not be changed by a command in a purely semantic way without appealing to the syntax. Moreover, we have formulated a more liberal version of the frame rule that rests on simpler side conditions and have proved it purely algebraically. Finally we have shown that, under mild conditions, the commands for which the frame rule is sound are closed under union and composition.

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 [11]. The term *in-place* means that there is no copying of whole structures, i.e., the reversal is done by simple pointer modifications.

Due to our relational embedding we can, as a next step, derive inference rules for if-statements and for the while-loop. This has been done for classical Hoare logic (see [37]); hence it should be straightforward to extend this into the setting of separation logical.

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 situations are given in [34, 19]. In future work, we will adapt these approaches to algebraic separation logic.

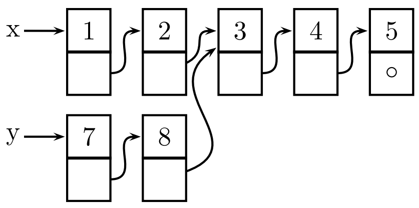

Figure 2: Two lists with shared cells.

Our algebraic approach to separation logic also paves the way for verifying properties with off-the-shelf theorem provers. Boolean semirings have proved to be reasonably well suitable for automated theorem provers [24]. Therefore the first-order part of our approach can easily be shown by automated theorem provers. If one needs the full power of quantales, the situation is a bit different: there are encodings for quantales for higher-order theorem provers [12]. However at the moment higher-order systems can only verify simply theorems fully automatically. Looking at the development of first-order provers in the past, we expect a rapid development in automated higher-order provers. 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 [41]. One central property mentioned in that paper is the rule of *disjoint concurrency* which reads

$$\frac{\{p_1\}\,C_1\,\{q_1\} \quad \{p_2\}\,C_2\,\{q_2\}}{\{p_1 * p_2\}\ C_1\,||\,C_2\ \{q_1 * q_2\}}\,,$$

assuming C_1 does not modify any free variables in p_2 and q_2 and conversely C_2 does not modify free variables of p_1 and q_1. Using the approach for the frame rule in this paper it should be possible to get an algebraic proof for this rule, too.

Acknowledgements: We are grateful to the anonymous referees for their careful scrutiny of the manuscript and their many helpful remarks. The material in Section 8 benefited greatly from discussions with C.A.R. Hoare and P. O'Hearn.

References

[1] R.-J. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction*. Graduate Texts in Computer Science. Springer, 1998.

[2] R. C. Backhouse and J. van der Woude. Demonic operators and monotype factors. *Mathematical Structures in Computer Science*, 3(4):417–433, 1993.

[3] B. Biering, L. Birkedal, and N. Torp-Smith. BI-hyperdoctrines, Higher-order Separation Logic, and Abstraction. *ACM Transactions on Programming Languages and Systems*, 29(5):24, 2007.

[4] G. Birkhoff. *Lattice Theory*, volume XXV of *Colloquium Publications*. Annals of Mathematics Studies, 3rd edition, 1967.

[5] T. Blyth and M. Janowitz. *Residuation theory*. Pergamon Press, 1972.

[6] A. Buch, T. Hillenbrand, and R. Fettig. Waldmeister: High Performance Equational Theorem Proving. In J. Calmet and C. Limongelli, editors, *Proceedings of the International Symposium on Design and Implementation of Symbolic Computation Systems*, number 1128 in Lecture Notes in Computer Science, pages 63–64. Springer, 1996.

[7] R. Burstall. Some techniques for proving correctness of programs which alter data structures. *Machine Intelligence*, 7:23–50, 1972.

[8] C. Calcagno, P. O'Hearn, and H. Yang. Local action and abstract separation logic. In *LICS '07: Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science*, pages 366–378. IEEE Press, 2007.

[9] E. Cohen. Using Kleene algebra to reason about concurrency control. Technical report, Telcordia, 1994.

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

[11] H.-H. Dang. Algebraic aspects of separation logic. Technical Report 2009-01, Institut für Informatik, 2009.

