Journal of Logical and Algebraic Methods in Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

On the relation between Concurrent Separation Logic and Concurrent Kleene Algebra

Peter W. O'Hearn [a], Rasmus L. Petersen [b], Jules Villard [a,*], Akbar Hussain [c]

[a] University College London, UK
[b] Microsoft Research Cambridge, UK
[c] GSM London, UK

ARTICLE INFO

Article history:
Received 4 June 2013
Received in revised form 2 April 2014
Accepted 1 August 2014
Available online xxxx

Keywords:
Compositionality
Concurrency
Separation Logic
Kleene Algebra

ABSTRACT

We investigate the connection between a general form of Concurrent Separation Logic (CSL), a logic for modular reasoning about concurrent programs, and Concurrent Kleene Algebra (CKA), which provides an axiomatic approach to models of concurrency. We show how the proof theory of a general form of CSL can be embedded in a variation on the notion of CKA. Our embedding, however, induces models of a particular form based on predicate transformers. We also investigate the relation between concrete models of CSL based on interleaving of traces and CKA. We find, curiously, that the validity of CSL's Concurrency proof rule in these models does not follow from or otherwise utilize CKA structure, but that a CKA structure exists nonetheless which can give a different model of the CSL proof rules.

Our results can be read as providing a completeness theorem showing a sense in which nothing is missing as far as proof power goes in (a variant on) the notion of CKA, while at the same time showing that CKAs impose constraints that rule out some natural CSL models.

1. Introduction

The topic of modular reasoning about concurrent programs has seen great progress in the past decade, with the result that intricate programs are proven much more efficiently (indeed, at all) see, *e.g.*, [34,15,44,45,31,6,7,12,20,38] for a selection. But the flourishing of ideas in the area has resulted in diversity of technique, and the question arises as to whether system can be brought to the developments by finding general principles which underpin these techniques.

The purpose of this paper is simple: to try to better understand the relation between two approaches which have sought such general principles: (abstract forms of) Concurrent Separation Logic (CSL) and Concurrent Kleene Algebra (CKA). We are not ready to propose a theory underpinning all of the above-mentioned developments. But we hope that the modest goal of comparing the generalist theories that we do have can pinpoint some of the strengths and weaknesses in them, and point to directions for future work.

Before describing our technical results we provide some context for the paper by describing background information on CSL and CKA.

* Corresponding author.
E-mail address: j.villard@ucl.ac.uk (J. Villard).

http://dx.doi.org/10.1016/j.jlamp.2014.08.002

1.1. Context on CSL

Concurrent Separation Logic (CSL [34]) is a logic for modular reasoning about concurrent processes, in which the proof rule

$$[\text{Concurrency}]\ \frac{\{p_1\}c_1\{q_1\} \quad \{p_2\}c_2\{q_2\}}{\{p_1 * p_2\}c_1 \parallel c_2\{q_1 * q_2\}}$$

allows one to reason independently about concurrent processes. Given Hoare triples for c_1 and c_2, the rule allows a conclusion about their concurrent effect on separate storage, described by the separating conjunction $*$ in the precondition $p_1 * p_2$ and postcondition $q_1 * q_2$.

CSL was presented first as a logic for heap mutation based on a model of $*$ that separates heaps, represented as partial functions from memory locations to values, into domain-disjoint sub-heaps. Abstract Separation Logic (ASL [5]) investigated a generalized logic, following Pym's original position that bunched logic (the logic underlying separation logic) can be understood generally in terms of resource instead of particularly in terms of memory [35,39]. ASL presumed a partial commutative monoid of resource composition in place of the particular monoid of heap partitioning, and the semantics of propositions was obtained by lifting resource composition to a $*$ connective on a powerset Boolean algebra. Examples of models different from pure separation include permission models allowing read sharing between concurrent processes [3] and a system that allows compositional reasoning about some uses of auxiliary variables [32].[1]

Remarkably, in [13] it is argued that the above proof rule based on $*$ can, with a non-standard model of separation logic, represent rely/guarantee reasoning; this is striking because the CSL Concurrency rule appears to be about non-interference, while the rely/guarantee proof rules are about interference. It has become clear that, with non-standard models distant from the original heap model, CSL-style reasoning about concurrency is much more general than first envisaged. Gardner has described the phenomenon as one of a "fiction of separation", where we can reason independently about processes that access and even mutate shared resources. These realizations by Gardner, Parkinson and others have spawned specific new logics [12,29,41] and a further generalization beyond ASL, called the Views framework [11].

In this paper we will consider a system ASL^{--}, which is intermediate between Views and ASL. This choice is driven by the desire to connect to CKA. Thus, our study will also help to pinpoint some of the similarities and differences between CKA and Views.

1.2. Context on CKA

Hoare and colleagues have been investigating an approach which shares some of the same aspirations as generalizations of CSL: a general approach to compositional reasoning about concurrency [24]. We remark right away that this is not the sole aim of CKA and relatives – in particular, it is emphasized that algebra provides a tool to unify various forms of semantics [26,25] – but the scope of this paper is limited to its relation to program logic.

Although it shares some of the aspirations of generalizations of CSL, the formal setup of the work on CKA is extremely different. A CKA is a complete lattice with operators for sequential and concurrent composition satisfying a collection of laws. The most important one is an ordered version of the exchange law from 2-categories and links parallel and sequential composition:

$$(p \parallel r); (q \parallel s) \sqsubseteq (p; q) \parallel (r; s).$$

CKA does not have Hoare triples as a primitive notion, but they are defined in terms of other data, as follows:

$$\{p\}c\{q\} \quad \Leftrightarrow \quad p; c \sqsubseteq q.$$

A significant point is that here the pre and post p and q are regarded as the same kind of entity as the program c. This can be jarring at first to those used to thinking of the p's and q's as state assertions, but it is not difficult to reconcile the two views in the case that p and q are "assume statements" $assume(P)$ and $assume(Q)$ where P and Q are state assertions (intuitively, $assume(P)$ is like a program that creates or allocates state satisfying P). Another way to understand this form of triples, going beyond the state reading, is in terms of history: if p and q describe histories up to a given point in time, then the triple says that the history q contains or overapproximates the history p followed by the behavior c. For this reason we call it the historic interpretation of triples. This historical reading chimes well, at least intuitively, with the use of past-time reasoning in proofs about highly-concurrent algorithms [36,16,19].

The remarkable fact is that, with the exchange law and the historic triples, one begins to get a link-up with CSL:

$$\frac{\dfrac{p_1; c_1 \sqsubseteq q_1 \qquad p_2; c_2 \sqsubseteq q_2}{(p_1; c_1) \parallel (p_2; c_2) \sqsubseteq q_1 \parallel q_2}\ \parallel\ \textit{Monotone}}{(p_1 \parallel p_2); (c_1 \parallel c_2) \sqsubseteq q_1 \parallel q_2}\ \textit{Exchange}$$

[1] Another extension of the models of (this time sequential) separation logic is Algebraic Separation Logic [8], which operates at the level of propositions, taking them to be any Boolean quantale, thus generalizing the case of propositions over stacks and heaps.

Please cite this article in press as: P.W. O'Hearn et al., On the relation between Concurrent Separation Logic and Concurrent Kleene Algebra, Journal of Logical and Algebraic Methods in Programming (2014), http://dx.doi.org/10.1016/j.jlamp.2014.08.002

If we squint, and think of the occurrences of $\|$ in $p_1 \| p_2$ and $q_1 \| q_2$ as $*$, then this derivation gives us the Concurrency proof rule of CSL.

Just as in the work on ASL and Views, CKA has demonstrated models far removed from the original heap model of CSL. One CKA is based on sets of traces, with $\|$ being interleaving and ; concatenation. Another is partially-ordered sets, where $\|$ is a notion of parallel composition and ; is weak sequential composition (which allows statement reordering according to dependence). Although these examples have no concept of resource separation built in, they formally provide models of the CSL Concurrency rule, and its relative the Frame rule, which raises the question of whether we might have modular proof methods even more widely applicable than envisaged in the generalizations of CSL mentioned above.

1.3. This work

The situation just sketched is tantalizing – as to the possibilities for a further-reaching theory of modular reasoning about processes – and at the same time puzzling – as to the merits, strengths and weaknesses of the separate theories. Our basic aim in this work is to better understand the relationship between these approaches.

The first step we make to enable this connection is to level the playing field. Both ASL and Views are presented in terms of a particular interleaving model which is biased towards shared memory or asynchronous communication, where CKA makes no such commitment. The interleaving models of ASL and Views limit their generality significantly, because there are many interesting models that do not fit into their setups; partial order models, models of synchrony, etc. To make a comparison we simply forget about the particular interleaving models and focus on proof theory; Views and ASL use cut-down and generalized versions of Concurrent Separation Logic. We can then compare the proof theory of the program logic with that derived from the algebraic laws of CKA.

As we mentioned above, CKA has demonstrated models far removed from those typically used to study CSL. Despite this diversity of models, CKA is not necessarily more general than the CSL proof theories. In particular, CKA has never been thoroughly related to the concrete models employed in the prior works on ASL and Views or other work on CSL, and the induced proof theory of CKA has not been thoroughly related to that of variants of CSL. CKA gives rise to a particular collection of models, but which models? Could it be that, despite its abstractness, CKA is *overly constraining*, so that it is not applicable to the particular models of separation logics?

The pivotal point is the assumption in CKA that $*$ and $\|$ be one and the same. This unification leads to a remarkable economy in the theory, but it is also a strong assumption. While there are some models, and interesting models, that satisfy it, the question is whether this assumption, and resultant economy, come at a price. Our answer will be yes and no.

In this paper we consider a more general logic than ASL, where there is only a quantale of propositions; *i.e.*, a complete lattice with an ordered monoid where $*$ preserves all lubs in both arguments. This logic, which we call ASL^{--}, assumes less than ASL in that it forgets the Boolean structure. It also does not include specialized proof rules for critical regions or the Hoare logic proof rule for conjunction, which ASL does. (ASL^{--}'s weakness means that it admits versions of CSL which do not include the rule of conjunction, which are being seen increasingly [18,17,42,12].)

We first show how the proof theory of ASL^{--} can be embedded in a variation on the notion of CKA. We weaken the definition of CKA slightly for this but the weakening maintains the general spirit and maintains the connection between $*$ and $\|$. We establish a completeness theorem showing a sense in which nothing is missing as far as proof power goes in (this weakening of) CKA.

