On locality and the exchange law for concurrent processes

C. A. R. Hoare, Akbar Hussain, Bernhard Möller, Peter W. O'Hearn, Rasmus Lerchedahl Petersen, Georg Struth

Angaben zur Veröffentlichung / Publication details:

Hoare, C. A. R., Akbar Hussain, Bernhard Möller, Peter W. O'Hearn, Rasmus Lerchedahl Petersen, and Georg Struth. 2011. "On locality and the exchange law for concurrent processes." *Lecture Notes in Computer Science* 6901: 250–64. https://doi.org/10.1007/978-3-642-23217-6_17.

On Locality and the Exchange Law for Concurrent Processes

C.A.R. Hoare[1], A. Hussain[2], B. Möller[3], P.W. O'Hearn[2], R.L. Petersen[2], and G. Struth[4]

[1] Microsoft Research Cambridge
[2] Queen Mary University of London
[3] Universität Augsburg
[4] University of Sheffield

Abstract. This paper studies algebraic models for concurrency, in light of recent work on Concurrent Kleene Algebra and Separation Logic. It establishes a strong connection between the Concurrency and Frame Rules of Separation Logic and a variant of the exchange law of Category Theory. We investigate two standard models: one uses sets of traces, and the other is state-based, using assertions and weakest preconditions. We relate the latter to standard models of the heap as a partial function. We exploit the power of algebra to unify models and classify their variations.

1 Introduction

The theory of concurrency is composed of a bewildering variety of models founded on diverse concepts. The general outlook in this paper is one of unification. Wouldn't it be wonderful if we could find a more general theory that accepted the diverse models as instances? Might it be possible to have a comprehensive treatment of concurrency in which shared memory (in weak or strong forms), message passing (in its numerous forms), interleaving and independence models were seen to be part of the same theory with the same core axioms, specialised in different ways, rather than founded on disparate models or calculi? Ideally, such a unifying theory would even connect up the models, program logics, and operational semantics. Of course, unification has long been an aim in concurrency theory, with many translations between models aiding understanding.

With such lofty general aims, we state right away that this paper is but a modest attempt to contribute to such a programme. We in no way have a grand unifying theory at the moment. But we are probing, and as a result altering, algebraic concepts by comparing them with concrete models, in the hope that evolution to a general theory is eventually possible.

This paper builds on previous work on Concurrent Kleene Algebra (CKA, [8]), where an algebra mixing primitives for concurrent ($*$) and sequential ($;$) composition was introduced, generalizing the Kleene algebra of sequential programs [9]. The standout feature of these algebras is the presence of an ordered version of the exchange law from 2-categories or bi-categories

$$(p * r); (q * s) \sqsubseteq (p\,; q) * (r\,; s)$$

This law makes immediate sense in an interleaving model of concurrency. Suppose you have a trace $t = t_1; t_2$ where t_1 is an interleaving of two traces t_p and

t_r, and t_2 of t_q and t_s. Then t is certainly also an interleaving of $t_p; t_q$ and $t_r; t_s$. But, as was shown in [8], the law also validates proof rules which originally arose in a completely different model of concurrency, based on separation of resources, the Concurrency and Frame Rules from Concurrent Separation Logic [10, 3],

$$\frac{\{P_1\}\, C_1\, \{Q_1\} \quad \{P_2\}\, C_2\, \{Q_2\}}{\{P_1 * P_2\}\, C_1 \| C_2\, \{Q_1 * Q_2\}} \qquad \frac{\{P\}\, C\, \{Q\}}{\{P * F\}\, C\, \{Q * F\}}.$$

These rules are derived at once from the algebraic structure of CKA, meaning that the modular reasoning that these rules support could conceivably be applicable in a wide variety of circumstances.

This is a remarkable situation. The concurrency and Frame Rules in Concurrent Separation Logic (CSL) have historically relied on somewhat subtle locality properties of the semantics of programs [11, 4], which say that programs access resources in a circumscribed fashion. But these laws flow extremely easily in a CKA, and examples of CKAs have been described which do not have anything that looks like locality conditions on resource access. Furthermore, in [8] the validity of Hoare logic laws is referred to, slightly tongue in cheek, as due to a 'cheat', of identifying assertions with programs. Our starting point was simply to understand: what is going on here? Is the easy validity of these laws in CKA a sleight of hand? Or, do they have the same import as in CSL (but generalized)?

We consider another concrete model, the Resource Model, based on the primitive of resource separation used in CSL. This model is built from predicate transformers, and is equipped with notions of parallel and sequential composition. It is directly a model of Concurrent Separation Logic. We show that two algebraic interpretations of Hoare triples coincide, in the model, with the usual interpretation of pre/post specs for predicate transformers. This shows that there is no cheating or sleight of hand: the various logical laws of Hoare and Separation Logic 'mean the same thing' when the algebra is instantiated to this standard and independently existing model of Separation Logic.

An interesting point is that we find some, but not all, of the structure of a CKA in the Resource Model. This leads us to propose a notion of 'locality bimonoid', which is derived from algebraic considerations as well as computational considerations concerning concurrency and resources. The basic structure is that of a pair of ordered monoids on the same underlying poset, linked by the exchange law. Locality is expressed using the equation $f = f * \mathsf{skip}$, where skip is the unit of the sequencing operator ; . We show that this provides a simpler and much more general alternate characterization of a semantic definition of locality used in work on Separation Logic. We end up with a situation where there is an algebra satisfying exchange but not locality, and this equation serves as a healthiness condition (in the sense of Dijkstra [5] and He/Hoare [7]) used to distinguish a subalgebra of local elements, which we call the *local core*. We also find, in contrast with our initial expectation, that the exchange law does not itself rely on locality: the non-local algebra admits exchange, and as a consequence it validates the Concurrency but not the Frame Rule of CSL.