[12] H.-H. Dang and P. Höfner. Automated Higher-Order Reasoning about Quantales. In B. Konev, R. Schmidt, and S. Schulz, editors, *Workshop on Practical Aspects of Automated Reasoning (PAAR-2010)*, pages 40–49, 2010.

[13] H.-H. Dang, P. Höfner, and B. Möller. Algebraic Separation Logic. Technical Report 2010-06, Institute of Computer Science, University of Augsburg, 2010.

[14] J. Desharnais, N. Belkhiter, S. Sghaier, F. Tchier, A. Jaoua, A. Mili, and N. Zaguia. Embedding a demonic semilattice in a relational algebra. *Theoretical Computer Science*, 149(2):333–360, 1995.

[15] J. Desharnais, A. Mili, and T. Nguyen. Refinement and demonic semantics. In C. Brink, W. Kahl, and G. Schmidt, editors, *Relational Methods in Computer Science*, pages 166–183. Springer, 1997.

[16] J. Desharnais and B. Möller. Characterizing Determinacy in Kleene Algebras. *Information Sciences*, 139:253–273, 2001.

[17] J. Desharnais, B. Möller, and F. Tchier. Kleene under a modal demonic star. *Journal of Logic and Algebraic Programming*, 66(2):127–160, 2006.

[18] E. Dijkstra. *A discipline of programming*. Prentice Hall, 1976.

[19] 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 *Lecture Notes in Computer Science*, pages 99–111. Springer, 2004.

[20] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[21] C. A. R. Hoare, B. Möller, G. Struth, and I. Wehrman. Concurrent Kleene algebra. In M. Bravetti and G. Zavattaro, editors, *CONCUR 09 — Concurrency Theory*, volume 5710 of *Lecture Notes in Computer Science*, pages 399–414. Springer, 2009.

[22] C. A. R. Hoare, B. Möller, G. Struth, and I. Wehrman. Foundations of concurrent Kleene algebra. In R. Berghammer, A. Jaoua, and B. Möller, editors, *Relations and Kleene Algebra in Computer Science*, volume 5827 of *Lecture Notes in Computer Science*. Springer, 2009.

[23] P. Höfner. Database for automated proofs of Kleene algebra. `http://www.kleenealgebra.de` (accessed March 31, 2011).

[24] P. Höfner and G. Struth. Automated reasoning in Kleene algebra. In F. Pfennig, editor, *Automated Deduction*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 279–294. Springer, 2007.

[25] P. Höfner, G. Struth, and G. Sutcliffe. Automated verification of refinement laws. *Annals of Mathematics and Artificial Intelligence, Special Issue on First-order Theorem Proving*, pages 35–62, 2008.

[26] S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. *ACM SIGPLAN Notices*, 36:14–26, 2001.

[27] B. Jónsson and A. Tarski. Boolean algebras with operators, Part I. *American Journal of Mathematics*, 73, 1951.

[28] D. Kozen. On Hoare logic and Kleene algebra with tests. *ACM Transactions on Computational Logic*, 1(1):60–76, 2000.

[29] D. Kozen. On Hoare logic, Kleene algebra, and types. In P. Gärdenfors, J. Woleński, and K. Kijania-Placek, editors, *In the Scope of Logic, Methodology, and Philosophy of Science: Volume One of the 11th Int. Congress Logic, Methodology and Philosophy of Science, Cracow, August 1999*, volume 315 of *Studies in Epistemology, Logic, Methodology, and Philosophy of Science*, pages 119–133. Kluwer, 2002.

[30] D. Kozen and M.-C. Patron. Certification of compiler optimizations using Kleene algebra with tests. In J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, and P. J. Stuckey, editors, *Proc. 1st Int. Conf. Computational Logic (CL2000)*, volume 1861 of *Lecture Notes in Computer Science*, pages 568–582. Springer, 2000.

