UNIVERSITÄT AUGSBURG
Universitätsbibliothek

OPEN ACCESS
OPUS AUGSBURG

Reverse exchange for concurrency and local reasoning

Han-Hing Dang, Bernhard Möller

Angaben zur Veröffentlichung / Publication details:

Dang, Han-Hing, and Bernhard Möller. 2012. "Reverse exchange for concurrency and local reasoning." *Lecture Notes in Computer Science* 7342: 177–97.
https://doi.org/10.1007/978-3-642-31113-0_10.

Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:
Deutsches Urheberrecht
Weitere Informationen finden Sie unter: / For more information see:
https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publiz/

Reverse Exchange for Concurrency and Local Reasoning

Han-Hing Dang and Bernhard Möller

Institut für Informatik, Universität Augsburg, D-86159 Augsburg, Germany
{h.dang,moeller}@informatik.uni-augsburg.de

Abstract. Recent research has pointed out the importance of the inequational exchange law $(P * Q) ; (R * S) \leq (P ; R) * (Q ; S)$ for concurrent processes. In particular, it has been shown that this law is equivalent to validity of the concurrency rule for Hoare triples. Unfortunately, the law does not hold in the relationally based setting of algebraic separation logic. However, we show that under mild conditions the reverse inequation $(P ; R) * (Q ; S) \leq (P * Q) ; (R * S)$ still holds there. Separating conjunction $*$ in that calculus can be interpreted as true concurrency on disjointly accessed resources. From the reverse exchange law we derive slightly restricted but still reasonably useful variants of the concurrency rule. Moreover, using a corresponding definition of locality, we obtain also a variant of the frame rule. By this, the relational setting can also be applied for modular and concurrency reasoning. Finally, we present several variations of the approach to further interpret the results.

Keywords: True concurrency, relational semantics, Hoare logic, concurrent separation logic, locality, frame rule

1 Introduction

Algebraic techniques nowadays have found widespread application, especially in the area of program logics. In particular, *separation logic* [14] has proved to be very useful in the domain of modular and concurrency reasoning [1, 12] — although originally it was only developed to facilitate reasoning about shared mutable data structures. For this logic there are already different abstract approaches that capture corresponding calculi [2, 4]. Recent investigations on these topics resulted in a general algebraic structure called *Concurrent Kleene Algebra* [8]. A central concept of that algebra is that it allows easy soundness proofs of important rules like the concurrency and frame rules used in logics for concurrency and modular reasoning.

The *concurrency* and *frame* rules have the form

$$\frac{\{P_1\}\, Q_1 \,\{R_1\} \qquad \{P_2\}\, Q_2 \,\{R_2\}}{\{P_1 * P_2\}\, Q_1 * Q_2 \,\{R_1 * R_2\}}\ (\mathit{conc}) \qquad \frac{\{P\}\, Q \,\{R\}}{\{P * S\}\, Q \,\{R * S\}}\ (\mathit{frame})\ .$$

Here Q and Q_i denote programs while all other letters denote assertions. Now the *separating conjunction* $*$, as it is called in the literature, is used in the conclusion of these rules to ensure disjointness of states or resources characterised by assertions. When used on programs, such as the Q_i above, separating conjunction can be interpreted as concurrent execution of programs.

Interestingly, it has been shown in [7] that validity of the *exchange law*

$$(P_1 * P_2)\,;(Q_1 * Q_2) \;\leq\; (P_1\,;Q_1) * (P_2\,;Q_2)\ ,$$

for programs P_i and Q_i and validity of the concurrency rule are equivalent. An analogous connection holds between the *small exchange law*

$$(P_1 * P_2)\,;Q_1 \leq (P_1\,;Q_1) * P_2$$

and the frame rule. In these laws, semicolon denotes sequential composition, while $\leq$ denotes a partial ordering expressing refinement. The exchange laws can be seen as an abstract characterisation of the interplay between sequential and concurrent composition. Each of them expresses that the program on the right-hand side has fewer sequential dependences than the one on the left-hand side.

Several models for algebraic structures obeying those laws exist; details may be found in [8, 7]. However, they either do not model concurrency adequately enough or fail to satisfy other important laws in connection with nondeterministic choice. The purpose of the present paper is to investigate an extension of the relational model of separation logic presented in [4] by a generalised separating conjunction. As a relational structure it copes well with nondeterminacy; moreover, it allows the re-use of a large and well studied body of algebraic laws in connection with assertion logic. Surprisingly, it turns out that, although the model satisfies neither of the mentioned exchange laws, it validates an exchange law with the reversed refinement order. Moreover, this entails variants of the concurrency and frame rules with similarly simple soundness proofs as in the original Concurrent Kleene Algebra approach. Also, we establish an analogous equivalence between the concurrency rule and the reverse exchange law as in [7]. Hence, the relational calculus can be applied in reasoning about programs involving true concurrency and modularity. To underpin this further, we also study a number of variations of our main relational model and discuss their adequacy and usefulness.

2 Basic Definitions and Properties

We start by repeating some basic definitions from [4] and some direct consequences. Summarised, the central concept of this paper is a relational structure enriched by an operator that ensures disjointness of program states or executions. Notationally, we follow [4, 7].

Definition 2.1 A *separation algebra* is a partial commutative monoid $(\Sigma, \bullet, u)$; the elements of Σ are called *states* and denoted by $\sigma, \tau, \ldots$. The operator $\bullet$ denotes state combination and the *empty state* u is its unit. A partial commutative

monoid is given by a partial binary operation satisfying the unity, commutativity and associativity laws w.r.t. the equality that holds for two terms iff both are defined and equal or both are undefined. The induced *combinability* or *disjointness* relation $\#$ is defined by

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

As a concrete example one can instantiate the states to heaps. For this we set $\Sigma \;=_{df}\; \mathbb{N} \rightharpoonup \mathbb{N}$, i.e., the set of partial functions from naturals to naturals. Moreover $\bullet \;=_{df}\; \cup$ and $u \;=_{df}\; \emptyset$, the empty heap. A possible combinability relation for this domain would be $h_0 \# h_1 \;\Leftrightarrow_{df}\; dom(h_0) \cap dom(h_1) = \emptyset$ for heaps h_0, h_1. More concrete examples can be found in [2].

Definition 2.2 We assume a separation algebra $(\Sigma, \bullet, u)$. A *command* is a relation $P \subseteq \Sigma \times \Sigma$ between states. Relational composition is denoted by $;$. The command skip is the identity relation between states. A *test* is a subidentity, i.e., a command P with $P \subseteq \mathsf{skip}$. In the remainder we will denote tests by lower case letters $p, q, \ldots$. A particular test that characterises the empty state u is provided by $\mathsf{emp} \;=_{df}\; \{(u,u)\}$. Moreover, the domain of a command P, represented as a test, will also be denoted by $dom(P)$. It is characterised by the universal property

$$dom(P) \subseteq q \;\Leftrightarrow\; P \subseteq q\,;P \;.$$

In particular, $P \subseteq dom(P)\,;P$ and hence $P = dom(P)\,;P$.

Note that tests form a Boolean algebra with skip as its greatest and $\emptyset$ as its least element. Moreover, on tests $\cup$ coincides with join and $;$ with meet. In particular, tests are idempotent and commute under composition, i.e., $p\,;p = p$ and $p\,;q = q\,;p$.

Next we give some definitions to introduce separation relationally. Separating conjunction of commands can be interpreted as their parallel execution on disjoint portions of the state or, in the special case of tests, by asserting disjointness of certain resources.