In addition to the aforementioned works on CKA and CSL, we mention other prior related work on the exchange law in concurrency. The structure of

two monoids with shared unit and exchange has been called 'concurrent monoid' in [8]. This structure was studied earlier by Gischer in the 1980s [6], and then by Bloom and Ésik [2] in the 1990s. Gischer showed that pomsets or series parallel posets are models of concurrent monoids, and that series parallel posets form the free ordered algebras in that variety. Bloom and Ésik showed that languages with shuffle form models of concurrent monoids. The relation of this paper to these works is similar to the relation to the CKA work: we introduce a more general structure, locality bimonoid, which has a concurrent monoid as the local core, and we connect this structure to a standard model of pre/post specs, complementing the models in [6, 2, 8] based on traces.

All calculational proofs in this paper have been formally verified by automated theorem proving in Isabelle; they can be found online [1].

2 Two Models

Before moving to algebra, we describe two concrete models that we will refer to throughout the paper.

The Trace Model. Let A be a set. Then *Traces* is the set of finite sequences over A. The powerset $P(\mathit{Traces})$ will be the carrier set of this algebra. We have the following two binary operations on $P(\mathit{Traces})$

1. $T_1 * T_2$ is the set of interleavings of traces in T_1 with traces in T_2;
2. $T_1 ; T_2$ is the set of concatenations of traces in T_1 with traces in T_2.

The set $\{\epsilon\}$ functions as a unit for both ; and $*$, where ϵ is the empty sequence. The exchange law in this model was already stated in [2]. $P(\mathit{Traces})$ is a prototypical CKA and even a quantale [8]. This means, among other things, that it is a complete lattice in which $*$ and ; distribute through $\bigsqcup$, and that both $*$ and ; have a left and right zero (the empty set, $\emptyset$).

The Resource Model. Let $(\Sigma, \bullet, u)$ be a partial, commutative monoid, given by a partial binary operation $\bullet$ and unit u where the unity, commutativity and associativity laws hold. Here equality means that both sides are defined and equal, or both are undefined. In examples we will often refer to the partial monoid of heaps, where Σ is the set of finite partial functions from naturals to naturals, $\Sigma = N \rightharpoonup_f N$, with $\bullet$ being the partial operation of union of functions with disjoint domains and u being the empty function. The domain of a heap h is denoted by $\mathit{dom}(h)$.

The powerset $P(\Sigma)$ has an ordered total commutative monoid structure $(*, \mathit{emp})$ given by

$$p * q = \{\sigma_0 \bullet \sigma_1 \mid \sigma_0 \# \sigma_1 \wedge \sigma_0 \in p \wedge \sigma_1 \in q\}$$
$$\mathit{emp} = \{u\}$$

where $\sigma_0 \# \sigma_1$ means that $\sigma_0 \bullet \sigma_1$ is defined.

The monotone function space $[P(\Sigma) \to P(\Sigma)]$ will be the carrier set of the algebra in the Resource Model. These functions represent (backwards) predicate transformers and the algebra has the following operators. Using F_i to range over

predicate transformers, and Y, Y_i... to range over subsets of Σ, we define

$$\begin{aligned}(F_1 * F_2)Y &= \bigcup\{F_1Y_1 * F_2Y_2 \mid Y_1 * Y_2 \subseteq Y\}\\ \mathsf{nothing}\ Y &= Y \cap emp\\ (F_1\,;F_2)Y &= F1(F2(Y))\\ \mathsf{skip}\ Y &= Y\end{aligned}$$

Here we are overloading $*$, relying on context to disambiguate: sometimes, as on the right of the definition of $F_1 * F_2$, it refers to an operator on predicates, and other times, as on the left of the definition, to an operation on predicate transformers. The definition of the predicate transformer $F_1 * F_2$ was suggested to O'Hearn by Hongseok Yang in 2002, shortly after CSL was introduced (and before it was published). It is motivated by the concurrency proof rule of CSL. The idea is that we start with a postcondition Y, split it into separate assertions Y_1 and Y_2, and apply the Concurrency Rule backwards to get a precondition $F_1Y_1 * F_2Y_2$ for the parallel composition of F_1 and F_2. We union over all such, to obtain the weakest possible precondition.

We use the reverse pointwise ordering $\sqsubseteq$ on $[P(\Sigma) \to P(\Sigma)]$: $F \sqsubseteq G$ iff $\forall X.\, FX \supseteq GX$. According to this definition, the least element is the function $\lambda X.\Sigma$, corresponding to the weakest (liberal) precondition transformer for divergence. We are thinking of preconditions for partial rather than total correctness here, so we do not insist on Dijkstra's 'law of the excluded miracle' $F\emptyset = \emptyset$.

Note that, in contrast to the Trace Model, the Resource Model has distinct units: `nothing` is the unit of $*$ and `skip` that of ;.

3 Exchange

In this section we introduce the main definitions surrounding the exchange law, we make the connection to program logic, and we classify our two running examples in terms of the introduced notions.

3.1 Algebra

Definition 3.1 (Ordered bisemigroup) An *ordered bisemigroup* $(M, \sqsubseteq, *, ;)$ is a partially ordered set $(M, \sqsubseteq)$ with two monotone compositions $*$ and ;, such that $(M, *)$ is a commutative semigroup and $(M, ;)$ is a semigroup.

Definition 3.2 (Exchange Laws) Over the signature of an ordered bisemigroup $(M, \sqsubseteq, *, ;)$, we consider the following formulas for $p, q, r, s \in M$:

$$\begin{aligned}(p * r)\,;(q * s) &\sqsubseteq (p; q) * (r; s) && \text{(full) exchange}\\ (r * p); q &\sqsubseteq r * (p; q) && \text{small exchange I}\\ p; (q * r) &\sqsubseteq (p; q) * r && \text{small exchange II}\end{aligned}$$

As mentioned in Section 1, it is easy to see that the exchange law holds in the Trace Model. The situation in the Resource Model is subtler, and it will be helpful to exemplify the full law and give counterexamples to the small versions.

