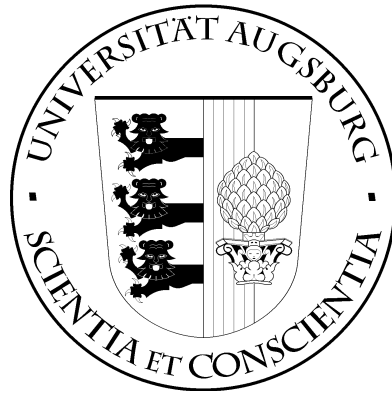
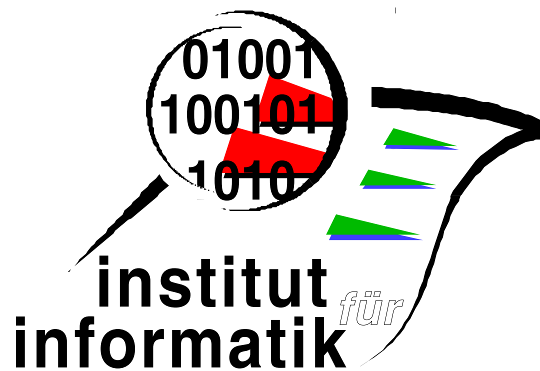
Universität Augsburg



Algebraic Separation Logic

H.-H. Dang P. Höfner B. Möller

Report 2010-06 July 2010



Institut für Informatik
D-86135 Augsburg

Copyright © H.-H. Dang P. Höfner B. Möller
Institut für Informatik
Universität Augsburg
D–86135 Augsburg, Germany
http://www.Informatik.Uni-Augsburg.DE
— all rights reserved —

Algebraic Separation Logic

H.-H Dang[1], P. Höfner[1], B. Möller[1]

Institut für Informatik, Universität Augsburg, D-86159 Augsburg, Germany

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 the simple programming language associated with separation logic. On this basis we prove the frame rule in an abstract and concise way. We also propose a more general version of separating conjunction which leads to a frame rule that is easier to prove. In particular, we show how to algebraically formulate the requirement that a command does not change certain variables; this is also expressed more conveniently using the generalised separating conjunction. 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 a more abstract level.

Keywords: separation logic, algebra

1. Introduction

Two prominent formal methods for reasoning about the correctness of programs are Hoare logic [20] and the wp-calculus of Dijkstra [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 [40, 42]. 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 [8] which was then explicitly described in an intutionistic 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. This language has been extended to concurrent programs that work on shared mutable data structures [39].

In this paper we present an abstract algebraic approach to separation logic. Our approach is based on quantales [37], also called standard Kleene algebras [11]. They are a special case of the fundamental algebraic structure of idempotent semirings, which have been used in various applications ranging from concurrency control [10, 21, 22] to program analysis and semantics. In particular, there are already algebraic characterisations for Hoare logic [27, 35] and the wp-calculus of Dijkstra [36], 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

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

re-use of a large body of existing theory. Last but not least, since it is largely formulated in pure first-order logic, the 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 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 and a more liberal version of it that also enables a simpler treatment of variable 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 [31] 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

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

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

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

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 if the state (s, h) satisfies the assertion p; an assertion p is called *valid* iff p holds in every state and, finally, p is *satisfiable* if there exists a state (s, h) which satisfies p. The semantics is defined inductively as follows (e.g. [42]).

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

Here, b is a *bexp*-expression, p, q are assertions and e_1, e_2 are *exp*-expressions. The first four clauses do not 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 $\circledast$ for splitting stores and heaps simultaneously.

The *separating implication* $p \mathrel{-\!\!*} q$ guarantees, that if a heap h is extended with a heap h' satisfying p, the combined heap $h \cup h'$ satisfies q (cf. Figure 1). An attempt to combine two non-disjoint heaps is interpreted as an error case and therefore the assertion is not satisfied for the corresponding states.

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

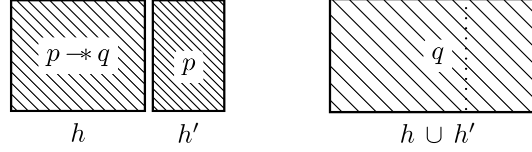


Figure 1: Separating implication [1]

4. Quantales

To present our algebraic semantics of separation logic in the next section, we now prepare the algebraic background. The main algebraic structure for assertions and commands which we discuss in the next sections is a *quantale*.

A *quantale* [37, 43] 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* [11] and a special case of the notion of an idempotent semiring.

A quantale is called *Boolean* if its underlying lattice is distributive and complemented, whence a Boolean algebra. An important Boolean quantale is REL, the algebra of binary relations over a set under set inclusion $\subseteq$, relation composition ; and set complement $\overline{}$.

A useful property we will need is shunting:

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

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

A quantale is called *commutative* if $a \cdot b = b \cdot a$ for all $a, b \in S$. In any quantale, the *right residual* $a\backslash b$ [4] exists and is characterised by the Galois connection

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

Symmetrically, the *left residual* b/a can be defined. However, if the underlying quantale is commutative then both residuals coincide, i.e., $a\backslash b = b/a$. In REL, one has $R_1\backslash R_2 = \overline{\overline{R_1}^{\smile} ; \overline{R_2}}$ and $R_1/R_2 = \overline{\overline{R_1} ; R_2^{\smile}}$, where $^{\smile}$ denotes relational converse. 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 [26]

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

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

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 [34] 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 arbitrary elements a.

Algebraic structures in general are suitable for automated theorem proving. However, quantales are not easy to encode and perform rather badly [13]. 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 [30] and Waldmeister [7]. 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. Most of the input files can be found at a web page [23]. However, to demonstrate the simplicity of our algebra, we will 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 (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 [12]. 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 $\cup\!\!\!\cdot$ 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

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

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*

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

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 [12].

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

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

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 $\cup$, we have

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

and therefore, for arbitrary set R of states,

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

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

Now all laws of [42] about $\mathrel{-\!\!*}$ follow from the standard theory of residuals (e.g. [5]). Many of these laws are proved algebraically in [12]. 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). For example, the two main properties of separating implication, namely the currying and decurrying rules, are nothing but the transcriptions of the defining Galois connection for right residuals.

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

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

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

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

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

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

Proof. By definition of residuals we immediately get

$$q \cdot (q \backslash p) \leq p \Leftrightarrow q \backslash p \leq q \backslash p \Leftrightarrow \mathsf{true}\,.$$

□

By the definitions of Section 4 we now give a concrete definition of the right detachment operator in the logic itself. This operation is also called *septraction* in the literature (see e.g. [45]).

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 can be found in Appendix B.

In Appendix A, 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 5.

Name in SL	Symbol in SL	Name in Quantales	Symbol in AS	Symbol in a quantale
disjunction	$\vee$	addition/join	$\cup$	$+$
conjunction	$\wedge$	meet	$\cap$	$\sqcap$
negation	$\neg$	complement	$\overline{}$	$\overline{}$
implication	$\Rightarrow$	natural order	$\subseteq$	$\leq$
separating conjunction	$*$	multiplication	$\cup\!\!\!\cdot$	$\cdot$
separating implication	$\mathrel{-\!\!*}$	residual	$/$	$/$
septraction	$\mathrel{-\!\circledast}$	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 [42]. We will 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 [42] 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 [42], an assertion p is *intuitionistic* iff

$$\forall\, s \in \mathit{Stores}, \forall\, h, h' \in \mathit{Heaps} : (h \subseteq h' \;\wedge\; s, h \models p) \;\Rightarrow\; s, h' \models p\,. \tag{3}$$

This means in detail, 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*

$$[\![\, p \,]\!] \cup\!\!\!\cdot\; [\![\, \mathsf{true} \,]\!] \subseteq [\![\, p \,]\!]\,.$$

Proof.

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

□

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

$$a \cdot \top \leq a\,. \tag{4}$$

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. [44, 29]). 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 (a) we calculate $(a \sqcap b) \cdot \top \leq a \cdot \top \leq a$. For a proof of (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. □

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

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

The last step follows from Example 5.2. □

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.,

$$p \text{ is pure} \quad \Leftrightarrow_{df} \quad (\forall\, s \in Stores : \forall\, h, h' \in Heaps : s, h \models p \Leftrightarrow s, h' \models p) \, . \tag{5}$$

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

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

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

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

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

$$a \cdot \top \le a \, , \tag{6}$$

$$a \sqcap (b \cdot c) \le (a \sqcap b) \cdot (a \sqcap c) \, . \tag{7}$$

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* (7) *is equivalent to* $a \lfloor \top \le a$*, where* $a \lfloor \top$ *forms the downward closure of* a*.*

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

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

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

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

($\Rightarrow$): From (7) we obtain $a \sqcap (\overline{a} \cdot \top) \le (a \sqcap \overline{a}) \cdot (a \sqcap \top) = 0 \cdot a = 0$ and hence, by shunting (shu) and the exchange law (exc), $a \lfloor \top \le a$. □

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

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

(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 B. Part (a) also holds for non-commutative quantales; Part (b) characterises pure elements as a 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

$$\forall\, s \in Stores,\ \forall\, h, h' \in Heaps : (s, h \models p \Leftrightarrow s, h' \models p)\,, \tag{8}$$

$$\forall\, s \in Stores,\ \forall\, h \in Heaps : (s, h \models p \Leftrightarrow s, h \models (p \wedge \mathsf{emp}) * \mathsf{true})\,. \tag{9}$$

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 $h \in Heaps$

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

Instantiating Equation (8) and using this result we obtain

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

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 (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 (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 Appendix A. 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

The next class of assertions we focus now are *precise* assertions. They play a main rôle in characterising best local-actions in [9]. Intuitively they point out precise portions of heaps that satisfy the predicate. An assertion p is called *precise* if and only if for all states (s, h), there is at most one subheap h' of h for which $(s, h') \models p$, 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 [41], this definition is equivalent to distributivity of $*$ over $\wedge$.

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

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

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 $b, c \in S$

$$(a \cdot b) \sqcap (a \cdot c) \leq a \cdot (b \sqcap c) . \tag{10}$$

Equation (10) can be strengthened to an equation since $a \cdot (b \sqcap c) \leq (a \cdot b) \sqcap (a \cdot c)$ 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

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

□

Lemma 6.16. *If a is precise and $a' \leq a$ 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 Appendix A.