Our general embedding, however, induces models of a particular form based on predicate transformers. We also investigate the relation between concrete models of CSL based on interleaving of traces and CKA. We find, curiously, that the interpretation of the Concurrency rule in these CSL models does not follow that of CKAs, but that a different CKA structure can be found within them which can give a different interpretation of the CSL proof rules. This shows that CKAs impose constraints that rule out some natural CSL models. In particular, models of CSL enjoy the freedom to interpret $*$ and $\|$ differently.

Our study will lead us to propose questions for future work, detailed at the end of the paper.

2. An Abstract Concurrent Separation Logic (ASL^{--})

In this section, we define a version of the Concurrent Separation Logic (CSL) proof system, ASL^{--}, which abstracts from the exact details of primitive commands or pre/post specs. We assume that we are given the following structures.

- A quantale $(\mathit{Props}, \leq, *, \mathit{emp})$ of propositions; a quantale is a complete lattice equipped with a commutative monoid where the multiplication $*$ preserves all joins in both arguments (as a consequence, it is monotone). We use the notations $\vee$ and $\bigvee$ for joins of propositions.
- A set Com closed under operations $c_1 \| c_2$, $c_1; c_2$, $c_1 + c_2$ and $\mathsf{iterate}(c)$, and with a distinguished element $\mathtt{skip} \in \mathit{Com}$.

Given the above data, and a collection *Axioms* of Hoare triples $\{p\}c\{q\}$, we write $\mathit{Axioms} \vdash \{p\}c\{q\}$ to mean that $\{p\}c\{q\}$ is derivable from *Axioms* and the following proof rules.

Please cite this article in press as: P.W. O'Hearn et al., On the relation between Concurrent Separation Logic and Concurrent Kleene Algebra, Journal of Logical and Algebraic Methods in Programming (2014), http://dx.doi.org/10.1016/j.jlamp.2014.08.002

PROOF RULES FOR ASL^{--}

$$[\text{Frame}]\ \frac{\{p\}c\{q\}}{\{p * r\}c\{p * r\}} \qquad [\text{Concurrency}]\ \frac{\{p_1\}c_1\{q_1\} \quad \{p_2\}c_2\{q_2\}}{\{p_1 * p_2\}c_1 \parallel c_2\{q_1 * q_2\}}$$

$$[\text{Skip}]\ \frac{}{\{p\}\texttt{skip}\{p\}} \qquad [\text{Seq}]\ \frac{\{p\}c_1\{q\} \quad \{q\}c_2\{r\}}{\{p\}c_1; c_2\{r\}}$$

$$[\text{Nondet}]\ \frac{\{p\}c_1\{q\} \quad \{p\}c_2\{q\}}{\{p\}c_1 + c_2\{q\}} \qquad [\text{Iteration}]\ \frac{\{p\}c\{p\}}{\{p\}\mathsf{iterate}(c)\{p\}}$$

$$[\text{Disjunction}]\ \frac{\{p_i\}c\{q\},\ \text{all } i \in I}{\{\bigvee_{i\in I} p_i\}c\{q\}} \qquad [\text{Consequence}]\ \frac{p' \leq p \quad \{p\}c\{q\} \quad q \leq q'}{\{p'\}c\{q'\}}$$

This structure is more general than that of Abstract Separation Logic, which assumes that propositions also have a Boolean algebra structure which, furthermore, are powerset Boolean algebras. It is also more general that Algebraic Separation Logic [8], which assumes the quantalic structure over propositions to be Boolean and does not have the Concurrency rule. It is similar to the system named BCSL in Hussain's thesis [27] but Hussain assumes only an ordered monoid of propositions and had no Disjunction rule. Hussain's system is in turn similar to but less general than the Views proof system [11], which does not require that entailment (here, $\leq$) is reflexive or transitive. Views does not presume Disjunction either, and it includes a new inference rule: the generalized Frame rule.

We have presumed a quantale of propositions, and not only an ordered monoid, in order to make a tight connection with CKA. In particular, CKAs validate the Disjunction rule above (as we shall see in Theorem 3.3).

We have not included the Conjunction rule

$$[\text{Conjunction}]\ \frac{\{p\}c\{q_i\},\ \text{all } i \in I}{\{p\}c\{\bigwedge_{i\in I} q_i\}} \quad I \neq \emptyset.$$

There are some models of variations on Concurrent Separation Logic that do not validate this rule. The relation of this rule to CKA and our embedding of ASL^{--} into it is delicate, and discussed further in Sections 3 and 4.

RAM instantiation. Throughout the paper we will draw upon the standard model of separation logic in examples. The RAM model of computation uses natural numbers as addresses and stores natural numbers. We will work with partial RAMs, or heaps: finite pieces of RAM. A partial RAM can be thought of as representing the portion of memory that a process "owns" or "has the right to access" [34]. In the terminology of [11], this can be thought of as providing a "view" of a more concrete, global memory. It is useful to keep in mind that the partial RAMs are an abstraction put on top of a runtime model, but we do not technically need the global memory to formulate the logic.

The structure of propositions is obtained by setting

$$(\mathit{Props}, \leq, *, \mathit{emp}) = \big(\mathcal{P}(\mathit{Heaps}), \subseteq, *, \{u\}\big)$$

where the material to the right is defined as follows: *Heaps* is the set $\mathbb{N} \rightharpoonup_f \mathbb{N}$ of finite partial functions from natural numbers to natural numbers, and $\mathcal{P}(\mathit{Heaps})$ its powerset; u is the empty partial function. Let $h \bullet h'$ denote the union of heaps with disjoint domains, which is undefined when the domains overlap. We write $h \sharp h'$ when $h \bullet h'$ is defined. Then, $X * Y = \{h_X \bullet h_Y \mid h_X \in X \wedge h_Y \in Y \wedge h_X \sharp h_Y\}$.

A typical predicate is the points-to fact $n \mapsto m$, where n and m are natural numbers. It is the singleton set $\{h\}$ where h is the heap that maps n to m and which is undefined on all numbers other than n. Another predicate is $n \mapsto -$, which is $\bigcup_m n \mapsto m$.

A typical command is the mutation statement $[n] := m$ where n and m are natural numbers. Associated with it is the following Hoare triple axiom:

$$\frac{}{\{n \mapsto -\}[n] := m\{n \mapsto m\}}\ .$$

A consequence of the use of $*$ in the precondition for the parallel rule is that a proved program will exhibit no race from any state satisfying the precondition. As an extreme example, for the program

$$[10] := 23 \ \parallel\ [10] := 44$$

we cannot find any consistent precondition whatsoever. The reason is that each process needs a precondition $10 \mapsto -$ according to the axiom for $[n] := m$ given above, and $10 \mapsto - \ * \ 10 \mapsto -$ is *false*.

At this stage, having given the model for preconditions and postconditions and some axioms, but not a model of programs, we have not committed yet whether to have the Conjunction rule in the proof theory for the RAM model. The model of Section 5 will validate it, while there exist others that do not.

Please cite this article in press as: P.W. O'Hearn et al., On the relation between Concurrent Separation Logic and Concurrent Kleene Algebra, Journal of Logical and Algebraic Methods in Programming (2014), http://dx.doi.org/10.1016/j.jlamp.2014.08.002

3. Concurrent Kleene Algebra (CKA)

As a precursor to the notion of CKA we define a weaker structure.

Definition 3.1 *(Concurrent monoids).* A *concurrent monoid* $(M, \sqsubseteq, \|, ;, \mathsf{skip})$ is a partially ordered set $(M, \sqsubseteq)$ hosting two ordered monoids $(M, \|, \mathsf{skip})$ and $(M, ;, \mathsf{skip})$ sharing the same unit, with $\|$ commutative, and satisfying the ordered exchange law [24]

$$(p \,\|\, r); (q \,\|\, s) \sqsubseteq (p; q) \,\|\, (r; s).$$

A concurrent monoid is *complete* if $(M, \sqsubseteq)$ is a complete lattice.

Definition 3.2 *(CKA and weak CKA).*

1. A *Concurrent Kleene Algebra*, or *CKA* for short, is a complete concurrent monoid where $\|$ and $;$ preserve all joins in both arguments.[2]
2. A *weak CKA* is a complete concurrent monoid together with a subset $A \subseteq M$ such that (i) $\mathsf{skip} \in A$; (ii) A is closed under $\|$ and all joins; (iii) the domain-restriction of $\|$ to elements of A preserves all joins in both arguments; (iv) for each $a \in A$ the function $a; (\cdot) : M \to M$ preserves all joins; and (v) for each $m \in M$ the function $(\cdot); m : A \to M$ (the domain-restriction of $(\cdot); m : M \to M$ to elements of A) preserves all joins.[3]
3. A CKA or weak CKA is *Boolean* if its lattice is a Boolean algebra, and *Intuitionistic* if its lattice is a Heyting algebra.

We use $\bigsqcap$ and $\bigsqcup$ to notate joins and meets in CKAs and relatives (to distinguish them from $\bigwedge$ and $\bigvee$, which we use for propositions in ASL^{--}).

The set A in the definition of a weak CKA is intended to represent a set of *assertions*, which has more structure than the weak CKA as a whole; in particular, A forms a quantale. This is similar to the role of tests in Kleene Algebra with tests [30], except we do not assume A to be Boolean.

An example of a weak CKA which is not a CKA is constructed in Section 4.

Example: linear trace model. If A is a set then let $Tr(A)$ be the set of finite and infinite sequences (of length at most ω) of elements from A. If $t_1, t_2 \in Tr(A)$ then let $t_1; t_2$ denote their concatenation if t_1 is finite, and t_1 if it is infinite, and let $t_1 \,\|\, t_2$ denote the set of their interleavings defined as:

$$t_1 \,\|\, t_2 = \left\{(a_i)_{i\in I} \mid \exists I_1 \uplus I_2 = I.\, \forall j \in \{1,2\}.\, t_j = (a_i)_{i\in I_j}\right\}.$$

We can lift these to trace sets in the usual way, so that $T_1 \,\|\, T_2 = \bigcup\{t_1 \,\|\, t_2 \mid t_1 \in T_1 \wedge t_2 \in T_2\}$ and $T_1; T_2 = \{t_1; t_2 \mid t_1 \in T_1 \wedge t_2 \in T_2\}$. Then $(Tr(A), \subseteq, \|, ;, \{\epsilon\})$ is a Boolean CKA.

Let $Tr{\Downarrow}(A)$ denote the set of downwards closed subsets of $Tr(A)$ w.r.t. the prefix order on traces. Then $(Tr{\Downarrow}(A), \subseteq, \|, ;, \{\epsilon\})$ is an intuitionistic CKA.

In either case there is a tremendous amount of logical structure: a complete Heyting or Boolean algebra, enough to interpret classical predicate logic, a non-commutative substructural logic similar to Lambek's, and a commutative substructural logic similar to the multiplicative fragment of Linear Logic.