Example 3.3 (Exchange in Heap Model) We consider a specialization of the Resource Model, based on heaps. With this model we will use commands

$$[n] := m \quad = \quad \lambda X.\ \{h[n \mapsto l] \mid h \in X, l \in \mathbb{N}, n \in dom(h), h(n) = m\}$$

where $h[n \mapsto l]$ is function update. The command $[n] := m$ updates the memory at location n to have the value m.

With this definition, we can give this example of the exchange law.

$$\begin{array}{ll} ([10] := 55 * [20] := 66)\,;([20] := 77 * [10] := 88) & (\neq \top) \\ \sqsubseteq & \\ ([10] := 55\,;[20] := 77) * ([20] := 66\,;[10] := 88) & (= \top) \end{array}$$

The reason that this holds is that, as there is a race on locations 10 and 20, the lower term is $\top$. The reader might enjoy verifying this in the model. On the other hand, the upper term is not $\top$ since, given a state whose domain contains both 10 and 20, we can split it for the first parallel composition, and again for the second: thus, there is a non-empty precondition for the upper program.

Note that this example also shows that the reverse direction of the exchange law is not valid in the model. ■

Example 3.4 (Invalidity of Small Exchange in Heap Model) To obtain a counterexample to the law $p\,;(q * r) \sqsubseteq (p\,;q) * r$, we choose $p = q = \mathsf{nothing}$ and $r = \mathsf{skip}$. The left hand side is then $\mathsf{nothing}\,;(\mathsf{nothing} * \mathsf{skip}) = \mathsf{nothing}$.

We now observe that in the Resource Model, $\mathsf{nothing}\,;\mathsf{nothing} = \mathsf{nothing}$, so the right hand side simplifies to skip. And clearly it is not the case that $\mathsf{nothing} \sqsubseteq \mathsf{skip}$ in the Resource Model.

The same instantiation yields a counterexample to $(r * p)\,;q \sqsubseteq r * (p\,;q)$. ■

Proposition 3.5 *The Resource Model satisfies the exchange law.*

Proof. Let F_1, F_2, G_1 and G_2 be (monotone) predicate transformers and $X \subseteq \Sigma$. Since the order on predicate transformers is pointwise reverse inclusion, we aim to show

$$((F_1\,;F_2) * (G_1\,;G_2))X \subseteq ((F_1 * G_1)\,;(F_2 * G_2))X$$

The left hand side expands to

$$\bigcup \{F_1(F_2X_F) * G_1(G_2X_G) \mid X_F * X_G \subseteq X\}$$

Now, for every subsplitting $X_F * X_G$ of X, we see that $F_2X_F * G_2X_G$ is a subsplitting of $(F_2 * G_2)X = \bigcup\{F_2A * G_2B \mid A * B \subseteq X\}$, simply because it is one of the elements of the union. So we can overapproximate the left hand side:

$$\bigcup\{F_1(F_2X_F) * G_1(G_2X_G) \mid X_F * X_G \subseteq X\} \subseteq \bigcup\{F_1A * G_1B \mid A * B \subseteq (F_2 * G_2)X\}$$

and this overapproximation quickly rewrites to the right hand side:

$$\bigcup\{F_1A * G_1B \mid A * B \subseteq (F_2 * G_2)X\} = (F_1 * G_1)((F_2 * G_2)X) = (F_1 * G_1)\,;(F_2 * G_2)X$$

□

It is a straightforward extension to consider units.

Definition 3.6 (Ordered bimonoid) An *ordered bimonoid* is a structure $(M, \sqsubseteq, *, \mathsf{nothing}, ;, \mathsf{skip})$ such that $(M, \sqsubseteq, *, ;)$ is an ordered bisemigroup and $(M, *, \mathsf{nothing})$ and $(M, ;, \mathsf{skip})$ are monoids.

A first important consequence is

Lemma 3.7 *In an ordered bimonoid with exchange,* $\mathsf{skip} \sqsubseteq \mathsf{nothing}$ *and* $\mathsf{nothing} \sqsubseteq \mathsf{nothing} \mathbin{;} \mathsf{nothing}$.

It is tempting to require the two units to be identical, but in the Resource Model they are not. Still, when the units are equal we are in a special situation.

Lemma 3.8 *In an ordered bimonoid* $(M, \sqsubseteq, *, \mathsf{nothing}, ;, \mathsf{skip})$ *with shared unit (meaning* $\mathsf{nothing} = \mathsf{skip}$*), the full exchange law implies the small exchange laws.*

3.2 Program Logic

In our algebra we may interpret pre/post specs in two ways.

Definition 3.9 Let $(M, \sqsubseteq, ;)$ be an ordered semigroup and $p, c, q \in M$.

1. The *Hoare triple* $\{p\}\, c\, \{q\}$ is defined by $\{p\}\, c\, \{q\} \;\Leftrightarrow\; p; c \sqsubseteq q$
2. The *Plotkin triple* $p \to_c q$ is defined by $p \to_c q \;\Leftrightarrow\; p \sqsupseteq c; q$

The Hoare triple can be thought of in terms of the past. In the Trace Model, it says that if p describes events done before command c is run, then q over-approximates p followed by c. Conversely, the Plotkin triple can be thought of in terms of the future: it says that if p describes a possible future, and q describes the future after c is run, then p over-approximates c followed by q.