6.4. Fully-allocated assertions

After giving algebraic characterisations for precise and intuitionistic elements we turn to the question if 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 [42]. 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. We call this class of assertions *fully allocated*. According to their name the heap storage is fully allocated and allows no further allocation of non-empty heap cells which can be characterised as follows

$$p \text{ is fully allocated} \quad \Leftrightarrow_{df} \quad (\forall\, s, h : \ s, h \models p \Rightarrow dom(h) = Addresses) . \tag{11}$$

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

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

Proof.

$$
\begin{array}{ll}
& \forall\, s,h:\;\; s,h \models p \;\Rightarrow\; dom(h) = Addresses \\
\Leftrightarrow & \{\!\mid \text{order theory} \mid\!\} \\
& \forall\, s,h:\;\; (s,h \models p \;\Rightarrow\; (\forall\, h'.\; h \subseteq h' \;\Rightarrow\; h' \subseteq h)) \\
\Leftrightarrow & \{\!\mid \text{logic} \mid\!\} \\
& \forall\, s,h,h':\;\; (s,h \models p \;\Rightarrow\; (h \subseteq h' \;\Rightarrow\; h' \subseteq h)) \\
\Leftrightarrow & \{\!\mid \text{logic, set theory} \mid\!\} \\
& \forall\, s,h,h':\;\; (s,h \models p \;\Rightarrow\; \neg(h \subseteq h' \;\wedge\; h' - h \neq \emptyset)) \\
\Leftrightarrow & \{\!\mid \text{logic} \mid\!\} \\
& \forall\, s,h':\;\; \neg(\exists\, h:\;\; s,h \models p \;\wedge\; h \subseteq h' \;\wedge\; h' - h \neq \emptyset) \\
\Leftrightarrow & \{\!\mid \text{logic} \mid\!\} \\
& \forall\, s,h':\;\; \neg(\exists\, h,h'':\;\; s,h \models p \;\wedge\; h'' \neq \emptyset \;\wedge\; dom(h) \cap dom(h'') = \emptyset \;\wedge\; h' = h \cup h'') \\
\Leftrightarrow & \{\!\mid \text{definition of } *\text{, logic} \mid\!\} \\
& \forall\, s,h':\;\; s,h' \models p * \neg\mathsf{emp} \;\Rightarrow\; \mathsf{false}
\end{array}
$$

□

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

$$a \cdot \overline{1} \leq 0\,. \tag{12}$$

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 [42] for reasoning about directed acyclic graphs.

An assertion p is called *supported* iff

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

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. In more details, the decomposition parts have to be compatible, i.e., the stores must agree on their intersection.

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* $[\![\, r \,]\!]$,

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

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 B. 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,

$$s,h \models p * q \ \wedge \ p * r \ \Rightarrow \ s,h \models p * (q * \mathsf{true} \ \wedge \ r * \mathsf{true}) \ . \tag{13}$$

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

$$\begin{array}{ll} & s,h_1 \models p \ \wedge \ s,h_2 \models p \\ \Leftrightarrow & \{\ h_1 \subseteq h_1 \cup h_2 \text{ and } h_2 \subseteq h_1 \cup h_2 \text{ and Lemma 6.25(b) } \} \\ & s,h_1 \cup h_2 \models p * [\![s,(h_1 \cup h_2) - h_1]\!] \wedge \\ & s,h_1 \cup h_2 \models p * [\![s,(h_1 \cup h_2) - h_2]\!] \\ \Rightarrow & \{\ \text{Assumption (13)} \ \} \\ & s,h_1 \cup h_2 \models p * ([\![s,(h_1 \cup h_2) - h_1]\!] * \mathsf{true} \ \wedge \ [\![s,(h_1 \cup h_2) - h_2]\!] * \mathsf{true}) \\ \Leftrightarrow & \{\ \text{definition of } * \ \} \\ & \exists h' : h' \subseteq h_1 \cup h_2 \ \wedge \ s,h' \models p \wedge \\ & s,(h_1 \cup h_2) - h' \models [\![s,(h_1 \cup h_2) - h_1]\!] * \mathsf{true} \ \wedge \ [\![s,(h_1 \cup h_2) - h_2]\!] * \mathsf{true} \\ \Leftrightarrow & \{\ \text{definition of } \wedge \ \} \\ & \exists h' : h' \subseteq h_1 \cup h_2 \ \wedge \ s,h' \models p \wedge \\ & s,(h_1 \cup h_2) - h' \models [\![s,(h_1 \cup h_2) - h_1]\!] * \mathsf{true} \ \wedge \\ & s,(h_1 \cup h_2) - h' \models [\![s,(h_1 \cup h_2) - h_2]\!] * \mathsf{true} \\ \Leftrightarrow & \{\ \text{Lemma 6.25(c) (twice)} \ \} \\ & \exists h' : h' \subseteq h_1 \cup h_2 \ \wedge \ s,h' \models p \wedge \\ & (h_1 \cup h_2) - h_1 \subseteq (h_1 \cup h_2) - h' \wedge \\ & (h_1 \cup h_2) - h_2 \subseteq (h_1 \cup h_2) - h' \\ \Leftrightarrow & \{\ \text{Lemma 6.24} \ \} \\ & \exists h' : h' \subseteq h_1 \cup h_2 \ \wedge \ s,h' \models p \ \wedge \ h' \subseteq h_1 \ \wedge \ h' \subseteq h_2 \\ \Leftrightarrow & \{\ \text{set theory} \ \} \\ & \exists h' : s,h' \models p \ \wedge \ h' \subseteq h_1 \ \wedge \ h' \subseteq h_2 \end{array}$$

□

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 Appendix A 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. $\square$

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,

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

$\square$

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 ;, are available for commands. Moreover, the structure $(\mathit{Cmds}, \subseteq, ;, I)$, where I is the identity relation, forms a quantale.

Some properties of commands can be described more conveniently using relations that involve pairs of states.

Definition 7.2.

(a) Two states $\sigma_1 = (s_1, h_1)$ and $\sigma_2 = (s_2, h_2)$ are $*$-*combinable* if $s_1 = s_2$ and $dom(h_1) \cap dom(h_2) = \emptyset$. In this case we set $\sigma_1 * \sigma_2 =_{df} (s_1, h_1 \cup h_2)$.

(b) The *split* relation $\triangleleft\!\!\!* \subseteq \mathit{States} \times (\mathit{States} \times \mathit{States})$ w.r.t. * is given by

$$\sigma \triangleleft\!\!\!* (\sigma_1, \sigma_2) \Leftrightarrow_{df} \sigma_1, \sigma_2 \text{ $*$-combinable and } \sigma = \sigma_1 * \sigma_2.$$

(c) The *join* relation $*\!\!\!\triangleright$ is the converse of split, i.e.,

$$(\sigma_1, \sigma_2) *\!\!\!\triangleright \sigma \Leftrightarrow_{df} \sigma_1, \sigma_2 \text{ $*$-combinable and } \sigma = \sigma_1 * \sigma_2.$$

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

(d) The *Cartesian product* $C_1 \times C_2 \subseteq (\mathit{States} \times \mathit{States}) \times (\mathit{States} \times \mathit{States})$ of two commands C_1, C_2 is defined by

$$(\sigma_1, \sigma_2)\,(C_1 \times C_2)\,(\tau_1, \tau_2) \Leftrightarrow_{df} \sigma_1\, C_1\, \tau_1 \ \wedge\ \sigma_2\, C_2\, \tau_2\ .$$

(e) The $*$ *composition* $C_1 * C_2 \subseteq \mathit{States} \times \mathit{States}$ of two commands C_1, C_2 is defined by

$$C_1 * C_2 =_{df} \triangleleft\!\!\!* \ ; \ (C_1 \times C_2) \ ; *\!\!\!\triangleright .$$

It is well known that $\times$ and ; satisfy the exchange property

$$(R_1 \times R_2)\,;(S_1 \times S_2) \;=\; (R_1\,; S_1) \times (R_2\,; S_2)\,. \tag{14}$$

In particular if R_1, R_2 and S_1 are subidentities with $R_1 \subseteq S_1$ then

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

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 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, 38] 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 if and only if 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.

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.

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 $P = dom(\widehat{P})$. Because of this isomorphism, for a command C we simply write $dom(C)$ instead of $\widehat{dom(C)}$.

Definition 7.3. Employing tests, we can give the following equivalent definitions of the meaning of Hoare triples (e.g. [28]):

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

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

$$\widehat{P} \mathbin{\dot\cup} \widehat{Q} =_{df} \widehat{P \mathbin{\dot\cup} Q}\,.$$

In contrast to the above general encoding of Hoare triples the interpretation of Reynolds is more restrictive. He requires that all states that satisfy the precondition actually can lead to transitions under the command C. This is reflected by the following definition.

Definition 7.4. The *strong Hoare triple* is given by

$$\pmb{\{}p\pmb{\}}\,C\,\pmb{\{}q\pmb{\}} \;\Leftrightarrow_{df}\; p \subseteq dom(C) \,\wedge\, \{p\}\,C\,\{q\} \;\Leftrightarrow\; p \subseteq dom(C) \,\wedge\, p\,; C \subseteq C\,; q\,.$$

Note that $dom(C)$, according to the above convention, stands for the corresponding partial identity relation.

7.3. Syntax and Semantics of a Simple Programming Language

We now introduce the program constructs associated with separation logic [42]. Syntactically, they are given by

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

We skip explanations for the well-known first three lines. 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 process is chosen non-deterministically. 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., $e^S \in dom(h)$ 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 execution the disposed cell is not valid anymore, i.e., dereferencing that cell would cause a fault in the program execution.

We inductively assign to every program P formed according to the above grammar a command $[\![P]\!]_c \in Cmds$ in the sense of the previous section. In particular, $[\![\mathsf{skip}]\!]_c =_{df} I$. For all programs we assume that the free variables of all expressions are within the domains of the stores involved.