The linear traces model formalizes the historic reading of triples $p; c \sqsubseteq q$ alluded to in the introduction, where $p; c \sqsubseteq q$ says that q's description of the past (a set of traces) covers that of $p; c$.

Non-example: fairness model. We change $\|$ in the linear traces model so that $t_1 \,\|\, t_2$ considers only the fair interleavings. We do not give a precise definition [37], but will use only that if t is finite and a^ω is an infinite sequence of a's, then any trace in $a^\omega \,\|\, t$ must include all of t. We claim that

$$\left(a^\omega \,\|\, b\right); (c \,\|\, d) \not\sqsubseteq \left(a^\omega; c\right) \,\|\, (b; d).$$

To see why, note that $ba^\omega \in (a^\omega \,\|\, b); (c \,\|\, d)$, but that $ba^\omega \notin (a^\omega; c) \,\|\, (b; d)$; the reason is that a fair interleaving t of $(a^\omega; c)$ and $(b; d)$ must be such that both b and d appear somewhere in t, and this is not the case for ba^ω.

We give this negative example not because fairness is a particular interest in this paper, but only to be clear that the algebra does impose constraints which are not automatically true in all models of concurrency that have been considered.

[2] What we call "CKA" here is called "concurrent quantale" in [24], as their notion of CKA does not require a complete lattice. Also, the exchange law is defined there as: $(p \,\|\, r); (q \,\|\, s) \sqsubseteq (r; q) \,\|\, (p; s)$, which encodes commutativity of $\|$ as well. Since we explicitly require that $\|$ is commutative, the two formulations of the law are equivalent.

[3] A similar structure was given this name in [33]. In that definition a Kleene star is assumed axiomatically, while here iteration is built out of the lattice structure on the carrier sets.

Please cite this article in press as: P.W. O'Hearn et al., On the relation between Concurrent Separation Logic and Concurrent Kleene Algebra, Journal of Logical and Algebraic Methods in Programming (2014), http://dx.doi.org/10.1016/j.jlamp.2014.08.002

Technical fine points concerning meets (and Conjunction law). A CKA is a more elegant structure than a weak CKA, but has stronger constraints. A few comments are in order on some of the fine points on CKAs versus weak CKAs.

First, because the "assertions" A in a weak CKA are closed under all joins from M and $\|$, and include skip, they form a quantale. Since assertions have all joins they also have all meets as is standard, but the meets need not be calculated the same as in M. We have $a \,\&\, b \sqsubseteq a \sqcap b$ where & is the meet in A and $\sqcap$ is the meet in M, but not the reverse. This means that the Conjunction rule of Hoare logic is valid in a CKA, but not in a weak CKA. To see why, suppose we require that the pre and post are always drawn from A in the historic triple $p; c \sqsubseteq q$ where c is an arbitrary element of M. Then we cannot make the inference

$$\frac{p; c \sqsubseteq q_1 \quad p; c \sqsubseteq q_2}{p; c \sqsubseteq q_1 \,\&\, q_2} \ \textit{Incorrect}$$

because $p; c$ might not be in A: since A could be such that $q_1 \sqcap q_2 \sqsupset q_1 \,\&\, q_2$, if $p; c = q_1 \sqcap q_2$ then the conclusion is invalid. However, we can conclude

$$\frac{p; c \sqsubseteq q_1 \quad p; c \sqsubseteq q_2}{p; c \sqsubseteq q_1 \sqcap q_2} \ \textit{Correct}.$$

For all we know, $q_1 \sqcap q_2$ just might take us outside of the assertions A, so we no longer have a "legal Hoare triple" because the postcondition is not an assertion (for a concrete example of this situation see the discussion just above Proposition 4.11).

Second, for the purpose of interpreting program logic under the historic triples we need joins preservation for $\|$ or for ; on the left to hold for assertions only, and not necessarily for all elements. Preservation of joins on the right for ;, though, is needed for validating the proof rule for iteration (see the proof of Theorem 3.3 below). The stronger laws, such as $(c \sqcup d) \,\|\, e = (c \,\|\, e) \sqcup (d \,\|\, e)$ can be handy, but are not necessary.

Interpretation of commands. The program constructs are modeled in a weak CKA by assuming that primitive commands are given an interpretation as elements of the weak CKA, and by sending the ; and $\|$ constructs in ASL^{--} to their counterparts in the (weak) CKA, and $+$ to $\sqcup$. We define iteration according to the usual least fixed-point approach: if $c \in M$ then take the Tarski definition of the least fixed-point of the function $i_c = \lambda m.\ \mathsf{skip} \sqcup c; m : M \to M$:

$$\mathsf{iterate}(c) = \bigsqcap \{m \mid (\mathsf{skip} \sqcup c; m) \sqsubseteq m\}.$$

Theorem 3.3 *(Model of ASL^{--}). Suppose we have a weak CKA $(M, \sqsubseteq, \|, ;, \mathsf{skip})$ with $A \subseteq M$. Define the quantale of ASL^{--} propositions $(Props, \leq, *, emp)$ to be $(A, \sqsubseteq, \|, \mathsf{skip})$, and define the semantics of Hoare triples by*

$$\text{"}\{p\}c\{q\} \textit{ is valid"} = p; c \sqsubseteq q$$

where $c \in M$ and $p, q \in A$. Then the proof rules of ASL^{--} are sound (preserve validity) according to this interpretation.

Proof. The proof for the Frame, Concurrency, Non-Determinism, Sequencing and Consequence rules is as in [24].

For the Disjunction rule, assume that $p_i; c \sqsubseteq q$ for all $i \in I$. Then, recalling that $(\cdot); c : A \to M$ preserves all joins,

$$\Big(\bigsqcup_{i \in I} p_i\Big); c = \bigsqcup_{i \in I} (p_i; c) \sqsubseteq q.$$

For the Iteration rule, we will argue $p; \mathsf{iterate}(c) \sqsubseteq p$, assuming that $p; c \sqsubseteq p$ (using the definition of iterate above). First we observe that, since $p; (\cdot)$ preserves all joins, it has a right adjoint $p \leftarrow (\cdot)$, given by the formula $p \leftarrow q = \bigsqcup\{m \mid p; m \sqsubseteq q\}$. Second we observe that p is the least fixed-point of the function $f_p = \lambda m.\ p \sqcup m$ and that $(p; (\cdot)) \circ i_c \sqsubseteq f_p \circ (p; (\cdot))$, as for all m

$$\big((p; (\cdot)) \circ i_c\big)m = (p; \mathsf{skip}) \sqcup (p; c; m) \sqsubseteq p \sqcup (p; m) = \big(f_p \circ (p; (\cdot))\big)m$$

where the middle step uses that $p; c \sqsubseteq p$ and that ; is monotone.

All this allows us to use the fusion rule (see, *e.g.*, [2]) to conclude that $(p; (\cdot))(\mu i_c) \sqsubseteq \mu f_p$, *i.e.* that $p; \mathsf{iterate}(c) \sqsubseteq p$. □

In fact, a weaker notion of CKA still would suffice for interpreting ASL^{--}, one where the exchange law

$$(a \,\|\, r); (b \,\|\, s) \sqsubseteq (a; b) \,\|\, (r; s)$$

held only for "assertions" $a, b \in A$, and not for arbitrary elements of M. Our embedding result will happen to produce a situation where the general exchange law holds, but where $a; (\cdot)$ preserves joins for assertions a and where $p; (\cdot)$ does not for general p.

Please cite this article in press as: P.W. O'Hearn et al., On the relation between Concurrent Separation Logic and Concurrent Kleene Algebra, Journal of Logical and Algebraic Methods in Programming (2014), http://dx.doi.org/10.1016/j.jlamp.2014.08.002

4. Completeness of ASL^{--} in weak CKAs

For the purpose of the completeness result of this section we will assume that the commands are freely built using non-determinism, sequencing, iteration and parallelism, from primitive commands c_{prim}, and that the set of *Axioms* refers to the primitive commands only. We shall see that the semantics of primitive commands is obtained from their corresponding axioms in a natural way (see Proposition 4.13).

Theorem 4.1 *(Completeness). Given an ASL^{--} theory, there is a weak CKA and interpretation $[\![\cdot]\!]$ of commands and propositions such that*

$$\forall p, c, q.\ \mathit{Axioms} \vdash \{p\}c\{q\} \quad \mathit{iff} \quad [\![p]\!]; [\![c]\!] \sqsubseteq [\![q]\!].$$

Our proof will work as follows. From an ASL^{--} theory we will construct a model based on predicate transformers over the ASL^{--} propositions. The predicate transformer lattice has much more structure than the (syntactic) commands, and we will find a weak CKA within it. Then, we will prove that the usual predicate transformer semantics of Hoare triples, the "Dijkstra triples", coincides with the historic interpretation in CKA, and with provability.

The results in this section include and extend results on predicate transformers from our preliminary conference paper [23] and from Hussain's thesis [27].

4.1. Concurrent monoid of local predicate transformers

Throughout this section we presume a fixed quantale $(\mathit{Props}, \leq, *, \mathit{emp})$ and a collection *Axioms* of Hoare triples. We define several operations on the monotone function space $[\mathit{Props} \to \mathit{Props}]$ of *predicate transformers* (an introduction to predicate transformer semantics can be found in [1]). In each case the input parameter q is an element of *Props*.

$$(F_1 \parallel F_2)q = \bigvee\{F_1 q_1 * F_2 q_2 \mid q_1 * q_2 \leq q\}$$
$$(F_1 + F_2)q = F_1 q \wedge F_2 q$$
$$(F_1; F_2)q = F_1\big(F_2(q)\big)$$
$$\mathtt{skip}\, q = q$$

The definition of $F_1 \parallel F_2$ follows the proof theory of ASL^{--} in that the clause $F_1 q_1 * F_2 q_2 \mid q_1 * q_2 \leq q$ is tantamount to using the rule of consequence on the right together with the proof rule for parallelism. The $\bigvee$ is taking the disjunction of all preconditions that can be generated in this way, using the quantalic join for disjunction, so as to define as weak a precondition as possible.