In the Resource Model, there is a further interpretation of pre/post specs:

Definition 3.10 (Dijkstra Triple) In the Resource Model, if X is a predicate (a set of states) and F is a predicate transformer, then the relationship $X \subseteq FY$ says that X is a valid precondition for F with post Y:

$$\langle\!\langle X \rangle\!\rangle\, F\, \langle\!\langle Y \rangle\!\rangle \quad \Leftrightarrow \quad X \subseteq FY$$

While this definition can be given for arbitrary predicate transformers, the rule of consequence for F, that $X \subseteq X' \wedge \langle\!\langle X' \rangle\!\rangle\, F\, \langle\!\langle Y' \rangle\!\rangle \wedge Y' \subseteq Y \Rightarrow \langle\!\langle X \rangle\!\rangle\, F\, \langle\!\langle Y \rangle\!\rangle$, holds iff F is monotone.

While the interpretations in Definition 3.9 make sense on their own, it is useful to try to connect them to the ordinary understanding of specs in state-based (rather than history-based) models; we will do that in Section 5.

We now leave the concrete model and the Dijkstra triples, and describe the program logic that results from Definition 3.9.

Lemma 3.11 (Hoare Logic) *In an ordered semigroup the following rules hold:*

Rule of Consequence:

$$\forall p, c, q, p', q'.\ \{p'\}\, c\, \{q'\} \ \wedge\ p \sqsubseteq p' \wedge q' \sqsubseteq q \quad \Rightarrow \quad \{p\}\, c\, \{q\}$$

Sequencing Rule:

$$\forall p, c, r, c', q.\ \{p\}\, c\, \{r\}\ \wedge\ \{r\}\, c'\, \{q\} \quad \Rightarrow \quad \{p\}\, c; c'\, \{q\}$$

If the ordered semigroup is an ordered monoid, then the following holds:

Skip Rule:

$$\forall p.\ \{p\}\, \mathsf{skip}\, \{p\}$$

Furthermore, this lemma also holds with Plotkin in place of Hoare triples, except that the ordering is reversed in the Rule of Consequence.

In a bisemigroup we can obtain proof rules from Separation Logic.

Theorem 3.12 (Separation Logic) *In an ordered bisemigroup,*

1. *The exchange law holds iff we have the*
   Concurrency Rule:
   $$\forall p, c, q, p', c', q'.\ \{p\}\, c\, \{q\}\ \wedge\ \{p'\}\, c'\, \{q'\} \quad \Rightarrow \quad \{p * p'\}\, c * c'\, \{q * q'\}$$
   Furthermore, the same holds true with Plotkin in place of Hoare triples.
2. *The first small exchange law holds iff we have*
   Frame Rule (for Hoare triples):
   $$\forall p, c, q, r.\ \{p\}\, c\, \{q\} \Rightarrow \{p * r\}\, c\, \{q * r\}$$
3. *The second small exchange law holds iff we have*
   Frame Rule (for Plotkin triples):
   $$\forall p, c, q, r.\ p \to_c q \Rightarrow p * r \to_c q * r$$

This abstract treatment does not cover all of the rules of Hoare or Separation Logic, notably the conjunction rule and rules for variable renaming and quantifiers, as there is not enough structure to state them.

It is worth remarking that we also have temporal frame rules.

Futuristic Frame Rule (for Plotkin Triples)

$$\forall p, c, q, r.\ p \to_c q \Rightarrow p; r \to_c q; r$$

Archaic Frame Rule (for Hoare Triples)

$$\forall p, c, q, r.\ \{p\}\, c\, \{q\} \Rightarrow \{r; p\}\, c\, \{r; q\}$$

We summarize the situation with our two example models as follows.

Proposition 3.13 (Classification)

1. *The Trace Model* $(P(\mathit{Traces}), \sqsubseteq, *, \{\epsilon\}, ;, \{\epsilon\})$ *is an ordered bimonoid with shared unit satisfying the exchange law. Consequently, it satisfies all of the Hoare and Separation Logic rules mentioned in this section.*
2. *The Resource Model* $([P(\Sigma) \to P(\Sigma)], \sqsubseteq, *, \mathsf{nothing}, ;, \mathsf{skip})$ *is an ordered bimonoid with distinct units, satisfying the full but not small exchange laws. Consequently, it satisfies the rules of Hoare Logic and the Concurrency Rule of Separation Logic, but not the Frame Rule (for Hoare or Plotkin triples).*

The Trace Model is an example of a CKA, defined in [8]. The Resource Model, however, is not. Having two different units causes a difference. Additionally, the model does not have a right zero for ;, an element 0 where $p; 0 = 0$ for all p (while all constant transformers are left zeros). The Trace Model does have a right zero, the empty set.

The properties of the Resource Model seem a bit odd at first sight. We have a situation where the Concurrency Rule (which expresses a form of modular reasoning) is present while the Frame Rule (which expresses locality) is absent. We now probe this issue further.

4 Locality

In this section we develop the concept of locality, culminating in an analysis of what we call a 'locality bimonoid' (an ordered bimonoid with exchange where skip is idempotent for $*$, Definition 4.7). In a locality bimonoid we have the Frame Rules exactly for the local elements, and we also have that the local elements form a locality sub-bimonoid where the inclusion is a Galois insertion. Some of the results in this section do not require all of the data of a locality bimonoid, so we build up to the definition as we go along.

4.1 Local Elements

In the Resource Model, a transformer F is called local if it can be described in terms of its action on parts of the state. This can be expressed as follows:

$$FP = \bigcup\{(FX) * R \mid X * R = P\} \tag{1}$$

We have already seen a non-local transformer, $\mathsf{nothing} = \lambda Y.\, Y \cap emp$, and it will be helpful to see why this contradicts the frame rule. Note that $emp \subseteq \mathsf{nothing}\ emp$, but that (in the heap model) $\{1 \mapsto 2\} \not\subseteq \mathsf{nothing}\{1 \mapsto 2\}$ where $1 \mapsto 2$ is a singleton heap. Thus, as $\{1 \mapsto 2\} * emp = \{1 \mapsto 2\}$, we have $(emp * \{1 \mapsto 2\}) \not\subseteq \mathsf{nothing}(emp * \{1 \mapsto 2\})$. So the usual semantics of pre/post specs for predicate transformers does not validate the frame rule for $\mathsf{nothing}$.