To abbreviate the next definition, we use a convention similar to that of the refinement calculus (e.g. [1]). We characterise relations by predicates linking the input states (s, h) and output states (s', h'). If P is such a predicate then $R \mathrel{\widehat{=}} P$ abbreviates the clause $(s, h)\, R\, (s', h') \Leftrightarrow_{df} P$.

$$
\begin{array}{rcl}
[\![[e_1] := e_2]\!]_c & \widehat{=} & s' = s \wedge h' = (e_1^S, e_2^S) \mid h \;\wedge\; e_1^S \in dom(h)\,, \\
[\![v := [e]]\!]_c & \widehat{=} & s' = (v, h(e^S)) \mid s \;\wedge\; h' = h\,, \\
[\![\,\mathsf{dispose}\; e]\!]_c & \widehat{=} & s' = s \wedge e^S \in dom(h) \wedge h' = h - \{(e^S, h(e^S))\}\,, \\
[\![v := \mathsf{cons}\,(e_1, ..., e_n)]\!]_c & \widehat{=} & \exists\, a \in Addresses : s' = (v, a) \mid s \;\wedge \\
& & \quad a, \ldots, a + n - 1 \notin dom(h) \wedge \\
& & \quad h' = \{(a, e_1^S), \ldots, (a + n - 1, e_n^S)\} \mid h\,.
\end{array}
$$

The condition for $\mathsf{dispose}$ guarantees that the dereferenced heap cell is already allocated to avoid a fault in the program execution. Similarly, the requirement $a, \ldots, a + n - 1 \notin dom(h)$ expresses that all these addresses have to be unallocated. More details can be found in [12].

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

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

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 $[e_1] := e_2$ is 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 $p = \mathtt{emp}$. 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 [42, 12]). Further inference rules, like for disposal, overwriting and non-overwriting allocation as well as lookups are presented and explained in [42]; 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* [40] that describes the use of the separating conjunction in pre- and postconditions for 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.

This rule is not valid for arbitrary commands, though. First, one has to assume that if C can act in a starting state that satisfies p then it can act in the same state enlarged according to r. Second, C needs to preserve certain parts of the heap. Finally, no free variable of r may be modified by C.

8.1. Safety-Monotonicity and the Frame Property

Before stating these properties algebraically, we give logical formalisations of the first two.

Definition 8.1. Consider a command C.

(a) If a state σ is in $dom(C)$ then C is called *safe at* σ, since by our above assumption then C cannot lead to program errors when starting from σ.

(b) C is called $*$-*safety-monotonic* if, for every state (s, h) and heap h' with $dom(h') \cap dom(h) = \emptyset$,

$$C \text{ is safe at } (s, h) \;\Rightarrow\; C \text{ is safe at } (s, h \cup h')\;.$$

(c) C has the $*$-*frame-property* if, for all states $(s, h_0), (\hat{s}, \hat{h})$ and heaps h_1 with $dom(h_1) \cap dom(h_0) = \emptyset$,

$$\begin{array}{ll} & C \text{ is safe at } (s, h_0) \;\land\; (s, h_0 \cup h_1)\, C\, (\hat{s}, \hat{h}) \\ \Rightarrow & \exists\, \hat{h}_0 : \; dom(\hat{h}_0) \cap dom(h_1) = \emptyset \;\land\; \hat{h} = \hat{h}_0 \cup h_1 \;\land\; ((s, h_0)\, C\, (\hat{s}, \hat{h}_0))\;. \end{array}$$

Intuitively, the frame property expresses that an execution of a command C can be tracked down to a possibly smaller heap portion it needs for the execution.

An immediate consequence of the definition is the following.

Lemma 8.2. *Consider a test relation p. Then command C is safe at all states in p iff $p \subseteq dom(C)$ holds.*

Let us now provide an algebraic statement of safety-monotonicity.

Lemma 8.3. *Command C is $*$-safety-monotonic iff*

$$dom(C) \cup I \subseteq dom(C)\;.$$

Proof. For arbitrary store s and heaps h, h' with $dom(h') \cap dom(h) = \emptyset$ we have

C is safe at $(s, h) \Rightarrow C$ is safe at $(s, h \cup h')$

$\Leftrightarrow$ $\{\!|$ Lemma 8.2 $|\!\}$

$(s, h) \in dom(C) \Rightarrow (s, h \cup h') \in dom(C)$

$\Leftrightarrow$ $\{\!|$ I contains every test element $|\!\}$

$(s, h) \in dom(C) \wedge (s, h') \in I \Rightarrow (s, h \cup h') \in dom(C)$

$\Leftrightarrow$ $\{\!|$ definition of $\uplus$ $|\!\}$

$dom(C) \uplus I \subseteq dom(C)$.

$\square$

Next, we formulate the $*$ frame property algebraically. This involves the decomposition operation as well as the Cartesian product of relations.

Lemma 8.4. *Command C has the $*$ frame property iff*

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

Proof. For all stores $s, \hat{s}$ and heaps $h_0, h_1, \hat{h}$ with $dom(h_1) \cap dom(h_0) = \emptyset$,

C is safe at $(s, h_0) \wedge (s, h_0 \cup h_1)\, C\, (\hat{s}, \hat{h})$

$\Rightarrow$ $\exists\, \hat{h}_0 : dom(\hat{h}_0) \cap dom(h_1) = \emptyset \wedge \hat{h} = \hat{h}_0 \cup h_1 \wedge (s, h_0 \cup h_1)\, C\, (\hat{s}, \hat{h})$

$\Leftrightarrow$ $\{\!|$ assumption, Lemma 8.2 $|\!\}$

$(s, h_0) \in dom(C) \wedge ((s, h_0) * (s * h_1))\, C\, (\hat{s}, \hat{h}_0))$

$\Rightarrow$ $\exists\, \hat{h}_0 : (\hat{s}, \hat{h}) = (\hat{s}, \hat{h}_0) * (\hat{s}, h_1) \wedge ((s, h_0)\, C\, (\hat{s}, \hat{h}_0))$

$\Leftrightarrow$ $\{\!|$ definitions $|\!\}$

$((s, h_0), (s * h_1))\, ((dom(C) \times I) \,;\, \overset{*}{\rhd} ; C)\, (\hat{s}, \hat{h}_0))$

$\Rightarrow$ $((s, h_0), (s * h_1))\ (C \times I) \,;\, \overset{*}{\rhd} (\hat{s}, \hat{h}_0))$.

$\square$

We will give a slightly modified algebraic formulation later on.

As a check for adequacy we show 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 $\overset{*}{\rhd}$ 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. As a consequence, we immediately get

$$\sigma \in dom([\![[e_1] := e_2]\!]_c) \Leftrightarrow e_1^s \in dom(h_\sigma) \tag{16}$$

From this, we get

$(\sigma_1, \sigma_2)\ (dom([\![[e_1] := e_2]\!]_c) \times I) \,;\, \overset{*}{\rhd} ; [\![[e_1] := e_2]\!]_c\ \tau$

$\Leftrightarrow$ $\{\!|$ definition of $\times$ and I $|\!\}$

$\sigma_1 \in dom([\![[e_1] := e_2]\!]_c) \wedge (\sigma_1, \sigma_2)\ \overset{*}{\rhd} ; [\![[e_1] := e_2]\!]_c\ \tau$

$\Leftrightarrow$ $\{\!|$ definition of $\overset{*}{\rhd}$ and (16) $|\!\}$

$(\sigma_1 * \sigma_2)\ [\![[e_1] := e_2]\!]_c\ \tau \wedge \sigma_1, \sigma_2$ combinable $\wedge\ e_1^s \in dom(h_{\sigma_1})$

$\Leftrightarrow$ $\{\!|$ definition of $[\![[e_1] := e_2]\!]_c$ $|\!\}$

$h_\tau = (e_1^s, e_2^s) \,|\, h_{(\sigma_1 * \sigma_2)} \wedge e_1^s \in dom(h_{(\sigma_1 * \sigma_2)}) \wedge \sigma_1, \sigma_2$ combinable $\wedge\ e_1^s \in dom(h_{\sigma_1})$

$\Leftrightarrow$ $\{\!|$ definition of $*$ $|\!\}$

$h_\tau = (e_1^s, e_2^s) \,|\, (h_{\sigma_1} \cup h_{\sigma_2}) \wedge dom(h_{\sigma_1}) \cap dom(h_{\sigma_2}) = \emptyset \wedge$
$e_1^s \in dom(h_{\sigma_1}) \cup dom(h_{\sigma_2}) \wedge \sigma_1, \sigma_2$ combinable $\wedge\ e_1^s \in dom(h_{\sigma_1})$

$\Leftrightarrow$ $\{\!|$ annihilation property for $|$ (see below), since $e_1^s \notin dom(h_{\sigma_2})$, isotony of dom $|\!\}$

$h_\tau = ((e_1^s, e_2^s) \,|\, h_{\sigma_1}) \cup h_{\sigma_2} \wedge dom(h_{\sigma_1}) \cap dom(h_{\sigma_2}) = \emptyset \wedge \sigma_1, \sigma_2$ combinable $\wedge\ e_1^s \in dom(h_{\sigma_1})$

$\Rightarrow$ $\{\!|$ since $e_1^s \notin dom(h_{\sigma_2})$ $|\!\}$

$h_\tau = ((e_1^s, e_2^s) \,|\, h_{\sigma_1}) \cup h_{\sigma_2} \wedge dom((e_1^s, e_2^s) \,|\, h_{\sigma_1}) \cap dom(h_{\sigma_2}) = \emptyset \wedge \sigma_1, \sigma_2$ combinable $\wedge\ e_1^s \in dom(h_{\sigma_1})$

$\Rightarrow$ $\{\!|$ logic, definition of $[\![[e_1] := e_2]\!]_c$ and $*$ $|\!\}$

$\sigma_1\ [\![[e_1] := e_2]\!]_c\ (s, (e_1^s, e_2^s) \,|\, h_{\sigma_1}) \wedge \tau = (s, (e_1^s, e_2^s) \,|\, h_{\sigma_1}) * \sigma_2 \wedge (s, (e_1^s, e_2^s) \,|\, h_{\sigma_1}), \sigma_2$ combinable

$\Leftrightarrow$ $\{\!|$ definitions $|\!\}$

$(\sigma_1, \sigma_2)\ ([\![[e_1] := e_2]\!]_c \times I)\ ;\ \circledast\!\!\triangleright\ \tau$

The annihilation property states that for partial function $f_1 \,|\, (f_2 \cup f_3) = (f_1 \,|\, f_2) \cup f_3$ if $dom(f_1) \notin dom(f_3)$. A proof for the annihilation property can be found in [31].