Our ordering $\sqsubseteq$ on predicate transformers is the inverse point-wise order:

$$F \sqsubseteq G \iff \forall q.\ Fq \geq Gq.$$

Defining the order of the predicate transformers in this way allows us to characterize the order in terms of fault-avoiding triples for partial correctness in the standard way, where if $\{p\}c\{q\}$ and $c' \sqsubseteq c$ then $\{p\}c'\{q\}$. That is we interpret a triple as $p \leq \mathit{wlp}(c, q)$, where $\mathit{wlp}(c, \cdot)$ is a weakest liberal precondition predicate transformer, with the additional expectation that the precondition ensures avoidance of memory faults. The top element of this predicate transformer lattice is the function $\lambda q.\bot$ (where $\bot$ is the bottom of the predicate lattice) which validates only the triples $\{\mathsf{false}\}c\{q\}$ and the bottom element of the lattice is the function $\lambda q.\mathit{Props}$ which validates all Hoare triples. We think of the top and bottom elements as universal faulting and diverging programs.

This order is an analogue of the ordering on state transformers used in the semantics of Abstract Separation Logic [5]. It is also in the correct direction for the exchange law.

Lemma 4.2. *The exchange law is valid.*

Proof.

$$\begin{aligned}
&\big((F_1 \parallel F_2); (G_1 \parallel G_2)\big)q \\
&\quad = \bigvee\big\{F_1 q_1 * F_2 q_2 \mid q_1 * q_2 \leq (G_1 \parallel G_2)q\big\} \\
&\quad = \bigvee\Big\{F_1 q_1 * F_2 q_2 \mid q_1 * q_2 \leq \bigvee\{G_1 p_1 * G_2 p_2 \mid p_1 * p_2 \leq q\}\Big\} \\
&\quad \geq \bigvee\big\{F_1(G_1 p_1) * F_2(G_2 p_2) \mid p_1 * p_2 \leq q\big\}
\end{aligned}$$

$$= \bigvee\{(F_1; G_1)p_1 * (F_2; G_2)p_2 \mid p_1 * p_2 \leq q\}$$
$$= \big((F_1; G_1) \parallel (F_2; G_2)\big)q$$

We are allowed to make the $\geq$ step because $p_1 * p_2 \leq q \Rightarrow G_1 p_1 * G_2 p_2 \leq \bigvee\{G_1 p_1 * G_2 p_2 \mid p_1 * p_2 \leq q\}$. □

Exchange in the heap model. As an example of the exchange law for heap programs, consider

$$\big([10] := 5 \parallel [20] := 7\big); \big([20] := 3 \parallel [10] := 4\big)$$
$$\sqsubseteq \big([10] := 5; [20] := 3\big) \parallel \big([20] := 7; [10] := 4\big).$$

On the left-hand side of the law we get a program with no races, whereas on the right-hand side we get a faulting program due to the race between addresses 10 and 20. The right-hand side corresponds to the universally faulting element $\lambda q.\bot$ in our semantics. The reason is that, given a postcondition, there is no way to split the precondition in a way that sends address 10 to both processes (each needs 10 to execute safely, and so we end up with false as the precondition).

At this point in our development we do not yet have a concurrent monoid because $\parallel$ and ; do not have the same unit. For an example of why skip is not the unit of $\parallel$, note that (skip $\parallel \lambda q.emp$) is different from $\lambda q.emp$: *e.g.* given an "atomic" $p \in Props$ ($p = a * b$ implies that a or b is emp), we have that (skip $\parallel \lambda q.emp)p = p \neq emp$. A unit of $\parallel$ does exist, but we will have no need for it: we will instead restrict attention to the *local* predicate transformers, those that satisfy the Frame rule, in which case skip will function as a unit for both.

Definition 4.3. A predicate transformer F is local iff $\forall p, r.\ (Fp) * r \leq F(p * r)$. We use $Loc[Props \to Props]$ to denote the set of all local predicate transformers.

As we are about to see, a predicate transformer is local just if $F = F \parallel$ skip. This more abstract characterization of locality is related to the idea that we can derive the Frame rule from an instance of the Concurrency rule

$$\frac{\{p\}c\{q\} \quad \{r\}\mathsf{skip}\{r\}}{\{p * r\}c \parallel \mathsf{skip}\{q * r\}}$$

if we have the identity $c = c \parallel$ skip. Indeed, the predicate transformers $[Props \to Props]$ give us a model of the Concurrency but not the Frame rule, whereas $Loc[Props \to Props]$ gives us both.

Lemma 4.4. *A predicate F is local if and only if $F = F \parallel$ skip.*

Proof. We consider each direction of the implication separately.
$\boxed{\Rightarrow}$ Assume F local; then for all q,

$$(F \parallel \mathsf{skip})q = \bigvee\{Fq_1 * q_2 \mid q_1 * q_2 \leq q\} \leq \bigvee\big\{F(q_1 * q_2) \mid q_1 * q_2 \leq q\big\} = F(q)$$

$\boxed{\Leftarrow}$ Assuming that $F = F \parallel$ skip, then for all p, r we have

$$F(p * r) = (F \parallel \mathsf{skip})(p * r) = \bigvee\{Fq * s \mid q * s \leq p * r\} \geq Fp * r.$$

The last step simply picks $q = p$ and $s = r$ in the preceding join. □

Note that skip is not a unit of $\parallel$ in general: given any transformer F,

$$\forall q.\ Fq = Fq * emp \leq \bigvee\{Fq_1 * q_2 \mid q_1 * q_2 \leq q\} = (F \parallel \mathsf{skip})q.$$

Lemma 4.5. *$Loc[Props \to Props]$ is closed under $\parallel$, ; , and* skip, *$\parallel$ and ; are associative with unit* skip, *$\parallel$ is commutative, and arbitrary joins are inherited from $[Props \to Props]$. It thus forms a complete concurrent monoid.*

Proof. We defer the proof that $Loc[Props \to Props]$ is closed under $\parallel$ slightly.
To see that $F; G$ is local for F and G local, we calculate:

$$\forall p, r.\ (F; G)p * r = F(Gp) * r \leq F(Gp * r) \leq F\big(G(p * r)\big) = (F; G)(p * r)$$

where we first use that F is local and second that G is local and F monotone.
The transformer skip is clearly local.

Please cite this article in press as: P.W. O'Hearn et al., On the relation between Concurrent Separation Logic and Concurrent Kleene Algebra, Journal of Logical and Algebraic Methods in Programming (2014), http://dx.doi.org/10.1016/j.jlamp.2014.08.002

To see that $\|$ is associative (for any transformers, as we have not yet proven that $\|$ preserves locality of transformers, which will then also establish that $\|$ is associative for local transformers), notice first that for any q, q_1, q_2, q_3, $\exists q'.\ q_1 * q' \leq q \wedge q_2 * q_3 \leq q'$ if and only if $q_1 * q_2 * q_3 \leq q$ (easily proved by considering each direction separately and picking $q' = q_2 * q_3$ for the right to left implication). Then

$$\begin{aligned} & F_1 \parallel (F_2 \parallel F_3)q \\ & = \bigvee \Big\{ F_1 q_1 * \bigvee \{ F_2 q_2 * F_3 q_3 \mid q_2 * q_3 \leq q' \} \;\Big|\; q_1 * q' \leq q \Big\} \\ & = \bigvee \{ F_1 q_1 * F_2 q_2 * F_3 q_3 \mid \exists q'.\ q_1 * q' \leq q \wedge q_2 * q_3 \leq q' \} \\ & = \bigvee \{ F_1 q_1 * F_2 q_2 * F_3 q_3 \mid q_1 * q_2 * q_3 \leq q \} \end{aligned}$$

We similarly find that $(F_1 \parallel F_2) \parallel F_3$ is equal to the same quantity by first proving that $\exists q'.\ q' * q_3 \leq q \wedge q_1 * q_2 \leq q'$ if and only if $q_1 * q_2 * q_3 \leq q$.

Sequential composition ; is clearly associative. skip is (still) a unit of ; , and is also a unit of $\|$ within $Loc[Props \to Props]$ by Lemma 4.4. $\|$ is clearly commutative.

Let us now show that $F \parallel G$ is local for F and G local by establishing $(F \parallel G) \parallel \mathsf{skip} = F \parallel G$, which suffices thanks to Lemma 4.4. This equality follows from the fact that $\|$ of transformers is associative, with unit skip for local transformers:

$$(F \parallel G) \parallel \mathsf{skip} = F \parallel (G \parallel \mathsf{skip}) = F \parallel G.$$

Finally, to see that joins are inherited, we simply check that the join of local transformers is itself local:

$$\forall p, r.\ \Big(\bigsqcup_i F_i\Big) p * r = \Big(\bigwedge_i F_i p\Big) * r \leq \bigwedge_i (F_i p * r) \leq \bigwedge_i F_i (p * r) = \Big(\bigsqcup_i F_i\Big)(p * r)$$

where we first use that $*$ preserves join and then that each F_i is local. □

4.2. Assertions, and connecting historic and Dijkstra triples

In the usual predicate transformer semantics of triples the specification $\{p\}c\{q\}$ is read as $p \leq [\![c]\!]q$ where $[\![c]\!]$ is the (backward) predicate transformer associated with a command. In the historic interpretation we can also define a triple using a statement that "materializes" all of the states satisfying a predicate [5].

To define this we recall the standard definition of the adjoint (also called residual) $-\!\!*$ of $*$ in the quantale of propositions.

Definition 4.6. $p -\!\!* q = \bigvee \{ m \mid p * m \leq q \}$.

This is the same as the definition of implication in terms of joins and conjunction in a complete Heyting algebra, but with $*$ in place of conjunction.

Fact 4.7. *$(-) * p$ is left adjoint to $p -\!\!* (-)$: $a * p \leq q$ if and only if $a \leq p -\!\!* q$.*

In addition to this standard fact, there is a handy connection to the idea of weakest precondition, which has its roots in the calculation of preconditions in the early work on Separation Logic [28,46]. Viewed as a predicate transformer, $p -\!\!* (-)$ satisfies our locality condition $(Fa) * b \leq F(a * b)$ as this modus ponens reasoning shows

$$\frac{(p -\!\!* a) * b * p \leq a * b}{(p -\!\!* a) * b \leq p -\!\!* (a * b).}$$

Thus, it gives us a way to map assertions into predicate transformers, which provides our basic tool for connecting historic and Dijkstra triples. One can think of $p -\!\!* (-)$ as a kind of allocator: it materializes p into the postcondition, in a way that is compatible with locality [5]. In fact, although we will not use this fact, it is the largest of the local predicate transformers F satisfying the Dijkstra triple $emp \leq Fp$.

Lemma 4.8 *(Agreement of Dijkstra and historic triples). If F is a local predicate transformer then*

$$p \leq Fq \quad \textit{iff} \quad p -\!\!* (-);\ F \sqsubseteq q -\!\!* (-).$$

Proof. $\boxed{\Rightarrow}$ Suppose $p \leq Fq$. Then we need to show

$$\bigvee\{m \mid q * m \le r\} \le \bigvee\{m \mid p * m \le Fr\}$$

for arbitrary r. Given any $m \mid q * m \le r$, it suffices to show that $p * m \le Fr$.

$$p * m \le (Fq) * m \le F(q * m) \le Fr$$

where the first step uses $p \le Fq$ and monotonicity of $*$, the middle step uses locality, and the last step monotonicity of F with $q * m \le r$.