Definition 2.3 We will frequently work with pairs of commands. Union, inclusion and composition of such pairs are defined componentwise. The *Cartesian product* $P \times Q$ of commands P, Q is given by

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

We assume that $;$ binds tighter than $\times$. It is clear that $\mathsf{skip} \times \mathsf{skip}$ is the identity of composition. Note that $\times$ and $;$ satisfy an *equational* exchange law:

$$P\,;Q \;\times\; R\,;S \;=\; (P \times R) \;;\; (Q \times S) \;. \tag{1}$$

Definition 2.4 *Tests* in the set of product relations are again subidentities; as before they are idempotent and commute under $;$. The Cartesian product of tests is a test again. However, there are other tests, such as the *combinability check* $\#$ [4], on pairs of states:

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

Definition 2.5 We define *split* $\lhd$ and its converse *join* $\rhd$ as in [4] by

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

Lemma 2.6 *We have* $\# = \rhd \,;\, \lhd \cap \mathsf{skip}$ *and hence* $\# \subseteq \rhd \,;\, \lhd$. *Moreover* $\# \,;\, \rhd = \rhd$ *and symmetrically* $\lhd \,;\, \# = \lhd$.

One might conjecture $\mathsf{skip} \times \mathsf{skip} \subseteq \rhd \,;\, \lhd$ at first. However, this is not true, since the left-hand side of the inequation also considers incombinable pairs of states which are not included in the right-hand side according to Lemma 2.6. We will see in the next section that this fact requires us to impose an additional compatibility condition on commands for proving soundness of the reverse exchange law, i.e., the exchange law with the inequation reversed.

Definition 2.7 Generalising [4], we define the *parallel composition (separating conjunction)* of commands as $P * Q \;=_{df}\; \lhd \,;\, (P \times Q) \,;\, \rhd$.

By this definition, a relation $\sigma \,(P * Q)\, \tau$ holds iff σ can be split as $\sigma = \sigma_1 \bullet \sigma_2$ with disjoint parts σ_1, σ_2 on which P and Q can act and produce results τ_1, τ_2 that are again disjoint and combine to $\tau = \tau_1 \bullet \tau_2$. Hence $P * Q$ may be viewed as a program that runs P and Q in a truly concurrent fashion as indivisible actions, at least conceptually. An actual implementation may still do this in an interleaved or even truly concurrent fashion, as long as non-interference is guaranteed. We note that for tests p, q the command $p * q$ is a test again. Moreover, $*$ is associative and commutative and emp is its unit. Finally, there is the following interplay between $*$ and the domain operator.

Lemma 2.8 *For commands* P, Q *we have* $dom(P * Q) \subseteq dom(P) * dom(Q)$.

The proof can be found in the Appendix.

3 Compatibility and the Reverse Exchange Law

According to the general results in [7], soundness of the concurrency rule in the relational setting would follow immediately if the exchange law

$$(P * Q) \,;\, (R * S) \;\subseteq\; (P \,;\, R) * (Q \,;\, S)$$

with relational inclusion $\subseteq$ as the refinement order were to hold there.

However, as also shown in [7], we have

Lemma 3.1 *The exchange law implies* $\mathsf{skip} \subseteq \mathsf{emp}$.

On the other hand, by definition $\mathsf{emp} \subseteq \mathsf{skip}$, so that by antisymmetry skip and emp would be equal, a contradiction.

Therefore the exchange law is not valid in the relational setting. Instead, and surprisingly, we were only able to show soundness of a restricted variant of the exchange law with the reversed inclusion order. The proof uses a restriction on pairs (P, Q) of commands: when P and Q start from combinable pairs of input states they produce combinable pairs of output states, or the other way around. This is formalised as follows.

Definition 3.2 Commands P and Q are *forward compatible* iff

$$\# \,;\, (P \times Q) \;\subseteq\; (P \times Q) \,;\, \# \;.$$

Symmetrically P and Q are *backward compatible* iff $(P \times Q);\# \subseteq \#;(P \times Q)$. Two commands are called *compatible* iff they are forward and backward compatible, i.e., $\# \,;\, (P \times Q) = (P \times Q) \,;\, \#$.

Again for a more concrete example of such commands we recapitulate our instantiation of states to heaps described in Section 2. Intuitively two compatible commands would work on disjoint portions of a heap, e.g. by only altering disjoint ranges of heap cells. Hence they ensure disjointness before and after their execution. In the following we list a few consequences of Definition 3.2.

Lemma 3.3 *All test commands are compatible with each other.*

Proof. For test commands p, q the relation $p \times q$ is a test in the algebra of relations on pairs. Since $\#$ is a test there, too, they commute, which means forward and backward compatibility of p and q. □

Since the combinability check $\#$ is a test on pairs of commands, it induces some useful closure properties.

Corollary 3.4 *If P, Q are forward compatible and $R \subseteq P$ then also R, Q are forward compatible. This result also holds for backward compatibility, hence compatibility is downward closed, too.*

Proof. We assume ; binds tighter than $\cap$. Now we show the following more general result: Let C, D, E be relations on pairs of states such that C is a test. If C is an invariant of D, i.e., $C \,;\, D \subseteq D \,;\, C$, and $E \subseteq D$ then C is also an invariant of E. For this we calculate

$$\begin{aligned} C \,;\, E \;&=\; C \,;\, (D \cap E) \;=\; C \,;\, D \cap C \,;\, E \;\subseteq\; D \,;\, C \cap C \,;\, E \;= \\ &\quad D \cap C \,;\, E \,;\, C \;=\; C \,;\, E \,;\, C \;\subseteq\; E \,;\, C \;. \end{aligned}$$

The last but one step follows since C is a test. A proof can e.g. be found in [10]. Now the main claim follows by setting $C = \#$, $D = P \times Q$ and $E = R \times Q$. □

We note that this proof extends to arbitrary test semirings.

Corollary 3.5 *Let P, Q and R, S be forward compatible. Then also $P \,;\, R$ and $Q \,;\, S$ are forward compatible. Again the same holds for backward compatibility.*

Proof.

$$\begin{aligned} &\# \,;\, (P \,;\, R \times Q \,;\, S) = \# \,;\, (P \times Q) \,;\, (R \times S) \subseteq (P \times Q) \,;\, \# \,;\, (R \times S) \subseteq \\ &\quad (P \times Q) \,;\, (R \times S) \,;\, \# = (P \,;\, R \times Q \,;\, S) \,;\, \# \;. \end{aligned}$$

□

Now we are ready for the central result mentioned at the beginning of this section. For forward or backward compatible commands we are able to prove soundness of a variant of the reverse exchange law using the inclusion order. Note that validity of the exchange law in [7] is proved for arbitrary predicate transformers. In the next section we will see that specialising validity of the reversed law to compatible commands does not impose any restrictions on our treatment.

Lemma 3.6 (Reverse Exchange) *If P, Q are forward compatible or R, S are backward compatible then*

$$(P\,;R) * (Q\,;S) \;\subseteq\; (P * Q)\,;(R * S)\ .$$

In particular, if P, R or Q, S are tests the inequation holds.