8.2. Preservation of Variables

It remains to express algebraically the requirement that a command does not modify certain variables. In this, we would like to avoid an explicit mention of syntax and free variables and find a suitable purely semantic condition instead. A logical formulation might read as follows: command C preserves a test r if for all stores $s, \hat{s}$ and heaps $h_0, h_1, \hat{h}_0$ with $dom(h_1) \cap dom(h_0) = \emptyset$,

$$\begin{array}{ll} & (s, h_0)\, C\, (\hat{s}, \hat{h}_0) \wedge (s, h_1) \models r \\ \Rightarrow & dom(h_1) \cap dom(\hat{h}_0) = \emptyset \wedge (s, h_0 \cup h_1)\, C\, (\hat{s}, \hat{h}_0 \cup h_1)\,. \end{array}$$

An attempt to bring this into pointfree form as before fails, however, since the result state $(\hat{s}, \hat{h}_0 \cup h_1)$ would need to be composed from $(\hat{s}, \hat{h}_0)$ and (s, h_1), which is not possible using the $*$ operation, as s and $\hat{s}$ will be different in general.

To remedy this, we introduce a generalised decomposition operator in the following section.

8.3. A Generalised Separation Operator

We relaxate the requirement that in the separating conjunction the states have to agree completely and only stipulate that they be compatible in the following sense.

Definition 8.5. Let A, B be sets.

(a) The *restriction* of a partial map $s : A \to B$ to a subset $X \subseteq A$ is $s|_X$ with $dom(s|_X) =_{df} dom(S) \cap X$ and $s|_X(x) =_{df} s(x)$ for $x \in dom(s|_X)$. In particular, $s|_\emptyset = \emptyset$.

(b) Two partial maps $s_1, s_2 : A \to B$ are *compatible* if they agree on the intersection of their domains, i.e., if

$$s_1|_{dom(s_2)} = s_2|_{dom(s_1)}\,.$$

Lemma 8.6. *Consider partial maps $s_1, s_2 : A \to B$.*

(a) *If $dom(s_1) \cap dom(s_2) = \emptyset$ then s_1 and s_2 are compatible.*

(b) *If s_1 and s_2 are compatible then $s_1 \cup s_2$ is a partial map again.*

(c) *Every partial map is compatible with all its submaps, in particular, with itself and with $\emptyset$.*

The proof can be found in the Appendix B.

For pointfree proofs of these properties in relation algebra see [31]. The property $s_2 = s_1|_{dom(s_2)}$ for all $s_2 \subseteq s_1$ may even serve as an abstract characterisation of partial maps and of determinacy [16].

We now define our generalised combination operator as follows.

Definition 8.7.

(a) Two states (s_1, h_1) and (s_2, h_2) are $\circledast$*-combinable* if the states s_1 and s_2 are compatible and $dom(h_1) \cap dom(h_2) = \emptyset$.

(b) For $\circledast$-combinable states (s_1, h_1) and (s_2, h_2) we define their *combination* by

$$(s_1, h_1) \circledast (s_2, h_2) =_{df} (s_1 \cup s_2, h_1 \cup h_2)\,.$$

From this and Lemma 8.6(c) it is immediate that, when $dom(h_1) \cap dom(h_2) = \emptyset$,

$$(s, h_1) * (s, h_2) \;=\; (s, h_1) \boxast (s, h_2)\ .$$

Lemma 8.8. *The operator $\boxast$ is commutative and associative on pairwise combinable states and has the state $(\emptyset, \emptyset)$ as neutral element.*

Proof. Straightforward from the definitions. □

The operator $\boxast$ admits arbitrary compatible splits of the store while preserving the requirement of disjointness of the heap parts. It will allow a convenient definition of non-interacting parallel composition in the next section. Moreover, it is useful when dealing with ghost variables for verification purposes, see [12]. The use of compatible union on the store part was inspired by the concept of the demonic $R_1 \sqcap R_2$ meet of relations $R_1, R_2 \subseteq A \times B$ for some sets A, B, which is defined as (see e.g. [6])

$$R_1 \sqcap R_2 \;=_{df}\; (R_1 - (dom(R_2) \times B)) \cup (R_2 - (dom(R_1) \times B)) \cup (R_1 \cap R_2)\ .$$

When R_1 and R_2 are compatible in the sense that $dom((R_1 \cap R_2)) = dom(R_1) \cap dom(R_2)$ the element $R_1 \sqcap R_2$ is their meet w.r.t. the demonic refinement relation. If R_1 and R_2 are partial maps compatibility is equivalent to the formula above and $R_1 \sqcap R_2 = R_1 \cup R_2$.

Definition 8.9. The *split* relation $\mathrel{\triangleleft\!\boxast}$ and its converse, the *join* relation $\mathrel{\boxast\!\triangleright}$ as well as $\boxast$-composition are defined as in Definition 7.2, replacing $*$ by $\boxast$.

Let us look at the pointwise meaning of $\boxast$ composition:

$$\sigma\,(C_1 \boxast C_2)\,\tau \;\Leftrightarrow\; \exists\,\sigma_1, \sigma_2, \tau_1, \tau_2 : \sigma = \sigma_1 \boxast \sigma_2 \;\wedge\; \tau = \tau_1 \boxast \tau_2 \;\wedge\; \sigma_1\, C_1\, \tau_1 \;\wedge\; \sigma_2\, C_2\, \tau_2\ .$$

Note that the decompositions of σ and τ each have to use compatible stores, i.e., stores in which the values of shared variables agree.

Using these new operators we propose the following point-free formulations for the three essential properties in terms of our generalised decomposition operator, in which the domain assertion for the frame property is dropped, because that property will only be used when the domain assertion holds anyway.

Definition 8.10. Consider a command C and a test r.

(a) C is *safety-monotonic* iff $dom(C) \boxast I \;\subseteq\; dom(C)$.

(b) C has the *frame property* iff $(dom(C) \times I)\ ;\ \mathrel{\boxast\!\triangleright}\ ;\ C \;\subseteq\; (C \times I)\ ;\ \mathrel{\boxast\!\triangleright}$.

(c) C *preserves* r iff $\mathrel{\triangleleft\!\boxast}\ ;\ (C \times r) \;\subseteq\; C\ ;\ \mathrel{\triangleleft\!\boxast}$.

Pointwise the latter spells out to

$$\begin{array}{ll} & (\sigma_0\, C\, \tau_0 \;\wedge\; \sigma_1 \;\models\; r \;\wedge\; \sigma_0, \sigma_1\ \boxast\text{-combinable} \\ \Rightarrow & \tau_0, \sigma_1\ \boxast\text{-combinable} \;\wedge\; (\sigma_0 \boxast \sigma_1)\, C\, (\tau_0 \boxast \sigma_1)\ . \end{array}$$

In particular, this expresses that the stores of τ_0 and σ_1 have to be compatible and hence the parts of these stores that mention the variables of r have to be identical; in other words, C is not allowed to change these variables. Moreover, (c) is equivalent to $\mathrel{\triangleleft\!\boxast}\ ;\ (C \times r) \;\subseteq\; C\ ;\ \mathrel{\triangleleft\!\boxast}\ ;\ (I \times r)$.

8.4. Soundness of the Frame Rule

Now we are in the position to give a purely algebraic proof of the frame rule for the $\boxast$ combination of assertions.

Theorem 8.11. *Consider a command C and a test r such that C has the frame property, is safety-monotonic and preserves r. Then the inference rule*

$$\frac{\{\!\!\{p\}\!\!\}\, C\, \{\!\!\{q\}\!\!\}}{\{\!\!\{p \boxast r\}\!\!\}\, C\, \{\!\!\{q \boxast r\}\!\!\}}$$

is valid.

Proof. Suppose $\{\!\!\{p\}\!\!\}\, c\, \{\!\!\{q\}\!\!\}$, i.e., $p \subseteq dom(C) \wedge \{p\}\, C\, \{q\}$. By safety-monotonicity of C we have $dom(C) \boxast I \subseteq dom(C)$ and hence, by isotony, $p \boxast r \subseteq dom(C) \boxast I \subseteq dom(C)$ as well. Moreover we calculate

$$
\begin{array}{ll}
 & (p \boxast r)\,;C \\
= & \quad \{\!\![\ \text{definition of } \boxast\ ]\!\!\} \\
 & \lhd_{\boxast}\,;(p \times r)\,;\rhd_{\boxast}\,;C \\
= & \quad \{\!\![\ \text{Equation (15) and } p \leq dom(C)\ ]\!\!\} \\
 & \lhd_{\boxast}\,;(p \times r)\,;(dom(C) \times I)\,;\rhd_{\boxast}\,;C \\
\subseteq & \quad \{\!\![\ \text{since } C \text{ has the frame property}\ ]\!\!\} \\
 & \lhd_{\boxast}\,;(p \times r)\,;(C \times I)\,;\rhd_{\boxast} \\
\subseteq & \quad \{\!\![\ \text{exchange law (14)}\ ]\!\!\} \\
 & \lhd_{\boxast}\,;(p\,;C) \times (r\,;I)\,;\rhd_{\boxast} \\
\subseteq & \quad \{\!\![\ \text{by } \{p\}\, C\, \{q\} \text{ and } r\,;I = r = r\,;r, \text{ since } r \text{ is a test}\ ]\!\!\} \\
 & \lhd_{\boxast}\,;(C\,;q) \times (r\,;r)\,;\rhd_{\boxast} \\
= & \quad \{\!\![\ \text{exchange law (14)}\ ]\!\!\} \\
 & \lhd_{\boxast}\,;(C \times r)\,;(q \times r)\,;\rhd_{\boxast} \\
\subseteq & \quad \{\!\![\ \text{since } C \text{ preserves } r\ ]\!\!\} \\
 & C\,;\lhd_{\boxast}\,;(q \times r)\,;\rhd_{\boxast} \\
= & \quad \{\!\![\ \text{definition of } \boxast\ ]\!\!\} \\
 & C\,;(q \boxast r)\,.
\end{array}
$$

□

Since only very few properties of the relational operators are used, this proof as well as the one in the following subsection carries over to the demonic setting where $\cup$ and ; are replaced by their demonic variants. The details of this will be spelled out in a subsequent paper.

8.5. Algebraic Proof of the Frame Rule for $*$ Decomposition

With an analogous method as in the previous sections we can now give a completely algebraic soundness proof of the standard frame rule, however, at the expense of more complicated formulations of the corresponding frame and preservation properties. We start by a definition to relate the splittings w.r.t. $*$ and $\boxast$.