The shape of the right-hand side of equation (1) suggests a connection between the transformers F and $F * \mathsf{skip}$. First, choosing $X = P$ and $R = emp$ we get, for arbitrary F,

$$FP \subseteq \bigcup\{(FX) * R \mid X * R = P\} \subseteq \bigcup\{(FX) * R \mid X * R \subseteq P\} = (F * \mathsf{skip})P$$

i.e., $F * \mathsf{skip} \sqsubseteq F$. We will see that this is a consequence of the exchange law.

So the nontrivial part of (1) is $\bigcup\{(FX) * R \mid X * R = P\} \subseteq FP$. Setting $P = X * R$ for arbitrary X, R, this implies $FX * R \subseteq F(X * R)$ and hence

$$(F * \mathsf{skip})P = \bigcup\{(FX) * R \mid X * R \subseteq P\} \subseteq \bigcup\{F(X * R) \mid X * R \subseteq P\} = FP$$

since F is monotone. This means $F \sqsubseteq F * \mathsf{skip}$. Conversely, if $F \sqsubseteq F * \mathsf{skip}$ then

$$FP \supseteq \bigcup\{(FX) * R \mid X * R \subseteq P\} \supseteq \bigcup\{(FX) * R \mid X * R = P\}$$

These results can be summarised by

Lemma 4.1 *A predicate transformer F satisfies (1) iff $F = F * \mathsf{skip}$.*

We now consider locality on the algebraic level. First, Lemma 3.7 implies that in an ordered bimonoid with exchange, all elements satisfy $p * \mathsf{skip} \sqsubseteq p$, so in this setting the definition below simply states the other direction.

Definition 4.2 An element p in an ordered bisemigroup is *local* if $p * \mathsf{skip} = p$.

Example 4.3 (Local skip) In the Resource Model, skip is local. ■

The idea of locality is to admit local reasoning, and indeed that can be verified on the level of algebra:

Lemma 4.4 *In an ordered bimonoid with exchange, the Frame rules for Plotkin and Hoare triples hold when the 'command' c is local. That is, for $c = c * \mathsf{skip}$*

$$\forall p, q, r.\ \{p\}\, c\, \{q\} \Rightarrow \{p * r\}\, c\, \{q * r\}$$

$$\forall p, q, r.\ p \to_c q \Rightarrow p * r \to_c q * r$$

The local elements form a subalgebra:

Lemma 4.5 *In an ordered bimonoid, the local elements are closed under $*$.*

Lemma 4.6 *In an ordered bimonoid with exchange, the local elements are closed under $;$.*

The units nothing and skip may not be local, though [1], so one cannot immediately form a sub-bimonoid. We now consider the special case of an ordered bimonoid where skip is local. This enables us to localize elements and thus recover a sub-bimonoid.

Definition 4.7 (Locality Bimonoid) A *locality bimonoid* is an ordered bimonoid with exchange where skip is local, i.e., $\mathsf{skip} * \mathsf{skip} = \mathsf{skip}$.

Both our running examples, the Trace Model and the Resource Model, are examples of locality bimonoids. Proposition 3.13 and Example 4.3 states this.

Lemma 4.8 *Let G be a locality bimonoid and $p \in G$. Then $p * \mathsf{skip}$ is local.*

Another way to say this is that the *localizer* $(-) * \mathsf{skip}$ is idempotent. This allows us to define a sub-bimonoid:

Definition 4.9 (Local Core) Let $G = (M, \sqsubseteq, *, \mathsf{nothing}, ;, \mathsf{skip})$ be a locality bimonoid. Then we define the *local core* G_{loc} of G by

$$G_{loc} = (M_{loc}, \sqsubseteq, *, \mathsf{skip}, ;, \mathsf{skip})$$

where $M_{loc} = \{p \in M \mid p = p * \mathsf{skip}\}$, and $*$ and $;$ are restricted appropriately.

Since the localizer is idempotent, selecting the local elements is the same as selecting the image of the localizer.

Lemma 4.10 *If G is a locality bimonoid, then so is G_{loc}.*

Thus, the notion of Local Core is well defined, but we can say more. It is an instance of the special situation of when all elements are local, which gives us a structure called a *concurrent monoid* in [8] (see Definition 4.12 below).

Lemma 4.11 *In a bimonoid, the following are equivalent*

(*i*) nothing *is local*
(*ii*) *All elements are local*
(*iii*) skip *is a unit of* $*$
(*iv*) nothing = skip

Definition 4.12 (Concurrent Monoid, [8]) A *concurrent monoid* is an ordered bimonoid with exchange where nothing = skip.

Since all elements of a concurrent monoid are local, skip is as well, and we immediately have that a concurrent monoid is a locality bimonoid.

Proposition 3.13 and Example 4.3 states that the Trace Model is a concurrent monoid, while the Resource Model is a locality bimonoid which is not a concurrent monoid.

From Lemma 3.8 we can infer that the full and small exchange laws are valid in a concurrent monoid.

Corollary 4.13 (to Lemma 4.11) *Let G be a locality bimonoid. Then G_{loc} is a concurrent monoid.*

There is a simple connection between G and G_{loc}, given by the localizing map $(-) * \mathsf{skip}$. We can even phrase it without needing skip to be local:

Lemma 4.14 *In an ordered bimonoid with exchange, $p * \mathsf{skip}$ is the greatest local element smaller than p.*

When we have a locality bimonoid, the same reasoning gives us a Galois connection between the bimonoid and its core.

Lemma 4.15 *Let G be a locality bimonoid. There is a Galois connection between G and G_{loc}, where localization $(-) * \mathsf{skip} : G \to G_{loc}$ is right adjoint to the inclusion from G_{loc} into G.*

The final result explains the name locality bimonoid. It states that locality coincides with local reasoning in the form of the Frame rule.