Proof. We assume that P and Q are forward compatible.

$$\begin{array}{ll} & (P\,;R) * (Q\,;S) \\ = & \quad \{\![\text{ definition of } * \ ]\!\} \\ & \lhd\,;(P\,;R \times Q\,;S)\,;\rhd \\ = & \quad \{\![\ ;/\times \text{ exchange (1) }]\!\} \\ & \lhd\,;(P \times Q)\,;(R \times S)\,;\rhd \\ = & \quad \{\![\text{ Lemma 2.6 }]\!\} \\ & \lhd\,;\#\,;(P \times Q)\,;(R \times S)\,;\rhd \\ \subseteq & \quad \{\![\text{ forward compatibility }]\!\} \\ & \lhd\,;(P \times Q)\,;\#\,;(R \times S)\,;\rhd \\ \subseteq & \quad \{\![\text{ Lemma 2.6 }]\!\} \\ & \lhd\,;(P \times Q)\,;\rhd\,;\lhd\,;(R \times S)\,;\rhd \\ = & \quad \{\![\text{ definition of } * \ ]\!\} \\ & (P * Q)\,;(R * S)\ . \end{array}$$

The proof for backward compatibility and R, S is symmetric. □

The reverse exchange law expresses an increase in granularity: while in the left-hand side program $P\,;R$ and $Q\,;S$ are treated as indivisible, they are split in the right-hand side program, at the expense of a "global" synchronisation point marked by the semicolon (which is the reason for the compatibility requirement).

4 Hoare Triples and the Concurrency Rule

To prepare our variant of the concurrency rule we now define Hoare triples in our setting.

Definition 4.1 For general commands P, Q, R, the *general Hoare triple* [7] is defined as

$$P\,\{Q\}\,R \;\Leftrightarrow_{df}\; P\,;Q \subseteq R\ .$$

For tests p, r and arbitrary command Q the *standard* Hoare triple [9] $\{p\}\,Q\,\{r\}$ is defined by

$$\{p\}\,Q\,\{r\} \quad \Leftrightarrow_{df} \quad p\,;Q \subseteq Q\,;r\ .$$

General Hoare triples also admit programs as assertions, in contrast to the standard ones that only allow tests to denote pre- and postconditions. As shown in [4], we have the relationship

$$\{p\}\,Q\,\{r\} \quad \Leftrightarrow \quad (U\,;p)\,\mathbf{\{}Q\mathbf{\}}\,(U\,;r)$$

where U denotes the universal relation. Hence our results for standard Hoare triples can be immediately translated into ones for general triples. The composition $U\,;p$ maps a test p to a command that makes no assumption about its starting state. Intuitively, starting from an arbitrary state that command will end up in one satisfying p. Trivially, a symmetrical command $p\,;U$ makes no restriction on the ending state or codomain.

Next we turn to the definition of properties and conditions that will allow us to prove variants of the concurrency rule for standard Hoare triples using the reverse exchange law.

The following observation is trivial, but useful for our first variant of the concurrency rule.

Corollary 4.2 $\{p\}\,Q\,\{r\} \Leftrightarrow \{p\}\,p\,;Q\,\{r\}$.

Proof. By idempotence of test p,

$$\{p\}\,Q\,\{r\} \quad \Leftrightarrow \quad p\,;Q \subseteq Q\,;r \quad \Leftrightarrow \quad p\,;p\,;Q \subseteq Q\,;r \quad \Leftrightarrow \quad \{p\}\,p\,;Q\,\{r\}\ .$$

□

The command $p\,;Q$ can be viewed as asserting the precondition p before executing Q.

The condition we need for our first variant of the concurrency rule is that the commands Q_i enforce the preconditions p_i in that all their starting states satisfy the respective p_i. Algebraically this is expressed by the formula $Q_i \subseteq p_i\,;Q_i$, which is equivalent to $Q_i = p_i\,;Q_i$ and to $dom(Q_i) \subseteq p_i$. This restriction is not essential: by Cor. 4.2 and the idempotence of tests we can always replace Q_i by $Q_i' =_{df} p_i\,;Q_i$ to achieve this.

Lemma 4.3 (Concurrency Rule I)

$$\frac{\{p_1\}\,Q_1\,\{r_1\} \quad \{p_2\}\,Q_2\,\{r_2\} \quad dom(Q_1) \subseteq p_1 \quad dom(Q_2) \subseteq p_2}{\{p_1 * p_2\}\,Q_1 * Q_2\,\{r_1 * r_2\}}$$

Proof. $(p_1 * p_2)\,;(Q_1 * Q_2)$

$\subseteq$ $\quad$ {[$p_1 * p_2$ a test, hence a subidentity]}

$Q_1 * Q_2$

$$
\begin{array}{ll}
\subseteq & \quad \{\![\ Q_i \ \subseteq\ p_i \,;\, Q_i\ ]\!\} \\
& (p_1 \,;\, Q_1) * (p_2 \,;\, Q_2) \\
\subseteq & \quad \{\![\ \text{by } \{p_i\}\, Q_i\, \{r_i\}\ ]\!\} \\
& (Q_1 \,;\, r_1) * (Q_2 \,;\, r_2) \\
\subseteq & \quad \{\![\ \text{reverse exchange law (Lemma 3.6), since } r_1 \text{ and } r_2 \text{ are tests} \\
& \qquad \text{and hence compatible by Lemma 3.3}\ ]\!\} \\
& (Q_1 * Q_2) \,;\, (r_1 * r_2)\ .
\end{array}
$$

□

Note that compatibility of the commands Q_i is not needed.

The proof might suggest that the preconditions p_i do not really matter, since they are discarded in the first step. However, they are re-introduced in the next step and hence indeed *do* matter.

A brief discussion of the relevance and use of this rule can be found at the end of the next section.

To round off this section, we prove the following result which, together with Lemma 4.3, provides the analogue of the equivalence between the full exchange law and the concurrency rule shown in [7].

Lemma 4.4 *Validity of Concurrency Rule I implies a special case of the reverse exchange law: for arbitrary commands P_i and tests r_i,*

$$(P_1 \,;\, r_1) * (P_2 \,;\, r_2)\ \subseteq\ (P_1 * P_2) \,;\, (r_1 * r_2)\ .$$

Proof. In Concurrency Rule I we set $Q_i = P_i \,;\, r_i$ and $p_i = \mathit{dom}(Q_i)$. By this the premise of the rule becomes valid since

$$\{\mathit{dom}(Q_i)\}\, Q_i\, \{r_i\} \ \ \Leftrightarrow\ \ \mathit{dom}(Q_i) \,;\, Q_i\ \subseteq\ Q_i \,;\, r_i \ \ \Leftrightarrow\ \ Q_i\ \subseteq\ Q_i \,;\, r_i$$

and $Q_i \,;\, r_i = (P_i \,;\, r_i) \,;\, r_i = P_i \,;\, r_i = Q_i$. Hence, by the conclusion of the rule we have

$$(\mathit{dom}(Q_1) * \mathit{dom}(Q_1)) \,;\, (Q_1 * Q_2)\ \subseteq\ (Q_1 * Q_2) \,;\, (r_1 * r_2)\ . \qquad (\dagger)$$

Now we calculate:

$$
\begin{array}{ll}
& (P_1 \,;\, r_1) * (P_2 \,;\, r_2) \\
= & \quad \{\![\ \text{definitions of } Q_i\ ]\!\} \\
& Q_1 * Q_2 \\
= & \quad \{\![\ \text{property of domain}\ ]\!\} \\
& \mathit{dom}(Q_1 * Q_2) \,;\, (Q_1 * Q_2) \\
\subseteq & \quad \{\![\ \text{by Lemma 2.8}\ ]\!\} \\
& (\mathit{dom}(Q_1) * \mathit{dom}(Q_2)) \,;\, (Q_1 * Q_2) \\
\subseteq & \quad \{\![\ \text{by } (\dagger)\ ]\!\} \\
& (Q_1 * Q_2) \,;\, (r_1 * r_2) \\
= & \quad \{\![\ \text{definitions of } Q_i\ ]\!\} \\
& ((P_1 \,;\, r_1) * (P_2 \,;\, r_2)) \,;\, (r_1 * r_2) \\
\subseteq & \quad \{\![\ \text{by } r_i \subseteq \mathsf{skip}\ ]\!\} \\
& (P_1 * P_2) \,;\, (r_1 * r_2)\ .
\end{array}
$$

□

We conclude by showing that the symmetric special case already follows without assuming reverse exchange or Concurrency Rule I or even mentioning the notion of compatibility.