Definition 8.12. The partial identity Q characterises those pairs of states which are combinable:

$$(\sigma_1, \sigma_2)\, Q\, (\tau_1, \tau_2) \;\Leftrightarrow_{df}\; \sigma_1 \text{ and } \sigma_2 \text{ } *\text{-combinable and } \sigma_1 = \tau_1 \text{ and } \sigma_2 = \tau_2.$$

With its help we can express the connection between our split and join operations.

Corollary 8.13.

1. $\lhd_{*} = \lhd_{\boxast}\,;Q$ *and* $\rhd_{*} = Q\,;\rhd_{\boxast}$.
2. $\lhd_{*} = \lhd_{*}\,;Q$ *and* $\rhd_{*} = Q\,;\rhd_{*}$.

Definition 8.14.

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

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

2. Command C has the $*$-*frame-property* if

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

3. Command C $*$-*preserves* test r if

$$\overset{*}{\lhd}\,;\; (C \times (r\,;H))\,; Q \;\subseteq\; C\,;\, \overset{*}{\lhd}\,;\; (I \times r)\;.$$

It is clear that H is an equivalence relation and hence $H\,;H = H$. To spell out the pointwise meaning of the $*$-frame-property and $*$-preservation we use the following notation.

Definition 8.15. Let $\sigma \;=_{df}\; (s,h)$ be a state and h' be a heap.

1. $h' \;\subseteq\; \sigma \;\Leftrightarrow_{df}\; h' \;\subseteq\; h$.
2. If $h' \;\subseteq\; \sigma$ then $\sigma \cap h' \;=_{df}\; (s, h \cap h')$ and $\sigma - h' \;=_{df}\; (s, h - h')$.

This immediately entails the following decomposition property.

Corollary 8.16. *For $h' \;\subseteq\; \sigma$ we have $\sigma = (\sigma \cap h') * (\sigma - h')$.*

Now straightforward calculations show the following.

Lemma 8.17.

1. *Command C has the $*$-frame-property iff, for all σ, τ, h,*

$$h \subseteq \sigma \;\wedge\; \sigma\, C\, \tau \;\Rightarrow\; h \;\subseteq\; \tau \;\wedge\; (\sigma - h)\, C\, (\tau - h)\;.$$

2. *Command C $*$-preserves test r iff, for all σ, τ, h,*

$$h \subseteq \sigma \;\wedge\; \sigma\, C\, \tau \;\wedge\; \sigma \cap h \models r \;\Rightarrow\; h \;\subseteq\; \tau \;\wedge\; \tau \cap h \models r\;.$$

The above notions now allow a purely algebraic proof of the $*$ frame rule.

Theorem 8.18. *Consider a command C and a test r such that C has the $*$-frame-property, is $*$-safety-monotonic and $*$-preserves r. Then the inference rule*

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

is valid.

Proof. Again we can infer by assumption, isotony and $*$-safety-monotonicity, $p * r \subseteq dom(C) * I \subseteq dom(C)$. Furthermore we calculate

$(p * r)\,;C$

$=\quad \{\![$ definition of $*$ $]\!\}$

$\lhd^{*}\,;\,(p \times r)\,;\,\rhd^{*}\,;C$

$=\quad \{\![$ Equation (15) and $p \le dom(C)$ $]\!\}$

$\lhd^{*}\,;\,(p \times r)\,;(dom(C) \times I)\,;(C \times H)\,;\,\rhd^{*}$

$\subseteq\quad \{\![$ since C has the $*$-frame-property $]\!\}$

$\lhd^{*}\,;\,(p \times r)\,;(C \times H)\,;\,\rhd^{*}$

$\subseteq\quad \{\![$ exchange $]\!\}$

$\lhd^{*}\,;\,(p\,;C) \times (r\,;H)\,;\,\rhd^{*}$

$\subseteq\quad \{\![$ by $\{p\}\,C\,\{q\}$ and isotony $]\!\}$

$\lhd^{*}\,;\,(C\,;q) \times (r\,;H)\,;\,\rhd^{*}$

$=\quad \{\![$ neutrality of I w.r.t. composition and exchange $]\!\}$

$\lhd^{*}\,;\,(C \times (r\,;H))\,;(q \times I)\,;\,\rhd^{*}$

$=\quad \{\![$ by Corollary 8.13.2 $]\!\}$

$\lhd^{*}\,;\,(C \times (r\,;H))\,;(q \times I)\,;Q\,;\,\rhd^{*}$

$=\quad \{\![$ $(q \times I)$ and Q are partial identities and hence commute $]\!\}$

$\lhd^{*}\,;\,(C \times (r\,;H))\,;Q\,;(q \times I)\,;\,\rhd^{*}$

$\subseteq\quad \{\![$ C $*$-preserves r $]\!\}$

$C\,;\lhd^{*}\,;\,(I \times r)\,;(q \times I)\,;\,\rhd^{*}$

$=\quad \{\![$ neutrality of I and exchange $]\!\}$

$C\,;\lhd^{*}\,;\,(q \times r)\,;\,\rhd^{*}$

$=\quad \{\![$ definition of $*$ $]\!\}$

$C\,;(q * r)\,.$

□

Next we show again exemplarily that the mutation statement $[e_1] := e_2$ $*$-preserves an arbitrary test r. Again we denote the heap of a state σ by h_σ. As in the proof that mutation satisfies the frame property, all stores are fixed.

$\sigma\ \lhd^{*}\,;\,(\llbracket [e_1] := e_2 \rrbracket_c \times (r\,;H))\,;Q\ (\tau_1,\tau_2)$

$\Leftrightarrow\quad \{\![$ definition of $\lhd^{*}$ $]\!\}$

$\exists\,\sigma_1,\sigma_2 : \sigma = \sigma_1 * \sigma_2 \ \land\ (\sigma_1,\sigma_2)(\llbracket [e_1] := e_2 \rrbracket_c \times (r\,;H))\,;Q\ (\tau_1,\tau_2)$

$\Leftrightarrow\quad \{\![$ definition of Q $]\!\}$

$\exists\,\sigma_1,\sigma_2 : \sigma = \sigma_1 * \sigma_2 \ \land\ (\sigma_1,\sigma_2)(\llbracket [e_1] := e_2 \rrbracket_c \times (r\,;H))(\tau_1,\tau_2) \ \land\ \tau_1,\tau_2 *-$combinable

$\Leftrightarrow\quad \{\![$ definition of $\times$ $]\!\}$

$\exists\,\sigma_1,\sigma_2 : \sigma = \sigma_1 * \sigma_2 \ \land\ \sigma_1\,\llbracket [e_1] := e_2 \rrbracket_c\,\tau_1 \ \land\ \sigma_2(r\,;H)\tau_2 \ \land\ \tau_1,\tau_2 *-$combinable

$\Leftrightarrow\quad \{\![$ definition of $\llbracket [e_1] := e_2 \rrbracket_c$ and r $]\!\}$

$\exists\,\sigma_1,\sigma_2 : \sigma = \sigma_1 * \sigma_2 \ \land\ h_{\tau_1} = (e_1^s, e_2^s)\,|\,h_{\sigma_1} \ \land\ e_1^s \in dom(h_{\sigma_1}) \ \land\ \sigma_2 H \tau_2 \ \land\ \sigma_2 \models r$
$\land\,\tau_1,\tau_2 *-$combinable

$\Leftrightarrow\quad \{\![$ since no command changes stores and τ_1,τ_2 $*$-combinable enforces $s_{\tau_1} = s_{\tau_2}$ $]\!\}$

$\exists\,\sigma_1,\sigma_2 : \sigma = \sigma_1 * \sigma_2 \ \land\ h_{\tau_1} = (e_1^s, e_2^s)\,|\,h_{\sigma_1} \ \land\ e_1^s \in dom(h_{\sigma_1}) \ \land\ \sigma_2 = \tau_2 \ \land\ \sigma_2 \models r$
$\land\,\tau_1,\tau_2 *-$combinable

$\Rightarrow\quad \{\![$ logic and $\sigma_2 = \tau_2$ $]\!\}$

$\exists\,\sigma_1 : \sigma = \sigma_1 * \tau_2 \ \land\ h_{\tau_1} = (e_1^s, e_2^s)\,|\,h_{\sigma_1} \ \land\ e_1^s \in dom(h_{\sigma_1}) \ \land\ dom(h_{\tau_1}) = dom(h_{\sigma_1}) \ \land\ \tau_2 \models r$
$\land\,\tau_1,\tau_2 *-$combinable

$\Rightarrow\quad \{\![$ isotony of $\cup$, definition of $*$-combinable and $dom(h_{\tau_1}) = dom(h_{\sigma_1})$ $]\!\}$

$\exists\,\sigma_1 : \sigma = \sigma_1 * \tau_2 \ \land\ h_{\tau_1} \cup h_{\tau_2} = ((e_1^s, e_2^s)\,|\,h_{\sigma_1}) \cup h_{\tau_2} \ \land\ e_1^s \in dom(h_{\sigma_1}) \,\land$
$\land\, dom(h_{\sigma_1}) \cap dom(h_{\tau_2}) = \emptyset \ \land\ \tau_2 \models r \ \land\ dom(h_{\tau_1}) \cap dom(h_{\tau_2}) = \emptyset$

$\Rightarrow\quad \{\![$ annihilation of $|$ (see above) and definition of $*$ $]\!\}$

$\exists\, \sigma_1 : \sigma = \sigma_1 \,*\, \tau_2 \;\wedge\; h_{(\tau_1 * \tau_2)} = (e_1^s, e_2^s) \,|\, h_{(\sigma_1 * \tau_2)} \;\wedge\; e_1^s \in dom(h_{\sigma_1}) \;\wedge\; dom(h_{\tau_1}) \cap dom(h_{\tau_2}) = \emptyset \;\wedge\; \tau_2 \models r$

$\Rightarrow$ ⦃ logic, isotony of *dom* and $\sigma = \sigma_1 \,*\, \tau_2$ ⦄

$h_{(\tau_1 * \tau_2)} = (e_1^s, e_2^s) \,|\, h_\sigma \;\wedge\; e_1^s \in dom(h_{\sigma_1}) \;\wedge\; dom(h_{\tau_1}) \cap dom(h_{\tau_2}) = \emptyset \;\wedge\; \tau_2 \models r$