$\boxed{\Leftarrow}$ This direction does not require locality of F. Supposing the right-hand side and evaluating at q, we know $p \mathrel{-\!\!*} Fq \ge q \mathrel{-\!\!*} q$. $emp \le q \mathrel{-\!\!*} q$ follows at once from the adjoint property, so we obtain $emp \le p \mathrel{-\!\!*} Fq$. The adjoint property implies that $emp \le a \mathrel{-\!\!*} b$ iff $a \le b$, so we obtain $p \le Fq$. □

Remark 4.9. The converse of Lemma 4.8 is also true: if F satisfies that

$$p \le Fq \quad \text{iff} \quad p \mathrel{-\!\!*} (-);\, F \sqsubseteq q \mathrel{-\!\!*} (-)$$

then F is local. Given a and b, set $p = Fa$ and $q = a$. As $Fa \le Fa$, we get $Fa \mathrel{-\!\!*} (-);\, F \sqsubseteq a \mathrel{-\!\!*} (-)$. If we apply both sides to $a * b$, we get $a \mathrel{-\!\!*} (a * b) \le Fa \mathrel{-\!\!*} F(a * b)$ which gives

$$\big(a \mathrel{-\!\!*} (a * b)\big) * Fa \le F(a * b).$$

Now we just observe that $b \le a \mathrel{-\!\!*} (a * b)$ to obtain $b * Fa \le F(a * b)$.

Note that Lemma 4.8 talks about only particular predicate transformers gotten from $p \mathrel{-\!\!*} (-)$ in the pre and post of the historic triple, and not arbitrary predicate transformers. This is linked to the requirement of weak CKAs that $a;(\cdot)$ preserve joins. For, in the predicate transformer model $F;(\cdot)$ does not always preserve joins. It does when F preserves conjunctions (meets), and all transformers of the form $p \mathrel{-\!\!*} (-)$ do, by virtue of $p \mathrel{-\!\!*} (-)$ being a right adjoint.

Lemma 4.10 *(Structure of assertions). The function $p \mapsto p \mathrel{-\!\!*} (-)$ mapping propositions to local predicate transformers*

(1) reverses order;
(2) sends joins $\bigvee$ of propositions to joins $\bigsqcup$ of predicate transformers;
(3) sends emp to skip *and $*$ to $\|$.*

Proof. (1) is a special case of Lemma 4.8, taking F in the lemma to be the identity function.

(2) can be seen by the calculation

$$\Big(\bigvee_{i\in I} p_i\Big) \mathrel{-\!\!*} q = \bigwedge_{i\in I} (p_i \mathrel{-\!\!*} q) = \Big(\bigsqcup_{i\in I} p_i \mathrel{-\!\!*} (-)\Big) q$$

where I is any set, the left equality is a standard identity of quantales, and the right uses the definition of $\sqcup$ of predicate transformers.

For (3),

$$emp \mathrel{-\!\!*} q = \bigvee\{m \mid emp * m \le q\} = \bigvee\{m \mid m \le q\} = q$$

and then

$$\begin{aligned}
(p_1 * p_2) \mathrel{-\!\!*} q &= \bigvee\{m \mid p_1 * p_2 * m \le q\} \\
&= \bigvee\{m_1 * m_2 \mid p_1 * m_1 * p_2 * m_2 \le q\} \\
&= \bigvee\big\{(p_1 \mathrel{-\!\!*} m_1) * (p_2 \mathrel{-\!\!*} m_2) \mid p_1 * (p_1 \mathrel{-\!\!*} m_1) * p_2 * (p_2 \mathrel{-\!\!*} m_2) \le q\big\} \\
&= \bigvee\big\{(p_1 \mathrel{-\!\!*} m_1) * (p_2 \mathrel{-\!\!*} m_2) \mid m_1 * m_2 \le q\big\} \\
&= \big((p_1 \mathrel{-\!\!*} -) \parallel (p_2 \mathrel{-\!\!*} -)\big)(q)
\end{aligned}$$

The only point that deserves attention is the $\le$ direction of the third equality above. Suppose that $p_1 * m_1 * p_2 * m_2 \le q$ and let us write $r_i = p_i * m_i$ $(i \in \{1, 2\})$. We first remark that $m_i \le p_i \mathrel{-\!\!*} r_i$, hence $m_1 * m_2 \le (p_1 \mathrel{-\!\!*} r_1) * (p_2 \mathrel{-\!\!*} r_2)$. Moreover,

$$p_1 * (p_1 \mathrel{-\!\!*} r_1) * p_2 * (p_2 \mathrel{-\!\!*} r_2) \le r_1 * r_2 = p_1 * m_1 * p_2 * m_2 \le q$$

For the $\le$ direction of the fourth equality, we observe that $p * p \mathrel{-\!\!*} m = m$ for all p and m. □

Please cite this article in press as: P.W. O'Hearn et al., On the relation between Concurrent Separation Logic and Concurrent Kleene Algebra, Journal of Logical and Algebraic Methods in Programming (2014), http://dx.doi.org/10.1016/j.jlamp.2014.08.002

Remark on Yoneda and Day. This mapping $p \mapsto p \multimap *(-)$ is reminiscent of the Yoneda embedding $d \mapsto Hom_C[-, d]$ mapping a category C into $Set^{C^{op}}$, and the preservation properties mapping monoidal to monoidal structure, and joins to joins is reminiscent of classic results of Day on closed categories [9,10] and the Yoneda embedding. The fact that here $p \mapsto p \multimap *(-)$ preserves joins and not meets, where for Yoneda it is the other way around, is simply a matter of reversing the propositions and transformers orderings (like, starting from C^{op} rather than C). Staton has even suggested to us that this similarity might be made a precise statement by working with quantale-enriched categories.

Technical fine points concerning meets (continued). This is related to the paragraph on technical fine points concerning meets in Section 3.

Since $p \mapsto p \multimap *(-)$ reverses order, it maps meets to meets within its image, *i.e.* $(\bigwedge_i p_i) \multimap *(-)$ is the greatest predicate transformer smaller than all $p_i \multimap *(-)$ *of the form* $p \multimap *(-)$. It may not, however be the same as $\bigsqcap_i (p_i \multimap *(-))$, *i.e.* the set of predicate transformers of the form $p \multimap *(-)$ may not be closed under meet of predicate transformers.

For an example of this situation, consider the quantale $(\mathcal{P}(\mathbb{N}), \subseteq, +, \emptyset)$, *i.e.* subsets of natural numbers with $*$ being $+$ lifted to sets. Then $(\{1\} \wedge \{2\}) \multimap *\emptyset = \emptyset \multimap *\emptyset = \mathbb{N}$ while $(\{1\} \multimap *\emptyset) \vee (\{2\} \multimap *\emptyset) = \emptyset \vee \emptyset = \emptyset$.

Note that $(\{1\} \multimap *(-)) \wedge (\{2\} \multimap *(-))$ cannot be expressed as $q \multimap *(-)$ for any q, because if it could then, by reflexion of order, q would have to be $\{1\} \wedge \{2\}$ and we just saw that choice lead to a different transformer.

Proposition 4.11. *Setting $A \subseteq Loc[Props \to Props]$ to be the set of predicate transformers of the form $p \multimap *(-)$ gives us a weak CKA structure.*

Proof. Lemma 4.10 gives us that assertions form a quantale, and Lemma 4.5 gives us that local transformers form a concurrent monoid. All that is left is to show that the function $F \mapsto p \multimap *(-); F$, taking a local predicate transformer as an argument and then sequentially composing on the right, and the function $A \mapsto A; F$, taking an assertion and then sequentially composing on the left, preserve joins of predicate transformers.

For $F \mapsto p \multimap *(-); F$, and given a predicate p, we have seen that since $p \multimap *(-)$ is a right adjoint it preserves meets of propositions. An immediate consequence from the definition of joins of predicate transformers in the inverse point-wise order is that $F \mapsto p \multimap *(-); F$ preserves joins of predicate transformers: for any I and q, we have that

$$\begin{aligned}
\Big(p \multimap *(-); \bigsqcup_{i\in I} F_i\Big)q &= p \multimap * \bigwedge_{i\in I} F_i q && \text{(def. of ; and } \sqcup\text{)}\\
&= \bigwedge_{i\in I} p \multimap * F_i q && (\wedge\text{-preservation})\\
&= \Big(\bigsqcup_{i\in I} p \multimap *(-); F_i\Big)q && \text{(def. of ; and } \sqcup\text{)}
\end{aligned}$$

For $A \mapsto A; F$, we require to show that $(\bigsqcup_{i\in I} A_i); F = \bigsqcup_{i\in I}(A_i; F)$ for any collection of assertions $\{A_i\}_{i\in I}$. Each assertion A_i being of the form $p_i \multimap *(-)$, we get the desired equality from the following derivation:

$$\begin{aligned}
\Big(\bigsqcup_{i\in I} p_i \multimap *(-)\Big); F &= \Big(\Big(\bigvee_{i\in I} p_i\Big) \multimap *(-)\Big); F && \text{(Lemma 4.10-(2))}\\
&= \Big(\bigvee_{i\in I} p_i\Big) \multimap * F(-) && \text{(def. of ;)}\\
&= \bigsqcup_{i\in I} p_i \multimap * F(-) = \bigsqcup_{i\in I}\big(p_i \multimap *(-); F\big) && \text{(Lemma 4.10-(2) and def. of ;)}
\end{aligned}$$

□

4.3. Completeness

We now prove the main technical results of the paper, which show that ASL^{--} can in a sense be embedded in a (weak) CKA. To begin, we need a result which expresses the least-fixed point semantics of while loops as a predicate transformer defined in accord with the proof theory.

Lemma 4.12 *(Logical characterization of iteration). Recall that iteration is defined in accord with Tarski's least fixed-point theorem:*

$$\mathsf{iterate}(F) = \mu i_F = \bigsqcap\{F' \mid (\mathsf{skip} + F; F') \sqsubseteq F'\}.$$

We can define a predicate transformer for iteration following the proof rules:

$$\mathsf{iterate}_{logical}(F) = \lambda q. \bigvee\{p \mid p \le Fp \text{ and } p \le q\}.$$

Then $\mathsf{iterate} = \mathsf{iterate}_{logical}$.