Lemma 4.5 *For arbitrary commands Q_i and tests p_i,*

$$(p_1 \,;\, Q_1) * (p_2 \,;\, Q_2) \;\subseteq\; (p_1 * p_2) \,;\, (Q_1 * Q_2) \;.$$

Proof. We calculate:

$$\begin{array}{ll} & (p_1 \,;\, Q_1) * (p_2 \,;\, Q_2) \\ = & \quad \{\![\text{ property of domain }]\!\} \\ & dom((p_1 \,;\, Q_1) * (p_2 \,;\, Q_2)) \,;\, ((p_1 \,;\, Q_1) * (p_2 \,;\, Q_2)) \\ \subseteq & \quad \{\![\text{ by Lemma 2.8 }]\!\} \\ & (dom(p_1 \,;\, Q_1) * dom(p_2 \,;\, Q_2)) \,;\, ((p_1 \,;\, Q_1) * (p_2 \,;\, Q_2)) \\ = & \quad \{\![\text{ property of domain }]\!\} \\ & ((p_1 \,;\, dom(Q_1)) * (p_2 \,;\, dom(Q_2))) \,;\, ((p_1 \,;\, Q_1) * (p_2 \,;\, Q_2)) \\ \subseteq & \quad \{\![\text{ by } dom(Q_i) \subseteq \mathsf{skip} \text{ and } p_i \subseteq \mathsf{skip} \,]\!\} \\ & (p_1 * p_2) \,;\, (Q_1 * Q_2) \;. \end{array}$$

□

Since Lemma 2.8 holds analogously for the codomain operator, this proof could also be adapted to a direct proof of the property in Lemma 4.4.

Finally, the special case of reverse exchange mentioned in Lemma 4.5 in turn implies Lemma 2.8:

$$\begin{array}{ll} & dom(P * Q) \;\subseteq\; dom(P) * dom(Q) \\ \Leftrightarrow & \quad \{\![\text{ universal characterisation of domain }]\!\} \\ & P * Q \;\subseteq\; (dom(P) * dom(Q)) \,;\, (P * Q) \\ \Leftarrow & \quad \{\![\text{ special case of reverse exchange }]\!\} \\ & P * Q \;\subseteq\; (dom(P) \,;\, P) * (dom(Q) \,;\, Q) \\ \Leftrightarrow & \quad \{\![\text{ property of domain }]\!\} \\ & P * Q \;\subseteq\; P * Q \\ \Leftrightarrow & \quad \{\![\text{ reflexivity of } \subseteq \,]\!\} \\ & \mathsf{TRUE} \;. \end{array}$$

5 Another Concurrency Rule

We now present a second variant of the concurrency rule. Its main idea is inspired by a more special property given in [4], which will also figure again in the next section.

Definition 5.1 Two commands Q_1, Q_2 have the *concurrency property* iff

$$(dom(Q_1) \times dom(Q_2)) \,;\, \rhd \,;\, Q_1 * Q_2 \;\subseteq\; (Q_1 \times Q_2) \,;\, \rhd \;. \qquad (2)$$

This property is "angelic" in the sense that whenever two combinable states σ_1 and σ_2 provide enough resource for the execution of the programs Q_i then each Q_i will be able to acquire its needed resource from the joined state $\sigma_1 \bullet \sigma_2$.

To see that this is not always possible, we present a concrete example again in the heap model (mentioned in Section 2) with two commands that do not satisfy the concurrency property. Consider

$$Q_1 =_{df} ([1] := 1) \cup ([2] =: 1) \quad \text{and} \quad Q_2 =_{df} ([1] := 2) \cup ([2] := 2)$$

where $[x] := y$ represents a command that changes the content of the heap cell x to y and $\cup$ denotes non-deterministic choice. Clearly, the commands show interference with each other since both may access the same heap locations.

To see that Q_1 and Q_2 do not satisfy the concurrency property, first note $dom(Q_1) = dom(Q_2) = \{(h, h) : 1 \in dom(h) \vee 2 \in dom(h)\}$. Next, we consider heaps $h_1 = \{(1, 0)\}$ and $h_2 = \{(2, 0)\}$ with $(h_i, h_i) \in dom(Q_i)$. Thus, using $h = h_1 \bullet h_2$ and $h_1 \# h_2$, we have $(h, h) \in dom(Q_1 * Q_2)$. Moreover, a possible execution of $Q_1 * Q_2$ is $(h, \{(1, 1), (2, 2)\})$. Hence, $((h_1, h_2), \{(1, 1), (2, 2)\})$ is included in the left hand side of the instantiated concurrency property but not in the right hand side since we only have $((h_1, h_2), \{(1, 2), (2, 1)\})$ there.

We are now interested in relating concurrency property to the exchange law for concurrent processes. It turns out that the property is sufficient for validating a special case of the exchange law which we use to prove soundness of the concurrency rule.

Definition 5.2 We call two commands Q_1, Q_2 *pre-concurrent* iff we have for all tests p_1, p_2

$$p_1 \subseteq dom(Q_1) \wedge p_2 \subseteq dom(Q_2) \Rightarrow (p_1 * p_2) ; (Q_1 * Q_2) \subseteq (p_1 ; Q_1) * (p_2 ; Q_2) .$$

Lemma 5.3 *If commands Q_1 and Q_2 have the concurrency property then they are pre-concurrent.*

Proof. Assume $p_i \subseteq dom(Q_i)$. Then

$$\begin{array}{ll}
 & (p_1 * p_2) ; (Q_1 * Q_2) \\
= & \quad \{\![\text{ definition of } *, \ p_i \subseteq dom(Q_i) \text{ for } i = 1, 2 \ ]\!\} \\
 & \lhd ; (p_1 ; dom(Q_1) \times p_2 ; dom(Q_2)) ; \rhd ; (Q_1 * Q_2) \\
= & \quad \{\![\ ; / \times \text{ exchange (1) }]\!\} \\
 & \lhd ; (p_1 \times p_2) ; (dom(Q_1) \times dom(Q_2)) ; \rhd ; (Q_1 * Q_2) \\
\subseteq & \quad \{\![\text{ concurrency property (2) }]\!\} \\
 & \lhd ; (p_1 \times p_2) ; (Q_1 \times Q_2) ; \rhd \\
= & \quad \{\![\ ; / \times \text{ exchange (1) }]\!\} \\
 & \lhd ; (p_1 ; Q_1 \times p_2 ; Q_2) ; \rhd \\
= & \quad \{\![\text{ definition of } *]\!\} \\
 & (p_1 ; Q_1) * (p_2 ; Q_2) .
\end{array}$$

□

Interestingly, this special case of the exchange law already suffices to prove our second variant of the concurrency rule although the complete exchange law is needed for the concurrency rule in [7]; note also that the inclusion relations between the preconditions and the domains of the commands are the reverses of the ones in Lemma 4.3.