[31] R. Maddux. *Relation Algebras*, volume 150 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 2006.

[32] W. W. McCune. Prover9 and Mace4. <`http://www.cs.unm.edu/~mccune/prover9`>. (accessed March 31, 2011).

[33] B. Möller. Towards pointer algebra. *Science of Computer Programming*, 21(1):57–90, 1993.

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

[35] B. Möller. Kleene getting lazy. *Science of Computer Programming*, 65:195–214, 2007.

[36] B. Möller, P. Höfner, and G. Struth. Quantales and temporal logics. In M. Johnson and V. Vene, editors, *Algebraic Methodology and Software Technology, Proc. AMAST 2006*, volume 4019 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2006.

[37] B. Möller and G. Struth. Algebras of modal operators and partial correctness. *Theoretical Computer Science*, 351(2):221–239, 2006.

[38] B. Möller and G. Struth. WP is WLP. In W. MacCaull, M. Winter, and I. Düntsch, editors, *Relational Methods in Computer Science*, volume 3929 of *Lecture Notes in Computer Science*, pages 200–211. Springer, 2006.

[39] C. Mulvey. &. *Rendiconti del Circolo Matematico di Palermo*, 12(2):99–104, 1986.

[40] T. T. Nguyen. A relational model of nondeterministic programs. *International J. Foundations Comp. Sci.*, 2:101–131, 1991.

[41] P. O'Hearn. Resources, concurrency, and local reasoning. *Theoretical Computer Science*, 375:271–307, 2007.

[42] P. 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 *Lecture Notes in Computer Science*, pages 1–19. Springer, 2001.

[43] P. O'Hearn, J. C. Reynolds, and H. Yang. Separation and information hiding. *ACM Trans. Program. Lang. Syst.*, 31(3):1–50, 2009.

[44] V. Preoteasa. Frame rule for mutually recursive procedures manipulating pointers. *Theoretical Computer Science*, 410(42):4216–4233, 2009.

[45] D. J. Pym, P. O'Hearn, and H. Yang. Possible worlds and resources: the semantics of BI. *Theoretical Computer Science*, 315(1):257–305, 2004. Mathematical Foundations of Programming Semantics.

[46] J. C. Reynolds. An introduction to separation logic. In M. e. a. Broy, editor, *In Engineering Methods and Tools for Software Safety and Security*, pages 285–310. IOS Press, 2009.

[47] K. Rosenthal. Quantales and their applications. *Pitman Research Notes in Mathematics Series*, 234, 1990.

[48] G. Schmidt and T. Ströhlein. *Relations and Graphs: Discrete Mathematics for Computer Scientists*. Springer, 1993.

[49] V. Vafeiadis and M. Parkinson. A marriage of rely/guarantee and separation logic. In *In 18th Int'l Conference on Concurrency Theory (CONCUR'07)*, volume 4703 of *Lecture Notes in Computer Science*, pages 256–271. Springer, 2007.

[50] H. Yang and P. O'Hearn. A semantic basis for local reasoning. In M. Nielsen and U. Engberg, editors, *Foundations of Software Science and Computation Structures, Proc. FOSSACS 2002*, volume 2303 of *Lecture Notes in Computer Science*, pages 402–416. Springer, 2002.

A. Deferred Proofs

Proof of Lemma 5.8. By Lemma 5.4 we have $[\![p \mathbin{-\!\!*} q]\!] = [\![q]\!]/[\![p]\!]$. Now it is easy to see that

$$
\begin{array}{ll}
 & s,h \models p \mathbin{-\!\circledast} q \\
\Leftrightarrow & \quad \{\!\!\{ \text{ definition of } \mathbin{-\!\circledast} \}\!\!\} \\
 & s,h \models \neg(q \mathbin{-\!\!*}(\neg p)) \\
\Leftrightarrow & \quad \{\!\!\{ \text{ definition of } \mathbin{-\!\!*} \}\!\!\} \\
 & \neg(\forall h' : ((dom(h') \cap dom(h) = \emptyset,\ s,h' \models q) \Rightarrow s,h' \cup h \models \neg p))
\end{array}
$$