$\Leftrightarrow$ ⦃ Equation (16) ⦄

$h_{(\tau_1 * \tau_2)} = (e_1^s, e_2^s) \,|\, h_\sigma \;\wedge\; \sigma \in dom([\![[e_1] := e_2]\!]_c) \;\wedge\; dom(h_{\tau_1}) \cap dom(h_{\tau_2}) = \emptyset \;\wedge\; \tau_2 \models r$

$\Leftrightarrow$ ⦃ definition of $[\![[e_1] := e_2]\!]_c$ ⦄

$\sigma\; [\![[e_1] := e_2]\!]_c \tau_1 \,*\, \tau_2 \;\wedge\; dom(h_{\tau_1}) \cap dom(h_{\tau_2}) = \emptyset \;\wedge\; \tau_2 \models r$

$\Leftrightarrow$ ⦃ definition of $\lhd$ and combinability ⦄

$\sigma\; [\![[e_1] := e_2]\!]_c \,;\, \lhd (\tau_1, \tau_2) \;\wedge\; \tau_2 \models r$

$\Leftrightarrow$ ⦃ definition of $\times$ and r ⦄

$\sigma\; [\![[e_1] := e_2]\!]_c \,;\, \lhd \,;\, (I \times r)\; (\tau_1, \tau_2)$

8.6. Closure Properties

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

Lemma 8.19. *The* $\circledast$*-frame property and* $\circledast$*-preservation of a test r are closed under union and composition of commands assuming that for a composition* $C \,;\, D$ *the inequation* $cod(C) \subseteq dom(D)$ *holds.*

Proof. Closure under union is straightforward from distributivity of ; and $\times$. Next, assume that C and D have the $\circledast$-frame property. Then

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

$\subseteq$ ⦃ by isotony ⦄

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

$\subseteq$ ⦃ C has the $\circledast$-frame property ⦄

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

$=$ ⦃ exchange ⦄

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

$\subseteq$ ⦃ by the assumption $cod(C) \subseteq dom(D)$ and isotony ⦄

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

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

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

$=$ ⦃ exchange ⦄

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

$=$ ⦃ neutrality of I ⦄

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

Now assume that C and D $\circledast$-preserve r. Then

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

$=$ ⦃ r is a test and hence idempotent ⦄

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

$=$ ⦃ exchange ⦄

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

$\subseteq$ ⦃ C $\circledast$-preserves r ⦄

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

$\subseteq$ ⦃ D $\circledast$-preserves r ⦄

$C \,;\, D \,;\, \lhd\,.$

$\square$

Lemma 8.20. *Also the $*$-frame property and $*$-preservation of a test r are closed under union and composition of commands assuming that for a composition $C \,;\, D$ the inequation $cod(C) \subseteq dom(D)$ holds.*

Proof. Closure under union is straightforward from distributivity of ; and $\times$. Next, assume that C and D have the $*$-frame property. Then

$$
\begin{array}{ll}
 & (dom(C \,;\, D) \times I) \,;\, \triangleright_{*} \,;\, C \,;\, D \\
\subseteq & \quad \{\!|\ \text{by isotony}\ |\!\} \\
 & (dom(C) \times I) \,;\, \triangleright_{*} \,;\, C \,;\, D \\
\subseteq & \quad \{\!|\ C \text{ has the } *\text{-frame property}\ |\!\} \\
 & (C \times H) \,;\, \triangleright_{*} \,;\, D \\
= & \quad \{\!|\ \text{exchange}\ |\!\} \\
 & (C \times H) \,;\, (cod(C) \times H) \,;\, \triangleright_{*} \,;\, D \\
\subseteq & \quad \{\!|\ \text{by the assumption } cod(C) \subseteq dom(D) \text{ and isotony}\ |\!\} \\
 & (C \times H) \,;\, (dom(D) \times I) \,;\, \triangleright_{*} \,;\, D \\
\subseteq & \quad \{\!|\ D \text{ has the } *\text{-frame property}\ |\!\} \\
 & (C \times H) \,;\, (D \times H) \,;\, \triangleright_{*} \\
= & \quad \{\!|\ \text{exchange}\ |\!\} \\
 & (C \,;\, D) \times (H \,;\, H) \,;\, \triangleright_{*} \\
= & \quad \{\!|\ H \text{ is an equivalence relation}\ |\!\} \\
 & (C \,;\, D) \times H \,;\, \triangleright_{*} \,.
\end{array}
$$

Now assume that C and D $*$-preserve r. Then

$$
\begin{array}{ll}
 & \triangleleft_{*} \,;\, ((C \,;\, D) \times (r \,;\, H) \,;\, Q \\
= & \quad \{\!|\ H \text{ is an equivalence relation}\ |\!\} \\
 & \triangleleft_{*} \,;\, ((C \,;\, D) \times (r \,;\, H \,;\, H) \,;\, Q \\
= & \quad \{\!|\ \text{exchange}\ |\!\} \\
 & \triangleleft_{*} \,;\, (C \times (r \,;\, H)) \,;\, (D \times H) \,;\, Q \\
\subseteq & \quad \{\!|\ C \ *\text{-preserves } r\ |\!\} \\
 & C \,;\, \triangleleft_{*} \,;\, (I \times r) \,;\, (D \times H) \,;\, Q \\
= & \quad \{\!|\ \text{exchange and neutrality of } I\ |\!\} \\
 & C \,;\, \triangleleft_{*} \,;\, (D \times (r \,;\, H)) \,;\, Q \\
\subseteq & \quad \{\!|\ D \ *\text{-preserves } r\ |\!\} \\
 & C \,;\, D \,;\, \triangleleft_{*} \,;\, (I \times r) \,.
\end{array}
$$

$\square$

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 [42] 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 were able to give algebraic characterisations of properties on which the frame rule relies. In particular, we are able to define the side condition that certain variables may not be changed by a command in a purely semantic way without

appealing to the syntax. Moreover we have algebraically proved the frame rule and defined a more liberal version of it that rests on simpler side conditions. Finally we have shown that 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 [12]. 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 [35]); 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

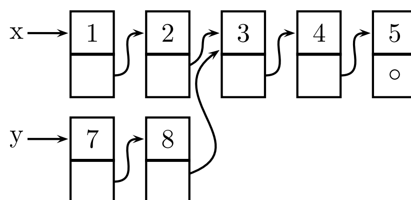


Figure 2: Two lists with shared cells.

towards an algebraic handling of such situations are given in [32, 19]. In future work, we will adapt these approaches for our algebra of separation logic.

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. However at the moment higher-order systems can only verify simply theorems fully automatically [13]. 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 [39]. One central property mentioned in this paper is the rule of *disjoint concurrency* which reads as

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

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.

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] N. Boudriga, F. Elloumi, and A. Mili. On the lattice of specifications: Applications to a specification methodology. *Formal Asp. Comput.*, 4(6):544–571, 1992.

[7] 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.

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

[9] C. Calcagno, P. W. 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.

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

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

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

[13] 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.

[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 July 31, 2010).

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

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

[28] 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.

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

[30] W. W. McCune. Prover9 and Mace4.
<`http://www.cs.unm.edu/~mccune/prover9`>. (accessed July 31, 2010).

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

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

[33] B. Möller. Residuals and detachments. Technical Report 2005-20, Institute of Computer Science, University of Augsburg, 2005.

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

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

[36] 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.

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

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

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

[40] P. W. O'Hearn, J. C. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In L. Fribourg, editor, *CSL '01: 15th International Workshop on Computer Science Logic*, volume 2142 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2001.

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

[42] 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.

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

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

[45] 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.

A. Deferred Properties

In this appendix, we list a couple of different properties for separation logic and for our algebraic characterisation.

A.1. Septraction

From Lemma 5.8 and the definitions, we immediately get

Corollary A.1.

1. $s, h \models p -\!\circledast\, \mathsf{true} \;\Leftrightarrow\; \exists\, \hat{h} : h \subseteq \hat{h},\ s, \hat{h} \models p.$
2. $s, h \models (p * q) -\!\circledast\, \mathsf{true} \;\Leftrightarrow\; \exists\, \hat{h}, \hat{h}_1, \hat{h}_2 : h \subseteq \hat{h},\ \hat{h}_1 \cup \hat{h}_2 = \hat{h},$ $dom(\hat{h}_1) \cap dom(\hat{h}_2) = \emptyset,\ s, \hat{h}_1 \models p,\ s, \hat{h}_2 \models q\ .$
3. $s, h \models p * (q -\!\circledast\, \mathsf{true}) \;\Leftrightarrow\; \exists\, h_1, h_2, \hat{h}_2 : h_1 \cup h_2 = h,\ h_2 \subseteq \hat{h}_2,$ $dom(h_1) \cap dom(h_2) = \emptyset,\ s, h_1 \models p,\ s, \hat{h}_2 \models q\ .$
4. $s, h \models \mathsf{true} -\!\circledast\, q \;\Leftrightarrow\; \exists\, h' : dom(h) \cap dom(h') = \emptyset,\ s, h' \models q\ .$
5. $s, h \models p -\!\circledast\, (\mathsf{true} -\!\circledast\, q) \;\Leftrightarrow\; \exists\, h', \hat{h} : h \subseteq \hat{h},\ dom(\hat{h} - h) \cap dom(h') = \emptyset,$ $s, \hat{h} \models p,\ s, h' \models q\ .$

Intuitively, $p -\!\circledast\, \mathsf{true}$ holds in a state iff p holds in some extension of that state.