Lemma 5.4 (Concurrency Rule II) *Let Q_1 and Q_2 have the concurrency property. Then*

$$\frac{\{p_1\}\, Q_1\, \{r_1\} \quad \{p_2\}\, Q_2\, \{r_2\} \quad p_1 \subseteq \mathit{dom}(Q_1) \quad p_2 \subseteq \mathit{dom}(Q_2)}{\{p_1 * p_2\}\, Q_1 * Q_2\, \{\, r_1 * r_2\}} \,.$$

Proof.
$$\begin{array}{ll} & (p_1 * p_2)\,;(Q_1 * Q_2) \\ \subseteq & \quad \{\![\text{ Lemma 5.3 }]\!\} \\ & (p_1\,; Q_1) * (p_2\,; Q_2) \\ \subseteq & \quad \{\![\{p_i\}\, Q_i\, \{r_i\} \,]\!\} \\ & (Q_1\,; r_1) * (Q_2\,; r_2) \\ \subseteq & \quad \{\![\text{ reverse exchange law (Lemma 3.6),} \\ & \qquad \text{since } r_1, r_2 \text{ as tests are compatible }]\!\} \\ & (Q_1 * Q_2)\,;(r_1 * r_2)\ . \end{array}$$

□

In summary, we have presented two variations of the concurrency rule in our relational calculus. An advantage of Lemma 4.3 is that it only requires that the domains of the commands Q_i coincide with the respective preconditions, but needs no connection between the Q_i. Contrarily, Lemma 5.4 is more liberal w.r.t. the preconditions but requires the Q_i to have the concurrency property.

Still, usually at least one of the concurrency rules can be applied. Consider, for instance, parallel mergesort `ms` [11]:

$$\frac{\begin{array}{c}\{\mathit{array}(a,i,m)\}\, \mathtt{ms}(a,i,m)\, \{\mathit{sorted}(a,i,m)\} \\ \{\mathit{array}(a,m+1,j)\}\, \mathtt{ms}(a,m+1,j)\, \{\mathit{sorted}(a,m+1,j)\}\end{array}}{\begin{array}{c}\{\mathit{array}(a,i,m) * \mathit{array}(a,m+1,j)\,\} \\ \mathtt{ms}(a,i,m) * \mathtt{ms}(a,m+1,j) \\ \{\,\mathit{sorted}(a,i,m) * \mathit{sorted}(a,m+1,j)\}\end{array}}$$

where $\mathit{array}(a,i,j)$, assuming $i < j$, asserts that the store range with addresses $a{+}i$ to $a{+}j$ forms an array, i.e., contains elements of equal type, and $\mathit{sorted}(a,i,j)$ ensures that the content in that range is sorted. It is easy to define $\mathtt{ms}(a,i,m)$ in such a way that its domain is characterised by $\mathit{array}(a,i,m)$. Moreover, in any reasonable implementation the commands $\mathtt{ms}(a,i,m)$ and $\mathtt{ms}(a,m+1,j)$ even satisfy the concurrency property.

Thus, the concurrency rules in our relational calculus represent a feasible approach to enable reasoning about disjoint true concurrency.

6 Locality and the Frame Rule

We now turn to another important proof rule for modular reasoning. Validity of that rule is based on the concept of *locality* which describes the behaviour of programs that only access certain subsets of the available resources. Hence locality allows embedding a program into a larger context so that any resource not accessed by that program remains unchanged. This fact is expressed by the *frame rule*

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

To obtain a suitable version of the frame rule in our relational calculus we first remind the reader of a central result of [7]. A predicate transformer F in that model is called *local* iff it satisfies the equation

$$F * \mathsf{skip} = F\,.$$

This equation characterises exactly the above-mentioned modularity concept. Each execution of the program F can be replaced by one that only operates on the necessary and possible smaller part of the state while the rest of it remains unchanged (abstractly denoted by the program skip). In the following we derive the same compact characterisation for commands.

First remember that emp is the unit of $*$ and $\mathsf{emp} \subseteq \mathsf{skip}$.

Lemma 6.1 *For arbitrary commands Q we have $Q \subseteq Q * \mathsf{skip}$.*

Proof. $Q = Q * \mathsf{emp} \subseteq Q * \mathsf{skip}$. □

To get the other inclusion, i.e., $Q * \mathsf{skip} \subseteq Q$, we need an additional assumption about Q. Surprisingly, this inequation can be derived from a property given in [4] which was called test preservation and used there to prove soundness of the frame rule.

Definition 6.2 A command Q *preserves* a test r iff

$$\lhd\,;\,(Q \times r)\,;\#\ \subseteq\ Q\,;\lhd\,;\,(\mathsf{skip} \times r)\ . \tag{3}$$

We call a command Q *local* iff Q preserves all tests.

Formula (3) means that when running Q on a part of the state such that the remainder of the state satisfies r one might also run Q first on the complete state and will still find an r-part in the result state.

Preservation of r by Q is an abstraction of the property that Q does not modify the free variables of r; a more refined version of this definition was given in [4] and another one was studied in [3]. Locality as preservation of all tests, does not seem very realistic in that domain. Nevertheless, it turns out to be equivalent to the algebraic formulation of [7]:

Theorem 6.3 *A command Q is local iff $Q * \mathsf{skip} \subseteq Q$.*

The proof can be found in the Appendix.

By Theorem 6.3 we may, as in [7], define local commands as fixpoints of the localising operation $(\ \cdot\ ) * \mathsf{skip}$.

With this definition of locality we now prove our variant of the frame rule. We take a similar direction as in Section 4 by defining sufficient conditions needed for a soundness proof. First notice that in [7] the compact definition of locality and the full exchange law are used to get validity of the small exchange law for local predicate transformers. The small exchange law reads

$$(P * Q)\,;R \ \leq\ (P\,;R) * Q$$

for programs P, Q, R and the refinement order $\leq$ of a locality bimonoid. Moreover this law is equivalent to soundness of the frame rule in such a structure. In our approach locality has the same definition, but the small exchange law does not hold. Therefore again a further sufficient condition is needed to simulate the relevant part of the small exchange law.

Definition 6.4 Call command Q *pre-framed* iff for all test commands p, r

$$p \subseteq \mathit{dom}(Q) \quad \Rightarrow \quad (p * r)\,;Q \ \subseteq\ (p\,;Q) * r\ .$$

The premise $p \subseteq \mathit{dom}(Q)$ informally states that p already ensures enough resources for the execution of Q. We will see that this is a sufficient condition to prove soundness of the frame rule. Notice that the conclusion is only a special case of the small exchange law.

In [4] we used a relational variant of the frame property to prove the frame rule. We will use it in this paper in a simplified form.

Definition 6.5 A command Q has the *frame property* iff

$$(\mathit{dom}(Q) \times \mathsf{skip})\,;\rhd\,;Q \ \subseteq\ (Q \times \mathsf{skip})\,;\rhd\ .$$

This property can be derived as a special case of the concurrency property by setting $Q_2 = \mathsf{skip}$ and assuming locality for Q_1, i.e., $Q_1 * \mathsf{skip} = Q_1$. Again, this property is sufficient for pre-framedness.

Lemma 6.6 *If Q has the frame property then Q is pre-framed.*

Proof. Assume $p \subseteq \mathit{dom}(Q)$. Then

$$\begin{array}{ll} & (p * r)\,;Q \\ = & \quad \{\!\![\ \text{assumption}\ ]\!\!\} \\ & ((p\,;\mathit{dom}(Q)) * r)\,;Q \\ = & \quad \{\!\![\ \text{definition of } * \text{ and } ;/\times \text{ exchange (1)}\ ]\!\!\} \\ & \lhd\,;(p \times r)\,;(\mathit{dom}(Q) \times \mathsf{skip})\,;\rhd\,;Q \\ \subseteq & \quad \{\!\![\ \text{frame property}\ ]\!\!\} \\ & \lhd\,;(p \times r)\,;(Q \times \mathsf{skip})\,;\rhd \\ = & \quad \{\!\![\ ;/\times \text{ exchange (1) and definition of } *\ ]\!\!\} \\ & (p\,;Q) * r\ . \end{array}$$

□

Now we can easily prove the frame rule.