$\Leftrightarrow$ $\quad$ {[logic: $\neg$ over $\forall$]}

$\quad \exists h' : \neg((dom(h') \cap dom(h) = \emptyset,\ s, h' \models q) \Rightarrow s, h' \cup h \models \neg p)$

$\Leftrightarrow$ $\quad$ {[logic: $\neg(A \Rightarrow B) \Leftrightarrow (A \wedge \neg B)$]}

$\quad \exists h' : dom(h') \cap dom(h) = \emptyset,\ s, h' \models q,\ s, h' \cup h \not\models \neg p$

$\Leftrightarrow$ $\quad$ {[logic]}

$\quad \exists h' : dom(h') \cap dom(h) = \emptyset,\ s, h' \models q,\ s, h' \cup h \models p$

$\Leftrightarrow$ $\quad$ {[setting for ($\Rightarrow$) $\hat{h} =_{df} h' \cup h$ and for ($\Leftarrow$) $h' =_{df} \hat{h} - h$]}

$\quad \exists \hat{h} : h \subseteq \hat{h},\ s, \hat{h} - h \models q,\ s, \hat{h} \models p$

$\square$

Proof of Lemma 6.9. ($\Leftarrow$): Using Equation (Ded), isotony and the assumption, we get

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

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

$$a \sqcap (b \cdot c) \;=\; a \sqcap a \sqcap (b \cdot c) \leq a \sqcap ((a \sqcap b) \cdot c) \leq (a \sqcap b) \cdot (a \sqcap c)\ .$$

($\Rightarrow$): From (8) we obtain $a \sqcap (\overline{a} \cdot \top) \leq (a \sqcap \overline{a}) \cdot (a \sqcap \top) = 0 \cdot a = 0$ and hence, by shunting (shu) and the exchange law (exc), $a\lfloor\top \leq a$.

Proof of Lemma 6.10.

(a) The claim follows immediately from Lemma 6.9.

(b) We first show that $a = (a \sqcap 1) \cdot \top$ follows from Inequations (7) and (8). By neutrality of $\top$ for $\sqcap$, neutrality of 1 for $\cdot$, meet-distributivity (8) and isotony, we get

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

The converse inequation follows by isotony and Inequation (7):

$$(a \sqcap 1) \cdot \top \leq a \cdot \top \leq a\ .$$

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

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

For the second inequation, we note that in a Boolean quantale the law $\overline{t \cdot \top} = (\overline{t} \sqcap 1) \cdot \top$ holds for all subidentities t ($t \leq 1$) (e.g. [16]). From this we get

$$\overline{a} \cdot \top \;=\; \overline{(a \sqcap 1) \cdot \top} \cdot \top \;=\; (\overline{a} \sqcap 1) \cdot \top \cdot \top \;=\; (\overline{a} \sqcap 1) \cdot \top \;=\; \overline{(a \sqcap 1) \cdot \top} \;=\; \overline{a}\ .$$

(c) $(a \sqcap b) \cdot c = a \sqcap b \cdot c$.
We show the equivalence of the equation with Definition 6.6. First we prove the $\Rightarrow$-direction and split the proof showing each inequation separately starting with $\geq$:

$$a \sqcap b \cdot c \leq (a \sqcap b) \cdot (a \sqcap c) \leq (a \sqcap b) \cdot c\ .$$

This holds by Equation 8 and isotony. To prove the $\leq$-direction we know $(a \sqcap b) \cdot c \leq a \cdot c \leq a \cdot \top \leq a$ which follows from isotony and Equation 7. Now using $(a \sqcap b) \cdot c \leq b \cdot c$ we can immediately conclude $(a \sqcap b) \cdot c \leq a \sqcap b \cdot c$.