Theorem 4.16 *In a locality bimonoid, the Frame rules for Plotkin and Hoare triples hold exactly when the 'command' c is local. That is, $c = c * \mathsf{skip}$ iff*

$$\begin{array}{rl} & \forall p, q, r.\ \{p\}\, c\, \{q\} \Rightarrow \{p * r\}\, c\, \{q * r\} \\ \textit{iff} & \forall p, q, r.\ p \to_c q \Rightarrow p * r \to_c q * r \end{array}$$

In terms of algebraic models of concurrency, this suggests to us that local elements should be considered 'programs', while in general a bimonoid might contain non-program-like elements, such as those that play the role of assertions.

This is a crucial difference between locality bimonoids and Concurrent Kleene Algebras. There seems to be no compelling reason for requiring locality of (the denotations of) assertions, and a locality bimonoid does not require them to be. But, we will see in Section 5 that neither does requiring locality of all elements lose us anything in characterizing pre/post specs in our state-based model.

4.2 Lubs and Fixed-points

The notion of locality bimonoid is not, on its own, rich enough to interpret programs. There are many further constructs one could consider beyond sequencing and parallelism, but perhaps the most important is recursion or iteration; without it, we don't even have Turing-completeness. This subsection does some basic sanity checking in this direction, and can be skipped or skimmed on a first reading, without loss of continuity: it shows that our motivating Resource Model admits a standard treatment of recursion.

Theorem 4.17 *If the order of a locality bimonoid is a complete lattice, then*

1. *The local core is a complete lattice as well.*
2. *Any expression e built from local elements using $*$, ; and $\bigsqcup$ denotes a local element.*
3. *Any expression $e(X)$ built from local elements using $*$, ; and $\bigsqcup$ and one unknown X denotes a monotone function on the local core; it thus has a least (local) fixed-point $\mu X.\, e(X)$ which is again in the local core.*

Part *1* of this theorem is immediate from the Tarski-Knaster fixed-point theorem because the local elements are the fixed-points of the monotonic function $(-) * \mathsf{skip}$. Parts *2* and *3* are consequences of *1*.

This gives us enough information to interpret a programming language with sequencing, parallelism, nondeterminism and recursion. The form of recursion is sufficient to encode iterators for both sequential and parallel composition, as $\mu X.\, F; X$ and $\mu X.\, F * X$. We will connect this to program logic in Section 5.

Finally, although we have noted that the lub of local transformers is local, curiously, we do *not* have that $(-) * \mathsf{skip}$ distributes through $\bigsqcup$.

Example 4.18 (Localization does not preserve binary $\bigsqcup$) We first define a number of heaps:

$$h = \{1 \mapsto 1, 2 \mapsto 2, 3 \mapsto 3\}$$
$$h_{12} = \{1 \mapsto 1, 2 \mapsto 2\} \qquad h_{23} = \{2 \mapsto 2, 3 \mapsto 3\}$$
$$h_1 = \{1 \mapsto 1\} \qquad h_2 = \{2 \mapsto 2\} \qquad h_3 = \{3 \mapsto 3\}$$

This allows us to define the following (constant) transformers:

$$F = \lambda P.\ \{h_1\} \qquad G = \lambda P.\ \{h_3\}$$

And now we obtain a counterexample to distribution through lubs as

$$((F \sqcup G) * \mathsf{skip})\{h_{12}, h_{23}\} \neq ((F * \mathsf{skip}) \sqcup (G * \mathsf{skip}))\{h_{12}, h_{23}\}$$

For the left hand side, we observe that $F \sqcup G = \lambda P.\ \{h_1\} \cap \{h_3\} = \lambda P.\ \emptyset$ and so $(F \sqcup G) * \mathsf{skip} = \lambda P.\ \emptyset$.

For the right hand side, we observe that $\{h\} = \{h_1\} * \{h_{23}\} = (F\ emp) * \{h_{23}\}$ and that $emp * \{h_{23}\} \subseteq \{h_{12}, h_{23}\}$, so $\{h\} \subseteq (F * \mathsf{skip})\{h_{12}, h_{23}\}$. Similarly, $\{h\} = \{h_3\} * \{h_{12}\} = (G\ emp) * \{h_{12}\}$ and $emp * \{h_{12}\} \subseteq \{h_{12}, h_{23}\}$ so $\{h\} \subseteq (G * \mathsf{skip})\{h_{12}, h_{23}\}$. Thus the right hand side is not empty. ■

Being a right adjoint, however, $(-) * \mathsf{skip}$ does distribute through $\bigsqcap$.

As a consequence of this result, we do *not* have that our locality bimonoid of predicate transformers is a quantale: a quantale requires that $p * (\cdot)$ preserves all lubs, for arbitrary p.

There are two ways to interpret this fact. The first reaction is that we expect distribution to hold; that the Resource Model is somehow defective. It is worth remarking that the Trace Model does satisfy distribution of $p * (\cdot)$ through lubs. The second reaction is that there is nothing in program logic (say, CSL) which *forces* this law of distribution of $p * (\cdot)$ through lubs, and thus we needn't insist upon it. We add that neither does sequencing (;) in the Resource Model preserve all lubs (in its second argument). Sequential composition of predicate transformers is so basic that it is hard to argue that one should *demand* full distribution laws (useful though they are when present), as that would rule out a host of natural models.

5 On Assertions, Triples and Programs

At the beginning of the paper we asked whether program logic rules 'meant the same thing' in the algebra models as in familiar state-based models of pre/post specs. This section answers that question in the affirmative, and also provides further perspective on whether we should insist that assertions satisfy locality.