Lemma 6.7 (Frame Rule) *Let Q be local and have the frame property. Moreover, assume $p \subseteq dom(Q)$. Then*

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

Proof.
$$\begin{array}{ll} & (p * r)\,;Q \\ \subseteq & \quad \{\!\![\ p \subseteq dom(Q) \text{ and } Q \text{ pre-framed by Lemma 6.6 }]\!\!\} \\ & (p\,;Q) * r \\ \subseteq & \quad \{\!\![\ \text{by } \{p\}\, Q\, \{q\} \ ]\!\!\} \\ & (Q\,;q) * r \\ \subseteq & \quad \{\!\![\ r = \mathsf{skip}\,;r \text{ and reverse exchange law (Lemma 3.6),} \\ & \qquad \text{since } r, \mathsf{skip} \text{ as tests are compatible }]\!\!\} \\ & (Q * \mathsf{skip})\,;(q * r) \\ \subseteq & \quad \{\!\![\ Q \text{ local }]\!\!\} \\ & Q\,;(q * r)\ . \end{array}$$

□

Next, we compare the structure of our proofs with the corresponding ones in [7] to point out the main differences. Since the small exchange law is not valid in our relational setting (not even for local commands), it was necessary to constrain the set of commands considered in the frame rule by an additional assumption. It turned out in [4] that the relational version of the frame property was an adequate substitute. We have shown here that this property also relates to pre-framed commands which already ensure the relevant part of the small exchange law. Structurally, the proof of this frame rule becomes as simple as the one for the predicate transformer approach in [7]. Due to the angelic character of relations, the rule itself needs the additional premise $p \subseteq dom(Q)$.

As a further remark, the approach of [7] requires special functions for the semantics of Hoare triples. They are called *best predicate transformers* and are used as an adequate substitute for assertions. Intuitively these functions simulate the allocation of resources that are characterised by pre- and postconditions. In our calculus this can be handled by composing tests with the universal relation. However, since we have a non-trivial test algebra in the relational setting, tests by themselves already admit a suitable representation of pre- and postconditions.

7 Dual Correctness Triples

The previous sections presented an approach to include the concurrency and frame rules in the given relational approach to separation logic [4] by requiring additional assumptions and hence restricting the proof rules. In this section we

present some further applications for the reverse exchange law. We link it with the definitions of triples dual to the ones of Hoare. By this we will again see that the concurrency and frame rules can be easily derived using the reverse exchange law.

Definition 7.1 As in [6], for commands P, Q, R we define *Plotkin* triples by

$$\langle P, Q\rangle \to R \;\; \Leftrightarrow_{df} \;\; R \subseteq P\,;Q$$

and dual partial correctness triples by

$$P\,[Q]\,R \;\; \Leftrightarrow_{df} \;\; P \subseteq Q\,;R\;.$$

Intuitively, the former characterise possible states satisfying the postcondition R after the execution of Q starting from P while the latter symmetrically describes possible starting states of P that end in R after the execution of Q. The notation is inspired by Plotkin's structural operational semantics [13] in which $\langle C, s\rangle \to t$ means that evaluation of term C starting in state s may lead to term t. According to [6], dual partial correctness triples can e.g. be used as a method for the generation of test cases. Assuming R represents erroneous final states of Q then P characterises some conditions that will lead to such error situations. Plotkin triples can used for a dual application.

Using the relationship between tests and commands given in Section 4, in our calculus the dual partial correctness triples transform into

$$(p\,;U)\,[Q]\,(q\,;U) \;\Leftrightarrow\; p\,;U \subseteq Q\,;(q\,;U) \;\Leftrightarrow\; p \subseteq (Q\,;q)\,;U \;\Leftrightarrow\; p \subseteq \mathit{dom}(Q\,;q)$$

and, symmetrically, Plotkin triples into

$$\langle U\,;p\,,Q\rangle \to U\,;q \;\Leftrightarrow\; U\,;q \subseteq (U\,;p)\,;Q \;\Leftrightarrow\; q \subseteq U\,;(p\,;Q) \;\Leftrightarrow\; q \subseteq \mathit{cod}(p\,;Q)\;.$$

We concentrate on dual partial correctness triples and use the abbreviation $p\;[\![\,Q\,]\!]\;q \;\Leftrightarrow_{df}\; (p\,;U)\,[Q]\,(q\,;U) \;\Leftrightarrow\; p \subseteq \mathit{dom}(Q\,;q)$. Dual results hold for Plotkin triples.

The central interest of these new triples lies in the following result.

Lemma 7.2 *The concurrency rule for dual partial correctness or Plotkin triples holds iff the reverse exchange law holds.*

A proof for this lemma can be derived dually to [7]. Unfortunately, in our setting the reverse exchange law does not hold unconditionally. However, we will see that under an assumption of compatibility the concurrency and frame rules can still be derived. Note that it was not needed to assume compatibility for the proof rules with Hoare triples since tests already come with that property. In contrast, the new triples do not need additional assumptions besides the compatibility condition.

We begin with an auxiliary result.

Lemma 7.3 *Assume P, Q are forward compatible. Then $dom(P) * dom(Q) = dom(P * Q)$, i.e., $*$ distributes over domain.*

A proof can be found in the Appendix.

Lemma 7.4 *If Q_1, Q_2 are forward compatible then the concurrency rule for dual partial correctness triples holds, i.e., for tests p_1, p_2, q_1, q_2*

$$\frac{p_1 \;[\![Q_1]\!]\; q_1 \qquad p_2 \;[\![Q_2]\!]\; q_2}{p_1 * p_2 \;[\![Q_1 * Q_2]\!]\; q_1 * q_2} \; .$$

Again this holds also when Q_1 and Q_2 are backward compatible and Plotkin instead of dual partial correctness triples are used.

Proof. By assumption we have $p_1 \subseteq dom(Q_1 \,;\, q_1)$, $p_2 \subseteq dom(Q_2 \,;\, q_2)$ and the restricted variant of the reverse exchange law. Hence

$$\begin{array}{ll} & p_1 * p_2 \\ \subseteq & dom(Q_1 \,;\, q_1) * dom(Q_2 \,;\, q_2) \\ = & dom((Q_1 \,;\, q_1) * (Q_2 \,;\, q_2)) \\ \subseteq & dom((Q_1 * Q_2) \,;\, (R_1 * R_2)) \; . \end{array}$$

□

We characterised the behaviour of the triples "dual" on purpose since the calculations given above are symmetric to the algebraic approach of [7]. It is not hard to see that a further application of the compact characterisation of locality presented in Section 6 also gives the following result.

Lemma 7.5 *If Q is local and forward compatible with* skip *then the frame rule for dual partial correctness triples holds, i.e.,*

$$\frac{p \;[\![Q]\!]\; q}{p * r \;[\![Q]\!]\; q * r} \; .$$

(A dual result again holds for Plotkin triples).