Next we give a proof for the $\Leftarrow$-direction and assume $(a \sqcap b) \cdot c = a \sqcap b \cdot c$ holds. Equation 7 follows by $a \cdot \top = (a \sqcap a) \cdot \top = a \sqcap a \cdot \top \leq a$. Furthermore we calculate

$$a \sqcap (b \cdot c) = a \sqcap a \sqcap (b \cdot c) = a \sqcap ((a \sqcap b) \cdot c) = a \sqcap (a \cdot (a \sqcap b)) = (a \sqcap b) \cdot (a \sqcap c)\ ,$$

which holds by using idempotence of $\sqcap$, the assumption, commutativity of $\cdot$ and again the assumption. $\square$

Proof of Lemma 6.25.

(a) Since $[\![s,h]\!]$ is a singleton set, this is obvious.

(b) By definition of $*$, Part (1), by the assumption $h \subseteq h'$ and Lemma 6.24, and set theory:

$$\begin{aligned}
& s,h' \models p * [\![s,h'-h]\!] \\
\Leftrightarrow\ & \exists\, \hat{h} : \hat{h} \subseteq h' \wedge s,\hat{h} \models p \wedge s,h'-\hat{h} \models [\![s,h'-h]\!] \\
\Leftrightarrow\ & \exists\, \hat{h} : \hat{h} \subseteq h' \wedge s,\hat{h} \models p \wedge h'-\hat{h} = h'-h \\
\Leftrightarrow\ & \exists\, \hat{h} : \hat{h} \subseteq h' \wedge s,\hat{h} \models p \wedge \hat{h} = h \\
\Leftrightarrow\ & s,h \models p
\end{aligned}$$

(c) By definition of $*$, $s,h'-\hat{h} \models \mathsf{true}$ is true, Part (1), and set theory:

$$\begin{aligned}
& s,h' \models [\![s,h]\!] * \mathsf{true} \\
\Leftrightarrow\ & \exists\, \hat{h} : \hat{h} \subseteq h' \wedge s,\hat{h} \models [\![s,h]\!] \wedge s,h'-\hat{h} \models \mathsf{true} \\
\Leftrightarrow\ & \exists\, \hat{h} : \hat{h} \subseteq h' \wedge s,\hat{h} \models [\![s,h]\!] \\
\Leftrightarrow\ & \exists\, \hat{h} : \hat{h} \subseteq h' \wedge \hat{h} = h \\
\Leftrightarrow\ & h \subseteq h'
\end{aligned}$$

□

Proof of Lemma 7.3. We first show the second equivalence.

$$\begin{aligned}
& \top ; p ; C \subseteq \top ; q \\
\Rightarrow\ & \quad \{\!\!\{\ \text{by } p \subseteq \top ; p \text{ and isotony}\ \}\!\!\} \\
& p ; C \subseteq \top ; q \\
\Rightarrow\ & \quad \{\!\!\{\ \text{composing } \top \text{ to both sides}\ \}\!\!\} \\
& \top ; p ; C \subseteq \top ; \top ; q \\
\Rightarrow\ & \quad \{\!\!\{\ \text{by } \top ; \top \subseteq \top\ \}\!\!\} \\
& \top ; p ; C \subseteq \top ; q
\end{aligned}$$

Now we tackle the first equivalence.
($\Rightarrow$) By isotony,

$$p ; C \subseteq C ; q \Rightarrow \top ; p ; C \subseteq \top ; C ; q \subseteq \top ; \top ; q \subseteq \top ; q\,.$$

($\Leftarrow$) First, since $p \subseteq I$, we have $p ; C \subseteq C$. Second, by neutrality isotony and the assumption,

$$p ; C = I ; p ; C \subseteq \top ; p ; C \subseteq \top ; q\,.$$

Therefore, $p ; C \subseteq C \cap \top ; q$. By a standard result on test semirings (e.g. [35]) the right-hand side is equal to $C ; q$.

Proof of Lemma 8.11.