Theorem A.2.

$$(p * q) -\!\circledast\, \mathsf{true} \subseteq (p -\!\circledast\, \mathsf{true}) * (q -\!\circledast\, \mathsf{true})$$

Proof. We show $s, h \models (p * q) -\!\circledast\, \mathsf{true} \Rightarrow s, h \models (p -\!\circledast\, \mathsf{true}) * (q -\!\circledast\, \mathsf{true})$.

$$\begin{array}{ll}
& s, h \models (p * q) -\!\circledast\, \mathsf{true} \\
\Leftrightarrow & \{\!\![\text{ Corollary 1.2(2) }]\!\!\} \\
& \exists\, \hat{h}, \hat{h}_1, \hat{h}_2 : h \subseteq \hat{h},\ \hat{h}_1 \cup \hat{h}_2 = \hat{h}, dom(\hat{h}_1) \cap dom(\hat{h}_2) = \emptyset, \\
& s, \hat{h}_1 \models p,\ s, \hat{h}_2 \models q \\
\Rightarrow & \{\!\![\text{ set } g_1 = \hat{h}_1 \cap h \text{ and } g_2 = \hat{h}_2 \cap h]\!\!\} \\
& \exists\, g_1, g_2, \hat{h}_1, \hat{h}_2 : g_1 \cup g_2 = h, dom(g_1) \cap dom(g_2) = \emptyset,\ g_1 \subseteq \hat{h}_1,\ g_2 \subseteq \hat{h}_2, \\
& s, \hat{h}_1 \models p,\ s, \hat{h}_2 \models q \\
\Leftrightarrow & \{\!\![\text{ Corollary 1.2(1) (twice) }]\!\!\} \\
& \exists\, g_1, g_2 : g_1 \cup g_2 = h, dom(g_1) \cap dom(g_2) = \emptyset, \\
& s, g_1 \models p -\!\circledast\, \mathsf{true},\ s, g_2 \models q -\!\circledast\, \mathsf{true} \\
\Leftrightarrow & \{\!\![\text{ definition of } *]\!\!\} \\
& s, h \models (p -\!\circledast\, \mathsf{true}) * (q -\!\circledast\, \mathsf{true})
\end{array}$$

□

The converse direction does not hold.

Theorem A.3.

$$s, h \models (p -\!\circledast\, \mathsf{true}) * (q -\!\circledast\, \mathsf{true}) \not\Rightarrow s, h \models (p * q) -\!\circledast\, \mathsf{true}$$

Proof. The counterexample consists of only one variable and a heap with one single cell having address a:

$$s =_{df} \{(x, a)\}\ , \qquad h =_{df} \{(a, 1)\}\ .$$

Setting $p = q = (x \mapsto 1)$ we get immediately $s, h \models (p * q) -\!\circledast\, \mathsf{true} = \mathsf{false} -\!\circledast\, \mathsf{true} = \mathsf{false}$ (cf. Law 5.5 of [33]). Therefore it is sufficient to show that $s, h \models (p -\!\circledast\, \mathsf{true}) * (q -\!\circledast\, \mathsf{true})$ is satisfied.

$$
\begin{array}{ll}
& s, \{(a,1)\} \models ((x \mapsto 1) -\!\!\circledast \text{ true}) * ((x \mapsto 1) -\!\!\circledast \text{ true}) \\
\Leftrightarrow & \{\!\!\{ \text{ definition of } * \text{ and Corollary 1.2(1) (twice) } \}\!\!\} \\
& \exists\, h_1, h_2, \hat{h}_1, \hat{h}_2 :, h_1 \cup h_2 = h,\ dom(h_1) \cap dom(h_2) = \emptyset, \\
& h_1 \subseteq \hat{h}_1,\ h_2 \subseteq \hat{h}_2,\ s,\ \hat{h}_1 \models (x \mapsto 1),\ \hat{h}_2 \models (x \mapsto 1) \\
\Leftarrow & \{\!\!\{ \text{ set } h_1 = \hat{h}_1 = \hat{h}_2 = h = \{(a,1)\} \text{ and } h_2 = \emptyset \}\!\!\} \\
& \text{true}
\end{array}
$$

□

A.2. Pure Assertions

Lemma A.4. *If t is a test element then the element $t \cdot \top$ is pure.*

Proof. We use the characterisation of Lemma 6.10(b) and simply calculate $((t \cdot \top) \sqcap 1) \cdot \top = (t \cdot (\top \sqcap 1)) \cdot \top = t \cdot \top$ which follows immediately from the equation (testdist). □

We now investigate the interplay between pure assertions and algebraic residuals.

Corollary A.5. *For arbitrary assertions a, b and c $a \sqcap (b\backslash c) \leq (a \sqcap b)\backslash c$ holds.*

Lemma A.6. *For pure element a and arbitrary elements b and c the following (in)equations hold:*

$$(a \sqcap b)\backslash(a \sqcap c) = (a \sqcap b)\backslash c \tag{A.1}$$
$$a \sqcap (b\backslash c) \leq b\backslash(a \sqcap c) \tag{A.2}$$
$$a \sqcap (b\backslash c) = a \sqcap (b\backslash(a \sqcap c)) \tag{A.3}$$
$$a \sqcap (b\backslash c) = a \sqcap ((a \sqcap b)\backslash c) \tag{A.4}$$
$$a \sqcap (b\backslash c) = a \sqcap ((a \sqcap b)\backslash(a \sqcap c)) \tag{A.5}$$
$$a \sqcap ((a \sqcap b)\backslash(a \sqcap c)) \leq b\backslash(a \sqcap c) \tag{A.6}$$

Proof. To show (A.1) we use the proof principle of indirect equality. By definition, pureness, shunting, distributivity, shunting, Lemma 6.10(c), and definition:

$$
\begin{array}{ll}
\forall\, x: & x \leq (a \sqcap b)\backslash(a \sqcap c) \\
\Leftrightarrow \forall\, x: & (a \sqcap b) \cdot x \leq a \sqcap c \\
\Leftrightarrow \forall\, x: & a \sqcap (b \cdot x) \leq a \sqcap c \\
\Leftrightarrow \forall\, x: & b \cdot x \leq \overline{a} + (a \sqcap c) \\
\Leftrightarrow \forall\, x: & b \cdot x \leq (a + \overline{a}) \sqcap (\overline{a} + c) \\
\Leftrightarrow \forall\, x: & a \sqcap (b \cdot x) \leq c \\
\Leftrightarrow \forall\, x: & (a \sqcap b) \cdot x \leq c \\
\Leftrightarrow \forall\, x: & x \leq (a \sqcap b)\backslash c
\end{array}
$$

Next we give a proof of Part (A.2). By definition, Lemma 6.10(c), and property of residual, isotony:

$$
\begin{array}{l}
\quad a \sqcap (b\backslash c) \leq b\backslash(a \sqcap c) \\
\Leftrightarrow b \cdot (a \sqcap (b\backslash c)) \leq a \sqcap c \\
\Leftrightarrow a \sqcap (b \cdot (b\backslash c)) \leq a \sqcap c \\
\Leftrightarrow \text{true}
\end{array}
$$

To prove (A.3) the $\leq$-direction follows immediately by (A.2) . The $\geq$-direction can be shown as follows: By definition of $\sqcap$, definition, Lemma 6.10(c), and property of residual, idempotency, isotony:

$$
\begin{array}{l}
\quad a \sqcap (b\backslash(a \sqcap c)) \leq a \sqcap (b\backslash c) \\
\Leftrightarrow a \sqcap (b\backslash(a \sqcap c)) \leq b\backslash c \\
\Leftrightarrow b \cdot (a \sqcap (b\backslash(a \sqcap c))) \leq c \\
\Leftrightarrow a \sqcap (b \cdot (b\backslash(a \sqcap c))) \leq c \\
\Leftrightarrow \text{true}
\end{array}
$$

The $\leq$-direction of (A.4) follows immediately by Corollary A.5. By definition of $\sqcap$, definition, Lemma 6.10(c), and property of residual:

$$
\begin{aligned}
& a \sqcap ((a \sqcap b)\backslash c) \leq a \sqcap (b\backslash c) \\
\Leftrightarrow\ & a \sqcap ((a \sqcap b)\backslash c) \leq b\backslash c \\
\Leftrightarrow\ & b \cdot (a \sqcap ((a \sqcap b)\backslash c)) \leq c \\
\Leftrightarrow\ & (a \sqcap b) \cdot ((a \sqcap b)\backslash c) \leq c \\
\Leftrightarrow\ & \mathsf{true}
\end{aligned}
$$

Finally (A.5) follows immediately from (A.4) and (A.1) and Equation (A.6) follows from (A.5) and (A.2). □

After looking at the interplay between pure assertions and residuals we now turn to detachments interacting with this class of assertions. In fact, we get analogous import and export laws for detachments as in the case of pure assertions interacting with the separation conjunction in Lemma 6.10(c).

Lemma A.7. *For arbitrary elements b, c and pure a the equation* $(a \sqcap b)\lfloor(a \sqcap c) = (a \sqcap b)\lfloor c$ *holds.*

Proof. The $\leq$-direction follow immediately from isotony of $\lfloor$ in both arguments. The other direction can be shown as follows. By Law (exc), Definition of $\lfloor$, Boolean algebra, Shunting, a pure, Lemma 6.10(c), a pure, Lemma A.6(A.3), (A.5), a pure, Lemma 6.10(c), Shunting, Boolean Algebra, and commutativity of $\cdot$, Lemma 5.6:

$$
\begin{aligned}
& (a \sqcap b)\lfloor c \leq (a \sqcap b)\lfloor(a \sqcap c) \\
\Leftrightarrow\ & \overline{(a \sqcap b)\lfloor(a \sqcap c)} \cdot c \leq \overline{a \sqcap b} \\
\Leftrightarrow\ & ((a \sqcap c)\backslash\overline{(a \sqcap b)}) \cdot c \leq \overline{a \sqcap b} \\
\Leftrightarrow\ & ((a \sqcap c)\backslash\overline{(a \sqcap b)}) \cdot c \leq \overline{a} + \overline{b} \\
\Leftrightarrow\ & a \sqcap (((a \sqcap c)\backslash\overline{(a \sqcap b)}) \cdot c) \leq \overline{b} \\
\Leftrightarrow\ & (a \sqcap ((a \sqcap c)\backslash\overline{(a \sqcap b)})) \cdot c \leq \overline{b} \\
\Leftrightarrow\ & (a \sqcap (c\backslash\overline{(a \sqcap b)})) \cdot c \leq \overline{b} \\
\Leftrightarrow\ & a \sqcap ((c\backslash\overline{(a \sqcap b)}) \cdot c) \leq \overline{b} \\
\Leftrightarrow\ & (c\backslash\overline{(a \sqcap b)}) \cdot c \leq \overline{a} + \overline{b} \\
\Leftrightarrow\ & (c\backslash\overline{(a \sqcap b)}) \cdot c \leq \overline{a \sqcap b} \\
\Leftrightarrow\ & \mathsf{true}
\end{aligned}
$$