Please cite this article in press as: P.W. O'Hearn et al., On the relation between Concurrent Separation Logic and Concurrent Kleene Algebra, Journal of Logical and Algebraic Methods in Programming (2014), http://dx.doi.org/10.1016/j.jlamp.2014.08.002

Proof. We show each direction separately.

(iterate$(F) \sqsubseteq$ iterate$_{logical}(F)$): The least fixed-point μf of a monotone function is also the least pre-fixed-point and so has the induction rule

$$f z \sqsubseteq z \;\Rightarrow\; \mu f \sqsubseteq z.$$

Taking $z =$ iterate$_{logical}(F)$ and $f = i_F = \lambda F'.\ \mathsf{skip} + F; F'$, this yields

$$\mathsf{skip} + F; \mathsf{iterate}_{logical}(F) \sqsubseteq \mathsf{iterate}_{logical}(F) \;\Rightarrow\; \mathsf{iterate}(F) \sqsubseteq \mathsf{iterate}_{logical}(F).$$

So, to get what we want to show we are left with proving

$$\mathsf{skip} + F; \mathsf{iterate}_{logical}(F) \sqsubseteq \mathsf{iterate}_{logical}(F).$$

It is evident that skip $\sqsubseteq$ iterate$_{logical}(F)$, so we need $F;$ iterate$_{logical}(F) \sqsubseteq$ iterate$_{logical}(F)$, which is to say

$$F\Big(\bigvee\{p \mid p \le Fp \wedge p \le q\}\Big) \ge \bigvee\{p \mid p \le Fp \wedge p \le q\} \tag{i}$$

for arbitrary q. If $p' \le Fp'$ and $p' \le q$ then $Fp' \le F(\bigvee\{p \mid p \le Fp \wedge p \le q\})$ by monotonicity of F, since $p' \le \bigvee\{p \mid p \le Fp \wedge p \le q\}$. We then have $p' \le F(\bigvee\{p \mid p \le Fp \wedge p \le q\})$ because $p' \le Fp'$.

Then since all such p' are $\le F(\bigvee\{p \mid p \le Fp \wedge p \le q\})$ so is their join $\bigvee\{p \mid p \le Fp \wedge p \le q\}$, which establishes (i) as required.

(iterate$_{logical}(F) \sqsubseteq$ iterate(F)): This is equivalent to showing

$$\bigvee\{p \mid p \le Fp \text{ and } p \le q\} \ge \bigvee\big\{F'q \;\big|\; \big(\mathsf{skip} + F; F'\big) \sqsubseteq F'\big\}.$$

It is sufficient to show

$$\big(\mathsf{skip} + F; F'\big) \sqsubseteq F' \;\Rightarrow\; \bigvee\{p \mid p \le Fp \wedge p \le q\} \ge F'q$$

because if the lhs is $\ge$ each $F'q$ then it is $\ge$ their join. Given an F' making the antecedent true, it is then enough to show that (choosing p as $F'q$ inside the $\bigvee$)

$$F'q \le F\big(F'q\big) \wedge F'q \le q. \tag{ii}$$

From $(\mathsf{skip} + F; F') \sqsubseteq F'$, we obtain that $(q \wedge F(F'q)) \ge F'q$, and this implies (ii) at once, and we are done. □

Our completeness proof is phrased in terms of the Dijkstra triple, which we will then connect to the main theorem.

Proposition 4.13 *(Provability of wlp). Suppose that commands c are built up from primitive commands c_{prim} and $+$, ;, $\|$, and* iterate*. For primitive commands c_{prim} we define the predicate transformer $[\![c_{prim}]\!]$ by*

$$[\![c_{prim}]\!]q = \bigvee\big\{p \mid \mathit{Axioms} \vdash \{p\}c_{prim}\{q\}\big\}.$$

Then the completeness property is

$$\mathit{Axioms} \vdash \big\{[\![c]\!]q\big\}c\{q\}.$$

Proof. The proof is by induction on the structure of c. The case of primitive commands is immediate from the Disjunction rule. Note that $[\![c_{prim}]\!]$ is a monotone and local predicate transformer because of the Consequence and the Frame rules.

Case $\|$. We want $\mathit{Axioms} \vdash \{[\![c_1 \,\|\, c_2]\!]q\}c_1 \,\|\, c_2\{q\}$ where

$$[\![c_1 \,\|\, c_2]\!]q = \bigvee\big\{[\![c_1]\!]q_1 * [\![c_2]\!]q_2 \mid q_1 * q_2 \le q\big\}$$

This is immediate from the induction hypothesis with the proof rules for parallel, consequence and disjunction. (We use the possibly infinitary disjunctive rule of ASL^{--}.)

Case $+$. By induction hypothesis we know that $\mathit{Axioms} \vdash \{[\![c_i]\!]q\}c_i\{q\}$, and since $[\![c_1]\!]q \wedge [\![c_2]\!]q \le [\![c_i]\!]q$ and by the rule of consequence we have $\mathit{Axioms} \vdash \{[\![c_1]\!]q \wedge [\![c_2]\!]q\}c_i\{q\}$. By the rule for $+$ and the definition of $[\![c_1 + c_2]\!]$ we obtain $\mathit{Axioms} \vdash \{[\![c_1 + c_2]\!]q\}c_1 + c_2\{q\}$.

Case ; is a similarly immediate application of the induction hypothesis and proof rules. *Case* skip is immediate.

Please cite this article in press as: P.W. O'Hearn et al., On the relation between Concurrent Separation Logic and Concurrent Kleene Algebra, Journal of Logical and Algebraic Methods in Programming (2014), http://dx.doi.org/10.1016/j.jlamp.2014.08.002

Case iterate. We use the alternate definition of iterate, justified by Lemma 4.12:

$$[\![\mathsf{iterate}(c)]\!]\, q = \bigvee \{p \mid p \leq [\![c]\!]p \text{ and } p \leq q\}.$$

Let $I = [\![\mathsf{iterate}(c)]\!]q$. If we can show

$$\text{(i)} \quad I \leq q, \quad \text{and} \quad \text{(ii)} \quad \{I\}c\{I\}$$

then we get the desired result $\{[\![\mathsf{iterate}(c)]\!]q\}\mathsf{iterate}(c)\{q\}$ from the rule of consequence on the right and the proof rule for iteration. (i) is immediate from the definition.

For (ii), it suffices to show

$$\bigvee \{p \mid p \leq [\![c]\!]p \wedge p \leq q\} \leq [\![c]\!]\Big(\bigvee \{p \mid p \leq [\![c]\!]p \wedge p \leq q\}\Big).$$

If $p \leq [\![c]\!]p \wedge p \leq q$ then $p \leq [\![c]\!](\bigvee\{p \mid p \leq [\![c]\!]p \wedge p \leq q\})$ by monotonicity of $[\![c]\!]$, since $p \leq \bigvee\{p \mid p \leq [\![c]\!]p \wedge p \leq q\}$. Then since all the p's are $\leq$ so is their join $\bigvee\{p \mid p \leq [\![c]\!]p \wedge p \leq q\}$. □

Setting the meaning $[\![p]\!]$ of an assertion as $p \mathrel{-\!\!*} (-)$, we can now establish Theorem 4.1, where we require

$$\mathit{Axioms} \vdash \{p\}c\{q\} \quad \text{iff} \quad [\![p]\!]; [\![c]\!] \sqsubseteq [\![q]\!].$$

Proof of Theorem 4.1. For the if direction if we have $[\![p]\!]; [\![c]\!] \sqsubseteq [\![q]\!]$ then $p \leq [\![c]\!]q$ by Lemma 4.8, $\mathit{Axioms} \vdash \{[\![c]\!]q\}c\{q\}$ from Proposition 4.13, and then $\mathit{Axioms} \vdash \{p\}c\{q\}$ using the rule of consequence on the left. For the only if direction, all of the proof rules are sound by Theorem 3.3 and the uses of Disjunction and the Consequence rule in the proof theory are validated by the interpretation of assertions in the model by Lemma 4.10. □

5. A standard CSL traces model as a CKA

Although we have seen that a version of CSL can be embedded in a version of CKA, we have not shown that CKA generalizes existing models of CSL. There are a number of models including both denotational [4,5,21] and operational [43, 11] ones, and as far as we know it has not been shown that any are instances of CKA.

In this section we look at a very basic, almost trivial model, and attempt to connect it to CKA. The model is based on the fact that the idea of interleaving is in a sense derivable in CSL. For, the following is a derivable rule.

$$\frac{\{p_1\}c_1\{q_1\} \quad \{p_2\}c_2\{q_2\}}{\{p_1 * p_2\}(c_1; c_2) + (c_2; c_1)\{q_1 * q_2\}}$$

It can be derived using the proof rules for non-determinism and sequencing, and the Frame rule. Indeed we first have

$$\frac{\dfrac{\{p_1\}c_1\{q_1\}}{\{p_1 * p_2\}c_1\{q_1 * p_2\}} \quad \dfrac{\{p_2\}c_2\{q_2\}}{\{q_1 * p_2\}c_2\{q_1 * q_2\}}}{\{p_1 * p_2\}c_1; c_2\{q_1 * q_2\}}$$

by framing on p_2 and q_1 in the top two steps, and a similar deduction from the same premises gives us $\{p_1 * p_2\}c_2; c_1\{q_1 * q_2\}$; then we can apply the $+$ rule.

The derivability of this interleaving rule for two actions suggests a simple model: the elements of the model are traces of actions, and the assertions are sets of states. Then, a triple is true of a set of traces just if each linear trace validates the pre/post spec under the usual sequential semantics of traces as sequential compositions.

The interleaving models of CSL which have appeared in the literature are based on this idea in initial conception, but become more complicated because they seek to account for critical regions or other synchronization primitives. We avoid these in order to study the link to CKA in the simplest setting possible.

5.1. Standard model

We first give a model of an instantiation of ASL^{--}, where the model is expressed without any reference to the exchange rule or historic triple. We call this the standard model.

As propositions we take the quantale

$$\big(\mathcal{P}(\mathit{Heaps}), \subseteq, \circledast, \{u\}\big)$$

from Section 2, where $\mathit{Heaps} = \mathbb{N} \rightharpoonup_f \mathbb{N}$ and the definitions of state disjointness $\sharp$, disjoint composition $\bullet$, and separating conjunction $\circledast$ are as there. In this section we use $\circledast$ instead of $*$ to refer to the separating conjunction in this model because we are going to define two models of ASL^{--}, and we want to keep the notation separate to avoid confusion when comparing them. (So neither will use the symbol $*$.)