(a) We calculate, assuming $p \bullet r \subseteq (p ; dom(C)) \bullet (r ; dom(K))$,

$$\begin{aligned}
& (p \bullet r) ; C \\
\subseteq\ & \quad \{\!\!\{\ \text{assumption}\ \}\!\!\} \\
& ((p ; dom(C)) \bullet (r ; dom(K))) ; C \\
=\ & \quad \{\!\!\{\ \text{definition of } \bullet \text{ on relations (Def. 8.2(d))}\ \}\!\!\} \\
& \lhd ; ((p ; dom(C)) \times (r ; dom(K))) ; \rhd ; C
\end{aligned}$$

$$\begin{aligned}
= \quad & \{\![\ \text{exchange (15)}\ ]\!\} \\
& \lhd \,;\, (p \times r) \,;\, (dom(C) \times dom(K)) \,;\, \rhd \,;\, C \\
\subseteq \quad & \{\![\ \text{since } C \text{ has the } K\text{-frame-property}\ ]\!\} \\
& \lhd \,;\, (p \times r) \,;\, (C \times K) \,;\, \rhd \\
\subseteq \quad & \{\![\ \text{exchange (15)}\ ]\!\} \\
& \lhd \,;\, (p \,;\, C) \times (r \,;\, K) \,;\, \rhd \\
= \quad & \{\![\ \text{definition of } \bullet \text{ on relations}\ ]\!\} \\
& (p \,;\, C) \bullet (r \,;\, K)
\end{aligned}$$

(b)

$$\begin{aligned}
& (C \,;\, q) \bullet (r \,;\, K) \\
= \quad & \{\![\ \text{definition of } \bullet \text{ on relations}\ ]\!\} \\
& \lhd \,;\, (C \,;\, q) \times (r \,;\, K) \,;\, \rhd \\
= \quad & \{\![\ \text{neutrality of } I \text{ w.r.t. composition and exchange (15)}\ ]\!\} \\
& \lhd \,;\, (C \times (r \,;\, K)) \,;\, (q \times I) \,;\, \rhd \\
= \quad & \{\![\ \text{Equation (16)}\ ]\!\} \\
& \lhd \,;\, (C \times (r \,;\, K)) \,;\, (q \times I) \,;\, \# \,;\, \rhd \\
= \quad & \{\![\ (q \times I) \text{ and } \# \text{ are partial identities and hence commute}\ ]\!\} \\
& \lhd \,;\, (C \times (r \,;\, K)) \,;\, \# \,;\, (q \times I) \,;\, \rhd \\
\subseteq \quad & \{\![\ C\ K\text{-preserves } r\ ]\!\} \\
& \top \,;\, \lhd \,;\, (I \times r) \,;\, (q \times I) \,;\, \rhd \\
= \quad & \{\![\ \text{exchange (15) and neutrality of } I\ ]\!\} \\
& \top \,;\, \lhd \,;\, (q \times r) \,;\, \rhd \\
= \quad & \{\![\ \text{definition of } \bullet \text{ on relations}\ ]\!\} \\
& \top \,;\, (q \bullet r)\,.
\end{aligned}$$

Proof of Lemma 8.14. Closure under union is straightforward from distributivity of ; and $\times$ over $\cup$. We now show closure under (pre-)composition.

(a) Assume that C has the K-frame property and that p is a test.

$$\begin{aligned}
& (dom(p \,;\, C) \times dom(K)) \,;\, \rhd \,;\, p \,;\, C \\
= \quad & \{\![\ \text{neutrality of } I,\ dom(p \,;\, C) = p \,;\, dom(C) \text{ and exchange (15)}\ ]\!\} \\
& (p \times I) \,;\, (dom(C) \times dom(K)) \,;\, \rhd \,;\, p \,;\, C \\
\subseteq \quad & \{\![\ \text{isotony}\ ]\!\} \\
& (p \times I) \,;\, (dom(C) \times dom(K)) \,;\, \rhd \,;\, C \\
\subseteq \quad & \{\![\ C \text{ has the frame property}\ ]\!\} \\
& (p \times I) \,;\, (C \times K) \,;\, \rhd \\
= \quad & \{\![\ \text{exchange (15)}\ ]\!\} \\
& ((p \,;\, C) \times K) \,;\, \rhd
\end{aligned}$$