□

Lemma A.8. *For arbitrary elements b, c and pure a the equation* $(\overline{a} \sqcap b)\lfloor(a \sqcap c) \leq 0$ *is valid. Therefore also* $(a \sqcap b)\lfloor(\overline{a} \sqcap c) \leq 0$ *holds.*

Proof. By Law (exc), Boolean algebra, pureness, isotony of +, and isotony of $\sqcap$:

$$
\begin{aligned}
& (\overline{a} \sqcap b)\lfloor(a \sqcap c) \leq 0 \\
\Leftrightarrow\ & \top \cdot (a \sqcap c) \leq \overline{\overline{a} \sqcap b} \\
\Leftrightarrow\ & \top \cdot (a \sqcap c) \leq a + \overline{b} \\
\Leftrightarrow\ & a \sqcap (\top \cdot c) \leq a + \overline{b} \\
\Leftarrow\ & a \sqcap (\top \cdot c) \leq a \\
\Leftrightarrow\ & \mathsf{true}
\end{aligned}
$$

□

Lemma A.9. *For arbitrary elements b, c and pure a the equation* $b\lfloor(a \sqcap c) = (a \sqcap b)\lfloor c$ *is valid.*

Proof. By Boolean algebra, distributivity of $\lfloor$, pureness, Lemma A.8, and pureness, Lemma A.7:

$$
\begin{aligned}
& b\lfloor(a \sqcap c) \\
=\ & ((a \sqcap b) + (\overline{a} \sqcap b))\lfloor(a \sqcap c) \\
=\ & ((a \sqcap b)\lfloor(a \sqcap c)) + ((\overline{a} \sqcap b)\lfloor(a \sqcap c)) \\
=\ & (a \sqcap b)\lfloor(a \sqcap c) \\
=\ & (a \sqcap b)\lfloor c
\end{aligned}
$$

□

Lemma A.10. *For arbitrary elements b, c and pure a the equation* $p \sqcap a\lfloor b = (p \sqcap a)\lfloor b$. *is valid.*

Proof. We first show the ≥-direction: By Law (exc), Boolean algebra, distributivity, isotony of +, isotony of ·, definition of $\lfloor$, and $\overline{a}$ is pure, Lemma 5.6:

$$
\begin{aligned}
& (a \sqcap b)\lfloor c \leq a \sqcap b\lfloor c \\
\Leftrightarrow\ & \overline{a \sqcap b\lfloor c} \cdot c \leq \overline{a \sqcap b} \\
\Leftrightarrow\ & (\overline{a} + \overline{b\lfloor c}) \cdot c \leq \overline{a} + \overline{b} \\
\Leftrightarrow\ & (\overline{a} \cdot c) + (\overline{b\lfloor c} \cdot c) \leq \overline{a} + \overline{b} \\
\Leftarrow\ & \overline{a} \cdot c \leq \overline{a} \quad \wedge \quad \overline{b\lfloor c} \cdot c \leq \overline{b} \\
\Leftarrow\ & \overline{a} \cdot \top \leq \overline{a} \quad \wedge \quad (\overline{b}/c) \cdot c \leq \overline{b} \\
\Leftrightarrow\ & true
\end{aligned}
$$

□

Corollary A.11. *For arbitrary elements b, c and pure a,* $a \sqcap b\lfloor c = (a \sqcap b)\lfloor c = b\lfloor(a \sqcap c) = (a \sqcap b)\lfloor(a \sqcap c)$ *holds.*

A.3. Precise Assertions

We now give some further useful properties of precise assertions which facilitates calculating with them.

Lemma A.12. *If b, c are precise and a is pure, then* $(a \sqcap b) + (\overline{a} \sqcap c)$ *is precise.*

Proof. We assume arbitrary elements $d, e \in S$. By distributivity of $*$, pureness of a and $\overline{a}$, distributivity of $\sqcap$, $a \sqcap \overline{a} = 0$, idempotence of $\sqcap$, preciseness of b and c, pureness of a and $\overline{a}$, and distributivity of $*$:

$$
\begin{aligned}
& ((a \sqcap b) + (\overline{a} \sqcap c)) \cdot d \sqcap ((a \sqcap b) + (\overline{a} \sqcap c)) \cdot e \\
=\ & ((a \sqcap p) \cdot d + (\overline{a} \sqcap c) \cdot d) \sqcap ((a \sqcap p) \cdot e + (\overline{a} \sqcap c) \cdot e) \\
=\ & (a \sqcap (b \cdot d) + \overline{a} \sqcap (c \cdot d)) \sqcap (a \sqcap (b \cdot e) + \overline{a} \sqcap (c \cdot e)) \\
=\ & a \sqcap (b \cdot d) \sqcap a \sqcap (b \cdot e) + \overline{a} \sqcap (c \cdot d) \sqcap a \sqcap (b \cdot e) + \\
& a \sqcap (b \cdot d) \sqcap \overline{a} \sqcap (c \cdot e) + \overline{a} \sqcap (c \cdot d) \sqcap \overline{a} \sqcap (c \cdot e) \\
=\ & a \sqcap (b \cdot d) \sqcap (b \cdot e) + \overline{a} \sqcap (c \cdot d) \sqcap (c \cdot e) \\
=\ & a \sqcap (b \cdot (d \sqcap e)) + \overline{a} \sqcap (c \cdot (d \sqcap e)) \\
=\ & ((a \sqcap b) \cdot (d \sqcap e)) + ((\overline{a} \sqcap c) \cdot (d \sqcap e)) \\
=\ & ((a \sqcap b) + (\overline{a} \sqcap c)) \cdot (d \sqcap e)
\end{aligned}
$$

□

Lemma A.13. *If a is precise then, for arbitrary b, c and d,* $((a \sqcap b) \cdot c) \sqcap (a \cdot d) = (a \sqcap b) \cdot (c \sqcap d)$

Proof. First we calculate

$$((p \sqcap a) \cdot (b \sqcap \overline{c})) \sqcap (p \cdot c) \leq (p \cdot (b \sqcap \overline{c})) \sqcap (p \cdot c) \ = \ p \cdot ((b \sqcap \overline{c}) \sqcap c) \ = \ 0 \qquad (*)$$

We prove the equation by showing each inequation separately. The ≤-direction can be shown as follows: By Boolean algebra, distributivity, by (∗), and definition of $\sqcap$:

$$
\begin{aligned}
& ((a \sqcap b) \cdot c) \sqcap (a \cdot d) \\
=\ & ((((a \sqcap b) \cdot (c \sqcap d)) + ((a \sqcap b) \cdot (c \sqcap \overline{d}))) \sqcap (a \cdot d) \\
=\ & ((((a \sqcap b) \cdot (c \sqcap d)) \sqcap (a \cdot d)) + (((a \sqcap b) \cdot (c \sqcap \overline{d})) \sqcap (a \cdot d)) \\
=\ & ((a \sqcap b) \cdot (c \sqcap d)) \sqcap (a \cdot d) \\
\leq\ & (a \sqcap b) \cdot (c \sqcap d)
\end{aligned}
$$

The converse inequation follows by isotony and the fact that $a \sqcap b$ is precise (cf. Lemma 6.16):

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

□

Corollary A.14. *If a or a' is precise, then $((a \sqcap b) \cdot a') \sqcap (a \cdot (a' \sqcap c)) = (a \sqcap b) \cdot (a' \sqcap c)$ for all b, c.*

This law characterises the interplay between $\cdot$ and $\sqcap$ w.r.t. precise assertions. Only the portions of the heaps that fit together with p and q remain in the intersection on the left-hand side of the equations.

A.4. Supported Assertions

Lemma A.15. *a is supported if $1 \leq a$.*

Proof. We calculate for arbitrary b and c

$$a \cdot b \sqcap a \cdot c \leq b \cdot \top \sqcap c \cdot \top = 1 \cdot (b \cdot \top \sqcap c \cdot \top) \leq a \cdot (b \cdot \top \sqcap c \cdot \top)$$

The first inequation follows from isotony of $\cdot$ and commutativity. Then by neutrality of 1 and assumption the conjecture holds. □

Corollary A.16. *$1, \top$ are supported. Moreover $(1 + a)$ is supported if a is.*

Lemma A.17. *a is supported iff $a \cdot \top$ is supported.*

Proof. The $\Rightarrow$-direction follows by definition and isotony:

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

The converse direction can be shown as follows:

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

This holds by isotony of $\cdot$, definition of supported elements, subdistributivity, commutativity and $\top = \top \cdot \top$. □

Corollary A.18. *If a is supported and b, c are intuitionistic then*

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

B. Deferred Proofs

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

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

$$\exists\, h' : \neg((dom(h') \cap dom(h) = \emptyset,\ s,h' \models q) \Rightarrow s,h' \cup h \models \neg p)$$
$$\Leftrightarrow \quad \{\!\mid \text{logic: } \neg(A \Rightarrow B) \Leftrightarrow (A \wedge \neg B) \mid\!\}$$
$$\exists\, h' : dom(h') \cap dom(h) = \emptyset,\ s,h' \models q,\ s,h' \cup h \not\models \neg p$$
$$\Leftrightarrow \quad \{\!\mid \text{logic} \mid\!\}$$
$$\exists\, h' : dom(h') \cap dom(h) = \emptyset,\ s,h' \models q,\ s,h' \cup h \models p$$
$$\Leftrightarrow \quad \{\!\mid \text{setting for } (\Rightarrow)\ \hat{h} =_{df} h' \cup h \text{ and for } (\Leftarrow)\ h' =_{df} \hat{h} - h \mid\!\}$$
$$\exists\, \hat{h} : h \subseteq \hat{h},\ s,\hat{h} - h \models q,\ s,\hat{h} \models p$$

□

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 (6) and (7). By neutrality of $\top$ for $\sqcap$, neutrality of 1 for $\cdot$, meet-distributivity (7) 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 (6):

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

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 7 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 6. 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 6 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 idempotency of $\sqcap$, the assumption, commutativity of $\cdot$ and again the assumption. □

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 8.6.

(a) Immediate from the definition and annihilation.

(b) Immediate from the definitions.

(c) Assume $s_1 \subseteq s_2$. By isotony of the domain operation we have $dom(s_1) \subseteq dom(s_2)$ and hence $s_2|_{dom(s_1)} = s_2$. Moreover, it is immediate from the definition of partial maps that also $s_2 = s_1|_{dom(s_2)}$.

□