5.1 Connections in the Resource Model

We first show how to connect the Dijkstra triple to the Hoare and Plotkin triples in the Resource Model. We remind the reader of the definitions:

Hoare triple	$\{p\}\, c\, \{q\}$	$\Leftrightarrow$	$p; c \sqsubseteq q$
Plotkin triple	$p \rightarrow_c q$	$\Leftrightarrow$	$p \sqsupseteq c; q$
Dijkstra triple	$\langle\!\langle X \rangle\!\rangle\, F\, \langle\!\langle Y \rangle\!\rangle$	$\Leftrightarrow$	$X \subseteq FY$

Comparing these requires that we can turn the predicates X and Y in the relationship $X \subseteq FY$ defining the Dijkstra triple into predicate transformers, as the pre and post of Hoare and Plotkin triples are the same kinds of entities as the programs. We achieve this via $bpt[X, Y]$, the 'best predicate transformer' corresponding to precondition X and postcondition Y:

$$bpt[X, Y] \ = \ \lambda P.\, \text{if } Y \subseteq P \text{ then } X \text{ else } \emptyset.$$

It is easily seen that $bpt[X, Y]$ is the largest monotone transformer F such that $\langle\!\langle X \rangle\!\rangle\, F\, \langle\!\langle Y \rangle\!\rangle$. Using this definition, for a given X we can define $before[X]$ as

$$\mathit{before}[X] \;=\; \mathit{bpt}[\mathit{true}, X]$$

where true is the predicate Σ. We think of $\mathit{before}[X]$ as covering a 'past' in which X eventually becomes true, and this lets us characterize Dijkstra triples in terms of Hoare triples: It is not difficult to see that, in the Resource Model,

$$\{\mathit{before}[X]\}\, F\, \{\mathit{before}[Y]\} \quad \text{iff} \quad \langle\!\langle X \rangle\!\rangle\, F\, \langle\!\langle Y \rangle\!\rangle$$

In this sense, the algebraic Hoare triple subsumes the traditional state-based one. In fact, $\mathit{before}[]$ is not the only choice; the next result summarizes a few:

Proposition 5.1 *In the Resource Model*

$$\begin{aligned}\langle\!\langle X \rangle\!\rangle\, F\, \langle\!\langle Y \rangle\!\rangle &\Leftrightarrow \{\mathit{bpt}[\mathit{emp}, X]\}\, F\, \{\mathit{bpt}[\mathit{emp}, Y]\} \\ &\Leftrightarrow \{\mathit{bpt}[\mathit{true}, X]\}\, F\, \{\mathit{bpt}[\mathit{true}, Y]\} \\ &\Leftrightarrow \mathit{bpt}[X, \mathit{emp}] \rightarrow_F \mathit{bpt}[Y, \mathit{emp}] \\ &\Leftrightarrow \mathit{bpt}[X, \mathit{true}] \rightarrow_F \mathit{bpt}[Y, \mathit{true}]\end{aligned}$$

Here, $\mathit{bpt}[Y, \mathit{emp}]$ and $\mathit{bpt}[Y, \mathit{true}]$, characterizing the Plotkin triple, talk about the future rather than the past. For instance, we can think of $\mathit{bpt}[Y, \mathit{emp}]$, the largest transformer satisfying $\langle\!\langle Y \rangle\!\rangle - \langle\!\langle \mathit{emp} \rangle\!\rangle$, as an operation that cleans up memory, returning it to the operating system, after checking that Y holds.

It would be ideal if these characterizations were obtained using embeddings of the monoid $(P(\Sigma), *, \mathit{emp})$ of predicates into the locality bimonoid, and this is possible to achieve by choosing embeddings carefully. We observe that

$$\mathit{bpt}[X, Y] * \mathit{bpt}[X', Y'] = \mathit{bpt}[X * X', Y * Y']$$

So fixing one variable to an idempotent (such as emp or true) yields that the embedding of predicates into predicate transformers preserves $*$. If the idempotent is chosen to be emp, then the characterization of emp is $\mathit{bpt}[\mathit{emp}, \mathit{emp}] = \mathsf{nothing}$, and so in this case, the characterization preserves the unit as well.

Conceptually, there is no reason to demand that assertions are local in the same way that programs arguably should be. But as it turns out, in the Resource Model, the question of whether one faithfully recovers program logic in the algebra is independent of the question of whether all the elements are required to be local, i.e. one only considers local transformers.

The best predicate transformers are not necessarily local. An example is that $\mathit{bpt}[\mathit{emp}, \mathit{emp}] = \mathsf{nothing}$. However, we can use localization to define the best local transformer $\mathit{blt}[X, Y] = \mathit{bpt}[X, Y] * \mathsf{skip}$ which also happens to characterize triples in the sense that for F local

$$\begin{aligned}\langle\!\langle X \rangle\!\rangle\, F\, \langle\!\langle Y \rangle\!\rangle &\Leftrightarrow \{\mathit{blt}[\mathit{emp}, X]\}\, F\, \{\mathit{blt}[\mathit{emp}, Y]\} \\ &\Leftrightarrow \mathit{blt}[X, \mathit{emp}] \rightarrow_F \mathit{blt}[Y, \mathit{emp}]\end{aligned}$$

5.2 Recursion Rule

We now give a sufficient condition for modelling the Recursion Rule in locality bimonoids. The best transformer $\mathit{blt}[X, Y]$ is an example of a *greatest satisfier* for a predicate, in this case the predicate $\langle\!\langle X \rangle\!\rangle - \langle\!\langle Y \rangle\!\rangle$. This idea is used in our sufficient condition:

Proposition 5.2 (Recursion Rule) *If the order of a locality bimonoid G is a complete lattice, and if greatest local satisfiers for triples exist, then we have the usual Hoare logic proof rule for recursive (parameterless) procedures.*

Recursion Rule: *For all $F, H \in [G_{loc} \to G_{loc}]$*

$$\begin{array}{rl} & \{a\}\, x\, \{b\} \Rightarrow \{a\}\, F\ x\, \{b\} \\ \wedge & \{a\}\, x\, \{b\} \Rightarrow \{p\}\, H\ x\, \{q\} \\ \Longrightarrow & \{p\}\, H(\mu F)\, \{q\} \end{array}$$

where $[A \to B]$ denotes the monotone function space and μF is the least fixed point of F.