Assume now that C and D have the K-frame property and satisfy $cod(C) \subseteq dom(D)$. Then

$$\begin{aligned}
& (dom(C \,;\, D) \times dom(K)) \,;\, \rhd \,;\, C \,;\, D \\
\subseteq \quad & \{\![\ \text{definition of domain}\ ]\!\} \\
& (dom(C) \times dom(K)) \,;\, \rhd \,;\, C \,;\, D \\
\subseteq \quad & \{\![\ C \text{ has the } K\text{-frame property}\ ]\!\} \\
& (C \times K) \,;\, \rhd \,;\, D
\end{aligned}$$

$=$ ⦃ $R = R\,;cod(R)$ for arbitrary R and exchange (15) ⦄

$(C \times K)\,;(cod(C) \times cod(K))\,;\rhd\,;D$

$\subseteq$ ⦃ assumption and definition of compensator ⦄

$(C \times K)\,;(dom(D) \times dom(K))\,;\rhd\,;D$

$\subseteq$ ⦃ D has the K-frame property ⦄

$(C \times K)\,;(D \times K)\,;\rhd$

$=$ ⦃ exchange (15) ⦄

$((C\,;D) \times (K\,;K))\,;\rhd$

$=$ ⦃ definition of compensators ⦄

$((C\,;D) \times K)\,;\rhd\,.$

(b) Assume that C K-preserves r and that p is a test. Then

$$\lhd\,;((p\,;C) \times (r\,;K))\,;\# \subseteq \lhd\,;(C \times (r\,;K))\,;\# \subseteq \top\,;\lhd\,;(I \times r)\,.$$

Assume now that C and D K-preserve r and $\#\,;((C\,;D) \times K)\,;\# \;=\; \#\,;(C \times K)\,;\#\,;(D \times K)\,;\#$. Then

$\lhd\,;((C\,;D) \times (r\,;K))\,;\#$

$=$ ⦃ neutrality of I and exchange (15) ⦄

$\lhd\,;(I \times r)\,;((C\,;D) \times K)\,;\#$

$=$ ⦃ by Equation (16) ⦄

$\lhd\,;\#\,;(I \times r)\,;((C\,;D) \times K)\,;\#$

$=$ ⦃ $\#$ and $I \times r$ are partial identities and hence are idempotent and commute ⦄

$\lhd\,;\#\,;(I \times r)\,;\#\,;((C\,;D) \times K)\,;\#$

$=$ ⦃ by Equation (16) ⦄

$\lhd\,;(I \times r)\,;\#\,;((C\,;D) \times K)\,;\#$

$=$ ⦃ by the assumption ⦄

$\lhd\,;(I \times r)\,;\#\,;(C \times K)\,;\#\,;(D \times K)\,;\#$

$\subseteq$ ⦃ $\#$ is a subidentity ⦄

$\lhd\,;(I \times r)\,;(C \times K)\,;\#\,;(D \times K)\,;\#$

$=$ ⦃ exchange (16) and neutrality of I ⦄

$\lhd\,;(C \times (r\,;K))\,;\#\,;(D \times K)\,;\#$

$\subseteq$ ⦃ C K-preserves r ⦄

$\top\,;\lhd\,;(I \times r)\,;(D \times K)\,;\#$

$=$ ⦃ exchange (16) and neutrality of I ⦄

$\top\,;\lhd\,;(D \times (r\,;K))\,;\#$

$\subseteq$ ⦃ D K-preserves r ⦄

$\top\,;\top\,;\lhd\,;(I \times r)$

$\subseteq$ ⦃ $\top$ largest element ⦄

$\top\,;\lhd\,;(I \times r)$