Please cite this article in press as: P.W. O'Hearn et al., On the relation between Concurrent Separation Logic and Concurrent Kleene Algebra, Journal of Logical and Algebraic Methods in Programming (2014), http://dx.doi.org/10.1016/j.jlamp.2014.08.002

The elements of our traces will be state transformations representing actions. Formally, these are functions $f : Heaps \to \mathcal{P}(Heaps)^\top$, where $\mathcal{P}(Heaps)^\top = \mathcal{P}(Heaps) \uplus \{\top\}$ is the "topped powerset" of *Heaps*, whose order $\sqsubseteq$ extends set inclusion $\subseteq$ with the greatest element $\top$. These functions satisfy a locality condition which makes the Frame rule true [5]:

locality: $\sigma_1 \natural \sigma_2$ implies $f(\sigma_1 \bullet \sigma_2) \sqsubseteq (f\sigma_1) \circledast \{\sigma_2\}$.

This condition ensures that, essentially, the Frame rule is true in the individual steps in an action trace (a trace of such functions), which provides the ingredient to semantically carry out and extend the reasoning used in the derivation of interleaving of two actions above. In the definition, $\circledast$ extends from $\mathcal{P}(Heaps)$ to $\mathcal{P}(Heaps)^\top$ by setting $p \circledast \top = \top \circledast p = \top$. We write *LocalFun* for the set of local functions.

The sequential composition $f; g$ of local functions functionally composes f with the obvious lifting $g^\dagger : \mathcal{P}(Heaps)^\top \to \mathcal{P}(Heaps)^\top$:

$$g^\dagger(S) = \bigsqcup_{s \in S} g(s), \qquad g^\dagger(\top) = \top$$

Definition 5.1. A trace is a finite or infinite sequence of local functions. We let *Traces* denote the set of all such traces.

For our instantiation of ASL^{--} we take $\|$ to be (lifted) interleaving of traces, ; to be (lifted) concatenation, skip to be the singleton set containing only the empty trace, $\sqcup$ to be union of trace sets, and iteration is calculated as a least fixed-point in the usual way.

A Hoare triple in this instantiation of ASL^{--} is a formula $\{p\}T\{q\}$ where $p, q \in \mathcal{P}(Heaps)$ are sets of heaps and T is a set of traces of local actions. We want to say that a triple $\{p\}T\{q\}$ is true just when $\{p\}t\{q\}$ holds in sequential separation logic for each of the traces $t \in T$. Let us define this.

Definition 5.2 *(Abstraction of a trace).*

- If t is a finite trace $f_1 \dots f_n$ then $\mathsf{abstract}(t)$ is its sequential composition (*i.e.*, a local action) $f_1; \dots; f_n$, with $\lambda\sigma.\{\sigma\}$ being the abstraction of the empty sequence.
- If t is infinite, then $\mathsf{abstract}(t)\sigma$ is $\emptyset$ if $\mathsf{abstract}(t')\sigma \neq \top$ for each finite prefix t' of t. Otherwise $\mathsf{abstract}(t)\sigma$ is $\top$.

Definition 5.3 *(Triples for traces).* We say that the triple $\{p\}t\{q\}$ is true of an individual trace t iff for all $\sigma \in p$. $\mathsf{abstract}(t)\sigma \sqsubseteq q$.

This is the semantic interpretation of triples from [5], adjusted so that in the case of an infinite trace we are just checking that none of the finite prefixes delivers a memory fault.

Definition 5.4 *(Triples for trace sets).* We say that the triple $\{p\}T\{q\}$ is *true in the standard model* just if $\{p\}t\{q\}$ is true for each $t \in T$.

Equivalently, we can write $\mathsf{abstract}(T)$ for $\lambda\sigma.\bigsqcup_{t \in T} \mathsf{abstract}(t)\sigma$ (with the convention that for $T = \emptyset$, $\mathsf{abstract}(T) = \lambda\sigma.\emptyset$) and then $\{p\}T\{q\}$ is true if and only if $\mathsf{abstract}(T)\sigma \subseteq q$ for all $\sigma \in p$.

We also assume a collection *Axioms* of axioms about the primitive commands of the form $\{p\}c\{q\}$ such that there is at least one axiom for each $c \in Com$.

Let us recall the definition of best local actions adapted from Abstract Separation Logic [5].

Definition 5.5 *(Best local action).* The *best local action from* p_1 *to* p_2, written $bla[p_1, p_2]$, is the function of type $Heaps \to \mathcal{P}(Heaps)^\top$ defined by

$$bla[p_1, p_2](\sigma) = \bigsqcap \{ p_2 \circledast \{\sigma_0\} \mid \sigma = \sigma_0 \bullet \sigma_1, \sigma_1 \in p_1 \}.$$

It is shown in [5] that every best local action $bla[p, q]$ is indeed local and is the greatest local function (for the point-wise inclusion order) satisfying the Hoare triple $\{p\} - \{q\}$.

The proof of the soundness of Abstract Separation Logic can be adapted to prove the soundness of ASL^{--} w.r.t. this trace model. We first prove an auxiliary lemma, which will help us assert the soundness of the Concurrency proof rule, before presenting the general soundness theorem for ASL^{--}.

Lemma 5.6. *If* $\sigma = \sigma_1 \bullet \sigma_2$, $\mathsf{abstract}(t_1)\sigma_1 \subseteq q_1$, $\mathsf{abstract}(t_2)\sigma_2 \subseteq q_2$, *and* $t \in t_1 \parallel t_2$, *then* $\mathsf{abstract}(t)\sigma \subseteq q_1 \circledast q_2$.

Please cite this article in press as: P.W. O'Hearn et al., On the relation between Concurrent Separation Logic and Concurrent Kleene Algebra, Journal of Logical and Algebraic Methods in Programming (2014), http://dx.doi.org/10.1016/j.jlamp.2014.08.002

Proof. Let us first prove the result for all pairs of *finite* traces t_1 and t_2, by induction on the sums of their lengths. The interesting case is the interleaving of two non-empty traces, *i.e.* when $t_i = f_i; t_i'$, $i \in \{1,2\}$ for some local functions f_1 and f_2 and traces t_1' and t_2'. Recall the definition of parallel composition of linear traces from Section 3, and notice that in this case, either $t = f_1; t'$ for some $t' \in t_1' \parallel (f_2; t_2')$ or $t = f_2; t'$ for some $t' \in (f_1; t_1') \parallel t_2'$. Let us assume, without loss of generality, that we are in the first case.

Let $\sigma = \sigma_1 \bullet \sigma_2$ and suppose that $\mathsf{abstract}(t_i)\sigma_i \subseteq q_i$ for $i \in \{1,2\}$. Let moreover $\sigma_1' \in f_1(\sigma_1)$. Since $\mathsf{abstract}(f_1; t_1')\sigma_1 \subseteq q_1$ and $q_1 \neq \top$, by the semantics of sequential composition we know that $\mathsf{abstract}(t_1')\sigma_1' \subseteq q_1$. Moreover, by induction hypothesis $\mathsf{abstract}(t')(\sigma_1' \bullet \sigma_2) \subseteq q_1 \circledast q_2$. Thus, $\mathsf{abstract}(t')\sigma' \subseteq q_1 \circledast q_2$ for all $\sigma' \in f_1(\sigma_1) \circledast \{\sigma_2\}$. Since f_1 is local, we have that

$$\mathsf{abstract}(t)\sigma = \mathsf{abstract}(t')^{\dagger}\big(f_1(\sigma_1 \bullet \sigma_2)\big) \subseteq \mathsf{abstract}(t')^{\dagger}\big((f_1\sigma_1) \circledast \{\sigma_2\}\big) \subseteq q_1 \circledast q_2$$

which concludes the proof for finite traces.

Assume now that at least one of t_1, t_2 is infinite. Then any trace $t \in t_1 \parallel t_2$ is infinite as well. Let t' be a finite prefix of such a trace t. There are (finite) prefixes t_1' and t_2' of t_1 and t_2 respectively such that $t' \in t_1' \parallel t_2'$. By our hypothesis that $\mathsf{abstract}(t_i)\sigma_i \subseteq q_i$, we deduce that $\mathsf{abstract}(t_i')\sigma_i \subseteq q_i'$ for some $q_i' \neq \top$ (taking $q_i' = \mathsf{abstract}(t_i')\sigma_i$ if t_i is infinite and $q_i' = q_i$ otherwise). Since t_1' and t_2' are finite, we have shown above that $\mathsf{abstract}(t')\sigma \subseteq q_1' \circledast q_2'$. Thus, $\mathsf{abstract}(t')\sigma \neq \top$ for all finite prefixes t' of t so $\mathsf{abstract}(t)\sigma = \emptyset \subseteq q_1 \circledast q_2$ as required. □

Theorem 5.7. *The proof rules of ASL^{--} preserve truth in the standard model.*

Proof. We proceed by case analysis. Frame follows directly from the locality of individual actions. All the other rules are straightforward except for Concurrency. Assume that $\{p_1\}T_1\{q_1\}$ and $\{p_2\}T_2\{q_2\}$ are true Hoare triples. We need to show that $\{p_1 \circledast p_2\}T_1 \parallel T_2\{q_1 \circledast q_2\}$ is true, *i.e.*, for all $t \in T_1 \parallel T_2$ and $\sigma \in p_1 \circledast p_2$, $\mathsf{abstract}(t)\sigma \subseteq q_2 \circledast q_2$. Let $t \in T_1 \parallel T_2$ and $\sigma \in p_1 \circledast p_2$. There are $t_1 \in T_1$ and $t_2 \in T_2$ such that $t \in t_1 \parallel t_2$, and there are $\sigma_1 \in p_1$ and $\sigma_2 \in p_2$ such that $\sigma = \sigma_1 \bullet \sigma_2$. By assumption, $\mathsf{abstract}(t_i) \subseteq q_i$ for $i \in \{1,2\}$, hence by Lemma 5.6 $\mathsf{abstract}(t)\sigma \subseteq q_1 \circledast q_2$ as required. □

5.2. CKA model

The local action traces induce a different semantics of ASL^{--}, based on the fact that the traces form a Boolean CKA (and hence a weak CKA). The CKA model gives us a model of the proof rules for triples of the form $A; c \sqsubseteq B$. Our purpose in this section is to study whether this structure can be used to tell us anything about the theory of the triples $\{p\}c\{q\}$ in the standard model.

The main step in connecting the two models is to relate state assertions and trace assertions, in order to compare the standard, state-based triples with historic triples. The natural way to do this, as discussed in connection with the linear trace model in Section 3, is to map a state assertion p to traces $\langle\!\langle p \rangle\!\rangle$ that end in states where p holds. Transporting this idea to the "traces of local functions" model we look at those traces t for which we can establish $\{emp\}t\{p\}$.