The Hoare triples can be replaced with any downwards closed predicate for which greatest local satisfiers exist (say, Plotkin or Dijkstra triples).

Proof. Since F is monotone it has a least (local) fixed point μF, given by

$$\mu F = \bigsqcap \{x \mid Fx \sqsubseteq x\}$$

Here x ranges over local elements and so μF is the greatest local lower bound of local prefixed points, besides being a fixed point.

We will argue that $\{a\}\, \mu F\, \{b\}$. Let s be the greatest local satisfier of $\{a\} - \{b\}$. Since s is a local satisfier, we know by assumption that $F(s)$ is a local satisfier. Since s is the greatest local satisfier, we know that $F(s) \sqsubseteq s$. Thus, s is a local prefixed point. This means that $\mu F \sqsubseteq s$, as μF is a lower bound of local prefixed points. We now use that satisfaction is downwards closed to conclude that $\{a\}\, \mu F\, \{b\}$. (This is easily seen as $a\,;\mu F \sqsubseteq a\,; s \sqsubseteq b$.)

Thus, by assumption, $H(\mu F)$ is a local satisfier of $\{p\} - \{q\}$. □

The statement in this lemma is another way of describing the usual Hoare proof rule for recursive parameterless procedures

$$\frac{\{A\}\, X\, \{B\} \vdash \{A\}\, F\, \{B\} \qquad \{A\}\, X\, \{B\} \vdash \{P\}\, H\, \{Q\}}{\vdash \{P\}\, letrec\ X = F\ in\ G\, \{Q\}}$$

where, by the previous results, this rule applies when commands are built using sequencing, composition, nondeterministic choice, and procedure declarations.

In the Resource Model we have greatest local satisfiers for Dijkstra triples, given by $blt[X, Y]$, and satisfaction for Dijkstra triples is downwards closed, so this provides a model for CSL with recursion.

Hoare triples, however, are a bit more general. To obtain greatest local satisfiers for the Hoare triple $\{p\} - \{q\}$, we look to

$$\bigsqcup \{c \mid p\,; c \sqsubseteq q\}$$

But this is not guaranteed to be a satisfier, because ; does not distribute through $\bigsqcup$ in its second argument.

As it turns out, ; *does* distribute through $\bigsqcup$ in its first argument, and so the Resource Model models recursion for Plotkin triples. Seeing as the Resource Model is built on *backwards* predicate transformers, it is perhaps natural to

expect the Plotkin triples to be more well-behaved than the Hoare triples. We shall just note here that with of definition of ; suitable for forward predicate transformers, it would distribute through $\bigsqcup$ in its second argument.

The Trace Model, being a quantale, models the Recursion Rule for both Hoare and Plotkin triples.

6 Conclusion

In this paper we have described links between a standard model of Concurrent Separation Logic, the Resource model, and algebraic models inspired by the recent Concurrent Kleene Algebra. By looking at such a concrete, and previously-existing, model we hope to have shown that the notion of locality in the algebra generalizes the understanding obtained from the concrete resource-based semantics of Separation Logic.

The algebraic structure used in this paper admits both state-based and history-based models (the Resource and Traces models), and this exemplifies the unifying nature of the algebra, which allows principles to be stated independently of the syntax in specific models. Precursor works even used 'true concurrency' in addition to interleaving models, giving further evidence of the generality [6, 8]. In fact, we ended up with a weaker structure than CKA, and perhaps this paper will also have some input into further developments of algebraic models for concurrency. The most pressing issues going forward include axiomatization of further primitives (e.g., distinguishing internal and external choice), exploring a wider range of concrete models, and determining the practical significance of the generalized program logic in any of the concrete models.

Acknowledgements. We are grateful to the referees for perceptive comments. Hussain, O'Hearn, Petersen and Struth were supported by the EPSRC. O'Hearn acknowledges the support of a Royal Soceity Wolfson Research Merit award and a gift from Microsoft Research.

References

1. http://staffwww.dcs.shef.ac.uk/people/G.Struth/isa/.
2. S. L. Bloom and Z. Ésik. Free shuffle algebras in language varieties. *TCS*, 163(1&2):55–98, 1996.
3. S. D. Brookes. A semantics of concurrent separation logic. *TCS*, 375(1-3):227–270, 2007. (Prelim. version appeared in CONCUR'04).
4. C. Calcagno, P.W. O'Hearn, and H. Yang. Local action and abstract separation logic. In *LICS*, pages 366–378. IEEE Computer Society, 2007.
5. E. W. Dijkstra. *A discipline of programming.* Prentice-Hall series in automatic computation. Prentice-Hall, 1976.
6. J. L. Gischer. The equational theory of pomsets. *TCS*, 61:199–224, 1988.
7. C. A. R. Hoare and He Jifeng. *Unifying Theories of Programming.* Prentice-Hall series in computer science. Prentice-Hall, 1998.

8. T. Hoare, B. Möller, G. Struth, and I. Wehrman. Concurrent Kleene algebra and its foundations. *Journal of Logic and Algebraic Programming*, 2011. (Prelim. version in CONCUR'09).
9. D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Inf. Comput.*, 110(2):366–390, 1994.
10. P. W. O'Hearn. Resources, concurrency and local reasoning. *TCS*, 375(1-3):271–307, May 2007. (Prelim. version appeared in CONCUR'04).
11. H. Yang and P. W. O'Hearn. A semantic basis for local reasoning. In *5th FOSSACS*, 2002. pp402-416.