Proof. Assume $p \subseteq dom(Q \,;\, q)$ for a command Q and test q. Hence

$$\begin{array}{ll} & p * r \\ \subseteq & dom(Q \,;\, q) * (\mathsf{skip} \,;\, r) \\ = & dom(Q \,;\, q) * dom(\mathsf{skip} \,;\, r) \\ = & dom((Q \,;\, q) * (\mathsf{skip} \,;\, r)) \\ \subseteq & dom((Q * \mathsf{skip}) \,;\, (q * r)) \\ \subseteq & dom(Q \,;\, (q * r)) \; . \end{array}$$

□

8 Further Variations

Both proof rules of the previous section have the restriction that compatible pairs of commands are needed. The reason for this is that, by Lemma 2.6, in the relational approach only $\# \subseteq \rhd ; \lhd$ holds. If we would have $\mathsf{skip} \times \mathsf{skip} \subseteq \rhd ; \lhd$ the proof of the reverse exchange law would not have any restrictions. However this requires an extension of the definition of $\rhd$ such that the composition $\rhd ; \lhd$ has the same behaviour as skip on incombinable pairs of states.

An idea would be to lift commands to relations between sets containing at most a single state. The empty set is then the result of joining incombinable pairs of states. We define $\Sigma_s =_{df} \{\{\sigma\} : \sigma \in \Sigma\} \cup \{\emptyset\}$ and, for sets $X, Y \in \Sigma_s$,

$$X *_s Y \lhd (X, Y) \tag{4}$$

where $X *_s Y =_{df} \{\sigma_1 \bullet \sigma_2 : \sigma_1 \# \sigma_2, \sigma_1 \in X, \sigma_2 \in Y\}$. In the special case with $X = \{\sigma_1\} \neq \emptyset$ and $Y = \{\sigma_2\} \neq \emptyset$ we have that

$$\{\sigma_1\} *_s \{\sigma_2\} \lhd (\{\sigma_1\}, \{\sigma_2\}) \ .$$

We denote the lifting of this operation to relations by $*_s$ again.

This modification allows a relational model of the algebraic structure of a *locality bimonoid* defined in [7].

Definition 8.1 A *locality bimonoid* is an algebraic structure $(S, \leq, *, 1_*, ;, 1_;)$ where $(S, \leq)$ is partially ordered and $*$, ; are monotone operations on S. Moreover, $(S, *, 1_*)$ needs to be a commutative monoid and $(S, ;, 1_;)$ a monoid. Additionally, the structure has to satisfy the exchange law and $1 * 1 = 1$.

To obtain a relational model for this structure one may interpret the order $\leq$ as the reverse set inclusion order $\supseteq$. Of course, by this the mentioned reverse exchange law turns into the normal one and the relational approach into a refinement-based setting. Moreover, we have the following result.

Lemma 8.2 skip *is idempotent w.r.t.* $*$*, i.e.,* $\mathsf{skip} * \mathsf{skip} = \mathsf{skip}$.

Proof. Since $\mathsf{skip} * \mathsf{skip}$ is a test the $\subseteq$-direction is immediate. □

In summary, we summarise the following result.

Lemma 8.3 $(\mathcal{P}(\Sigma_s \times \Sigma_s), \supseteq, *_s, \mathsf{emp}, ;, \mathsf{skip})$ *forms a locality bimonoid.*

Note that by this modification $\sqcup$ and $\sqcap$ turn into $\cap$ and $\cup$. In particular, the test subalgebra is used as an algebraic counterpart to model assertions. Hence, the interpretation of the notion of a test becomes very unnatural, since e.g. $p \wedge q$ will be identified, unusually, in the algebra with $p \sqcup q$ and $p \vee q$ with $p \sqcap q$. Algebraically these modifications of the model entail simplifications. There are no additional constraints needed to validate the reverse exchange law and hence the original concurrency and frame rules hold. The reason for this is the inequation $\mathsf{skip} \times \mathsf{skip} \subseteq \rhd ; \lhd$ that requires the introduction of an extra failure-state to capture the join of incombinable states. However, considering this extra failure-state makes the whole approach more complicated and artificial from the model-theoretical view.

9 Conclusion

Although neither the full nor small exchange law holds in the relational calculus, we were still able to obtain reasonable variants of the concurrency and frame rules. The proofs greatly benefit from the (restricted) reverse exchange law and hence are almost as simple as the ones in [7]. The advantage of the relational framework is that it admits choice and the corresponding distributivity laws without effort by using relational union.

Further work on this approach includes investigations on so-called interference relations [5]. The intention with such relations is to provide admissible behaviour of commands in a concurrent context so that interference between these commands is excluded. By this we hope to include more concrete models for the application domain of the presented relational approach.

Acknowledgements: We are grateful to Tony Hoare for fruitful discussions and comments and to Andreas Zelend for valuable remarks. Moreover, we thank all reviewers for their comments that helped to significantly improve the presentation of this paper. This research was partially funded by the DFG project *MO 690/9-1 AlgSep — Algebraic Calculi for Separation Logic.*

References

1. Brookes, S.: A semantics for concurrent separation logic. Theoretical Computer Science 375, 227–270 (2007)
2. Calcagno, C., O'Hearn, P.W., Yang, H.: Local Action and Abstract Separation Logic. In: Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science. pp. 366–378. IEEE Press (2007)
3. Dang, H.H., Höfner, P.: Variable side conditions and greatest relations in algebraic separation logic. In: de Swart, H. (ed.) Proceedings of the 12th international conference on Relational and Algebraic Methods in Computer Science. Lecture Notes in Computer Science, vol. 6663, pp. 125–140. Springer (2011)
4. Dang, H.H., Höfner, P., Möller, B.: Algebraic separation logic. Journal of Logic and Algebraic Programming 80(6), 221–247 (2011)
5. Dinsdale-Young, T., Dodds, M., Gardner, P., Parkinson, M., Vafeiadis, V.: Concurrent abstract predicates. In: D'Hondt, T. (ed.) ECOOP 2010 — Object-Oriented Programming, 24th European Conference, Maribor, Slovenia, June 21-25, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6183, pp. 504–528. Springer (2010)
6. Hoare, C.A.R.: An Algebra for Program Designs. Notes on Summer School in Software Engineering and Verification in Moscow (2011), http://research.microsoft.com/en-us/um/redmond/events/sssev2011/slides/tony_1.pptx
7. Hoare, C.A.R., Hussain, A., Möller, B., O'Hearn, P.W., Petersen, R.L., Struth, G.: On Locality and the Exchange Law for Concurrent Processes. In: Katoen, J.P., König, B. (eds.) CONCUR 2011 - 22nd International Conference on Concurrency Theory. Lecture Notes in Computer Science, vol. 6901, pp. 250–264. Springer (2011)
8. Hoare, C.A.R., Möller, B., Struth, G., Wehrman, I.: Concurrent Kleene Algebra and its Foundations. Journal of Logic and Algebraic Programming 80(6), 266–296 (2011)

9. Kozen, D.: Kleene algebra with tests. ACM Transactions on Programming Languages and Systems 19(3), 427–443 (1997)
10. Möller, B.: Kleene getting lazy. Science of Computer Programming 65, 195–214 (2007)
11. O'Hearn, P.W.: Resources, Concurrency, and Local Reasoning. Theoretical Computer Science 375(1-3), 271–307 (2007)
12. O'Hearn, P.W., Reynolds, J.C., Yang, H.: Local reasoning about programs that alter data structures. In: Fribourg, L. (ed.) CSL '01: Proceedings of the 15th International Workshop on Computer Science Logic. Lecture Notes in Computer Science, vol. 2142, pp. 1–19. Springer (2001)
13. Plotkin, G.D.: A structural approach to operational semantics. J. Log. Algebr. Program. 60–61, 17–139 (2004)
14. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: LICS '02: Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science. pp. 55–74. IEEE Computer Society (2002)

10 Appendix: Deferred Proofs

Proof of Lemma 2.8.
For arbitrary σ we have

$$
\begin{array}{ll}
 & \sigma\ dom(P * Q)\ \sigma \\
\Leftrightarrow & \quad \{\![\text{ definitions of } * \text{ and domain }]\!\} \\
 & \exists\, \sigma_1, \sigma_2, \tau_1, \tau_2 \,.\, \sigma_1 \# \sigma_2 \wedge \sigma = \sigma_1 \bullet \sigma_2 \wedge \tau_1 \# \tau_2 \wedge \sigma_1\ P\ \tau_1 \wedge \sigma_2\ Q\ \tau_2 \\
\Rightarrow & \quad \{\![\text{ omitting conjunct } \tau_1 \# \tau_2 \text{ and shifting quantification over } \tau_1, \tau_2 \text{ }]\!\} \\
 & \exists\, \sigma_1, \sigma_2 \,.\, \sigma_1 \# \sigma_2 \wedge \sigma = \sigma_1 \bullet \sigma_2 \wedge \exists\, \tau_1, \tau_2 \,.\, \sigma_1\ P\ \tau_1 \wedge \sigma_2\ Q\ \tau_2 \\
\Leftrightarrow & \quad \{\![\text{ definition of domain }]\!\} \\
 & \exists\, \sigma_1, \sigma_2 \,.\, \sigma_1 \# \sigma_2 \wedge \sigma = \sigma_1 \bullet \sigma_2 \wedge \sigma_1\ dom(P)\ \sigma_1 \wedge \sigma_2\ dom(Q)\ \sigma_2 \\
\Leftrightarrow & \quad \{\![\text{ definition of } * \text{ }]\!\} \\
 & \sigma\ (dom(P) * dom(Q))\ \sigma\ .
\end{array}
$$

□

In Def. 6.2 we stated that a command Q *preserves* a test r iff

$$\lhd\,;\,(Q \times r)\,;\#\ \subseteq\ Q\,;\lhd\,;\,(\mathsf{skip} \times r)$$

and called a command Q *local* iff Q preserves all tests.

We first list a few useful properties in connection with these notions.

Lemma 10.1

1. skip *preserves* skip .
2. *For arbitrary* Q *and* r *we have*

$$\lhd\,;\,(Q \times r)\,;\#\ \ \subseteq\ \ (Q * \mathsf{skip})\,;\lhd\,;\,(\mathsf{skip} \times r)\ .$$

3. *If* Q *preserves a test* r *then* $Q * r \subseteq Q\,;(\mathsf{skip} * r)$.
   In particular, $\mathsf{skip} * \mathsf{skip} \subseteq \mathsf{skip}$. *Hence if* Q *is local then* $Q * \mathsf{skip} \subseteq Q$.

Proof.

1. The claim follows immediately by setting $Q = \mathsf{skip} = r$ in Definition 6.2.
2. We calculate:

$$\begin{array}{ll} & \lhd\,;\,(Q \times r)\,;\,\# \\ = & \quad \{\![\text{ neutrality of } \mathsf{skip} \,]\!\} \\ & \lhd\,;\,(Q\,;\mathsf{skip} \times \mathsf{skip}\,;r)\,;\,\# \\ = & \quad \{\![\,;/\times \text{ exchange (1) }]\!\} \\ & \lhd\,;\,(Q \times \mathsf{skip})\,;\,(\mathsf{skip} \times r)\,;\,\# \\ = & \quad \{\![\text{ by Definition 2.4 }]\!\} \\ & \lhd\,;\,(Q \times \mathsf{skip})\,;\,\#\,;\,(\mathsf{skip} \times r) \\ \subseteq & \quad \{\![\,\# \subseteq \rhd\,;\lhd \text{ and isotony }]\!\} \\ & \lhd\,;\,(Q \times \mathsf{skip})\,;\rhd\,;\lhd\,;\,(\mathsf{skip} \times r) \\ = & \quad \{\![\text{ definition of } * \,]\!\} \\ & (Q * \mathsf{skip})\,;\lhd\,;\,(\mathsf{skip} \times r)\;. \end{array}$$

3. The first claim is immediate from the definition of locality by right-composing both sides of the inclusion with $\rhd$, isotony and the definition of $*$. Hence the second claim is trivial by isotony. The third claim follows by setting $r = \mathsf{skip}$ and using $\mathsf{skip} * \mathsf{skip} = \mathsf{skip}$.

□

We can now give the

Proof of Theorem 6.3
The direction ($\Rightarrow$) is just Lemma 10.1.3. For ($\Leftarrow$) we obtain by Lemma 10.1.3 and the assumption, for arbitrary test r,

$$\lhd\,;\,(Q \times r)\,;\,\# \;\subseteq\; (Q * \mathsf{skip})\,;\lhd\,;\,(\mathsf{skip} \times r) \;\subseteq\; Q\,;\lhd\,;\,(\mathsf{skip} \times r)\;.$$

□

Corollary 10.2 $Q * \mathsf{skip} \subseteq Q \;\Leftrightarrow\; \lhd\,;(Q \times \mathsf{skip})\,;\#\subseteq Q\,;\lhd\,.$

Proof. The direction ($\Leftarrow$) follows from isotony. For the other direction we immediately get by definition and isotony $\lhd\,;(Q \times \mathsf{skip})\,;\rhd\,;\lhd \subseteq Q\,;\lhd$ since $Q * \mathsf{skip} \subseteq Q$. Now the claim follows from Lemma 2.6 using $\# \subseteq \rhd\,;\lhd$. □

Next we turn to Section 7. To prove Lemma 7.3 we first sum up a few results.

Corollary 10.3 $\#\,;(U \times U)\,;\rhd = \rhd\,;U.$

For a proof we refer to [4].

Lemma 10.4 *If commands P, Q are forward compatible then* $(P\,;U) * (Q\,;U) = (P * Q)\,;U.$

Proof. We calculate

$$
\begin{array}{ll}
 & (P\,;U) * (Q\,;U) \\
= & \quad \{\![\text{ definition of } *]\!\} \\
 & \lhd\,;(P\,;U \times Q\,;U)\,;\rhd \\
= & \quad \{\![\text{ Lemma 2.6, Equation (1) }]\!\} \\
 & \lhd\,;\#\,;(P \times Q)\,;(U \times U)\,;\rhd \\
\subseteq & \quad \{\![\ P, Q \text{ forward compatible }]\!\} \\
 & \lhd\,;(P \times Q)\,;\#\,;(U \times U)\,;\rhd \\
= & \quad \{\![\text{ Corollary 10.3 }]\!\} \\
 & \lhd\,;(P \times Q)\,;\rhd\,;U \\
= & \quad \{\![\text{ definition of } *]\!\} \\
 & (P * Q)\,;U \ .
\end{array}
$$

The reverse inequation follows similarly from Corollary 10.3 and isotony. □

Finally we are able to prove Lemma 7.3.

Proof of Lemma 7.3
First note that $dom(P) = P\,;U \cap \mathsf{skip}$. The same holds for Q. By this we calculate

$$dom(P) * dom(Q) = (P;U \cap \mathsf{skip}) * (Q;U \cap \mathsf{skip}) \subseteq (P;U) * (Q;U) = (P * Q);U \ .$$

Moreover $dom(P) * dom(Q) \subseteq \mathsf{skip}$ since both are tests. Hence we can conclude $dom(P) * dom(Q) \subseteq (P * Q)\,;U \cap \mathsf{skip} = dom(P * Q)$.

The reverse inclusion was shown in Lemma 2.8. □