Definition 5.8 *(Assertions).* We define the set $\mathcal{A}$ of "assertions" to be the set of those trace sets whose elements can safely execute from the empty heap, *i.e.*

$$\mathcal{A} = \big\{T \mid \{emp\}T\{true\} \text{ is true}\big\}.$$

Definition 5.9. We interpret a proposition $p \in \mathcal{P}(Heaps)$ as the following trace-set $\langle\!\langle p \rangle\!\rangle \in \mathcal{A}$:

$$\langle\!\langle p \rangle\!\rangle = \big\{t \mid \{emp\}t\{p\} \text{ is true}\big\}.$$

Proposition 5.10. *Writing* skip *for* $\{\epsilon\}$, $(\mathcal{P}(Tr(LocalFun)), \subseteq, \|, ;, \mathsf{skip})$ *is a Boolean CKA and* $(\mathcal{P}(Tr(LocalFun)), \mathcal{A}, \subseteq, \|, ;, \mathsf{skip})$ *is a weak CKA.*

Proof. Straightforward. □

Definition 5.11 *(Historic triples).* We say that the triple $\{p\}T\{q\}$ is *true in the CKA* if $\langle\!\langle p \rangle\!\rangle; T \subseteq \langle\!\langle q \rangle\!\rangle$.

Thus, using the historic interpretation of triples in conjunction with the mapping $p \mapsto \langle\!\langle p \rangle\!\rangle$ gives us a way to compare triples in the standard model with those in the CKA.

First, we have a positive result.

Proposition 5.12 *(Agreement of triples). Historic triples in the CKA agree with triples in the standard model. For all p, q and T,*

$$\{p\}T\{q\} \textit{ is true in the CKA} \quad \Leftrightarrow \quad \{p\}T\{q\} \textit{ is true in the standard model.}$$

Please cite this article in press as: P.W. O'Hearn et al., On the relation between Concurrent Separation Logic and Concurrent Kleene Algebra, Journal of Logical and Algebraic Methods in Programming (2014), http://dx.doi.org/10.1016/j.jlamp.2014.08.002

Proof. $\boxed{\Rightarrow}$ Suppose that $\{p\}T\{q\}$ is true in the CKA, and let $\sigma \in p$. The definition of CKA triples and the fact that $\{emp\}bla[emp, p]\{p\}$ entail $bla[emp, p]; T \subseteq \langle\langle q \rangle\rangle$, hence $\{emp\}bla[emp, p]; T\{q\}$. Thus, for all finite $t \in T$, $\mathsf{abstract}(bla[emp, p]; t)u \subseteq q$. Moreover, it is easy to check that $bla[emp, p](u) = p$, hence by definition of sequential composition and since $\sigma \in p = bla[emp, p](u)$, $\mathsf{abstract}(t)\sigma \subseteq q$. A similar reasoning gives $\mathsf{abstract}(t')\sigma = \emptyset \subseteq q$ for all infinite traces t'.
$\boxed{\Leftarrow}$ Assume that $\{p\}T\{q\}$ is true in the standard model, and let $t \in \langle\langle p \rangle\rangle; T$. We require to prove that $t \in \langle\langle q \rangle\rangle$. By definition, either $t \in \langle\langle p \rangle\rangle$ (t is infinite) and we are done, or there are $t_p \in \langle\langle p \rangle\rangle$ and $t' \in T$ such that t_p is finite and $t = t_p t'$. Then, $\mathsf{abstract}(t)u = \mathsf{abstract}(t')^\dagger(\mathsf{abstract}(t_p)u) \sqsubseteq \mathsf{abstract}(t')^\dagger p$ hence $\mathsf{abstract}(t)u \sqsubseteq q$ by hypothesis, which shows that $t \in \langle\langle q \rangle\rangle$. □

Thus, we have been able to use CKA to recover the same meaning of Hoare triples as in the previous section. The soundness theorem for weak CKAs, Theorem 3.3, applies. It implies that the Frame and Concurrency rules are sound, if we interpret $*$ as $\|$. However, in the standard model we are interpreting $*$ as $\circledast$. The crucial question is whether the CKA structure can be used to derive the soundness of the Frame and Concurrency rules expressed w.r.t. $\circledast$.

Note that the soundness of these rules is not in question, as we established them with a direct proof above. The question, rather, is whether the soundness of the logic in the standard model, using $\circledast$, is a *consequence* of results concerning CKAs.

Now we run into a problem. Trying, for instance, to establish the Frame rule, we assume $\{p\}T\{q\}$ and try to prove $\{p \circledast r\}T\{q \circledast r\}$ using the CKA structure. We get as far as

$$\{p\}T\{q\} \quad \Leftrightarrow \quad \langle\langle p \rangle\rangle; T \subseteq \langle\langle q \rangle\rangle \quad \Rightarrow \quad \big(\langle\langle p \rangle\rangle \parallel \langle\langle r \rangle\rangle\big); T \subseteq \langle\langle q \rangle\rangle \parallel \langle\langle r \rangle\rangle.$$

which is the Frame rule for $\|$. We might hope to use $\langle\langle p \circledast r \rangle\rangle; T \subseteq \langle\langle q \circledast r \rangle\rangle \Leftrightarrow \{p \circledast r\}T\{q \circledast r\}$, but we are blocked by a mismatch between $\langle\langle p \circledast q \rangle\rangle$ and $\langle\langle p \rangle\rangle \parallel \langle\langle q \rangle\rangle$. In fact, we have that $\langle\langle p \rangle\rangle \parallel \langle\langle q \rangle\rangle \subseteq \langle\langle p \circledast q \rangle\rangle$ but the other direction does not hold in general and this prevents us from establishing the desired result $\{p \circledast r\}T\{q \circledast r\}$.

To see why we have this problem, consider the following local function below, taken from [5] and analogous to the assembly language load instruction:

$$load(l, x) = \prod_v bla[l \mapsto v \circledast x \mapsto -, l \mapsto v \circledast x \mapsto v].$$

Then the two-action trace $t = bla[emp, l \mapsto - \circledast x \mapsto -]load(l, x)$ appears in $\langle\langle l \mapsto v \circledast x \mapsto v \rangle\rangle$ but not in $\langle\langle l \mapsto v \rangle\rangle \parallel \langle\langle x \mapsto v \rangle\rangle$. Indeed, the latter is comprised of interleavings of traces ending in states satisfying $l \mapsto v$ and $x \mapsto v$ respectively, neither of which contains t.

The problem here is that $\langle\langle l \mapsto v \rangle\rangle \parallel \langle\langle x \mapsto v \rangle\rangle$ includes actions that generate $l \mapsto v$ and $x \mapsto v$ one at a time, but not the trace t which allocates l and x in one step. This is the essential difference, in this model, between $\|$ and $\circledast$.

Our main conclusion from this exercise is the following.

The CKA structure does not give us a way of proving that the standard model is in fact a model of ASL^{--}.

This conclusion should be taken with a grain of salt, as there might be *another* way to see the standard model as a CKA. But we are not aware of another such way. At any rate, to us there is no obvious way to see this standard, basic model of concurrent separation logic as an instance of CKA. If CKA were to strictly generalize concurrent separation logic, we would expect to find the most basic models as direct instances.

5.3. Postscript: a curiosity

The mismatch in the proof theories of the standard model and the CKA outlined above hampers using the CKA structure for reasoning about programs when specifications that use $\circledast$ are involved (as is the case for $load(l, x)$ above), as it clashes with the use of $\|$ in its Frame rule.

Despite this, we remark that this CKA model of ASL^{--} is still (curiously) interesting. To see why, consider the historic triple

$$\langle\langle l \mapsto v \rangle\rangle \parallel \langle\langle x \mapsto - \rangle\rangle; command \subseteq \langle\langle l \mapsto v \rangle\rangle \parallel \langle\langle x \mapsto v \rangle\rangle$$

where we are leaving *command* unspecified (or, we could take it to be the greatest local predicate transformer satisfying the triple). The precondition $\langle\langle l \mapsto v \rangle\rangle \parallel \langle\langle x \mapsto - \rangle\rangle$ is indeed satisfied by traces where l and x are separate locations at the end, and similarly for the postcondition $\langle\langle l \mapsto v \rangle\rangle \parallel \langle\langle x \mapsto v \rangle\rangle$. So the *command* will consist of traces that happen to end up with x storing v. There will be strange traces, such as one that first havocs l, $l \mapsto w$, then writes x, $x \mapsto v$, and then restores l, $l \mapsto v$, but not one that takes a state containing both $l \mapsto v$ and $x \mapsto -$ and then updates it in place. The connection of such commands to operational intuition concerning the load command is perhaps difficult to see, and we would need a further semantic analysis to connect the meaning of such triples to a standard semantics of in-place update. But, *if* we accept such specifications for primitive commands as the one for load, *then* we can recast all the proofs done in the standard model, by replacing $\circledast$ by $\|$. The proofs and specifications will just mean something different than in the standard model.

So, the discussion in this section should not be taken to mean that the CKA structure in the particular model is somehow "wrong". It is simply not the standard model. And yet it has reasonable proof properties.

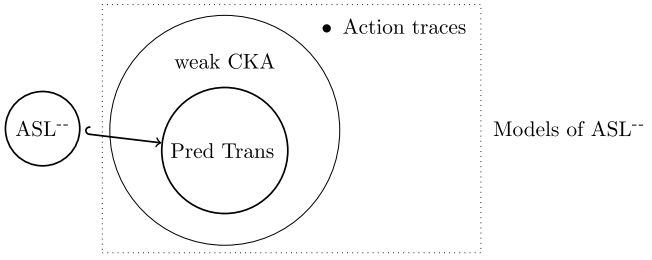
Please cite this article in press as: P.W. O'Hearn et al., On the relation between Concurrent Separation Logic and Concurrent Kleene Algebra, Journal of Logical and Algebraic Methods in Programming (2014), http://dx.doi.org/10.1016/j.jlamp.2014.08.002

At present, it seems that using CKAs to achieve CSL-like reasoning *requires* such non-standard models. One reaction to this situation might be that one can or should accept CKAs the way they are and learn to live with the non-standard models (this would require understanding the non-standard models better), while another reaction is to try to change CKAs so as to cover more models.

6. Conclusion

Our main technical results may be summarized as follows, and are depicted in diagrammatic form below.

- We have found a class of algebras, weak CKAs, that are sound and complete for a general form of Concurrent Separation Logic, ASL^{--}.
- At the same time we have found natural models of separation logic reasoning, based on action traces, that do not fit the CKA theory as it stands.



The situation is roughly analogous to what one finds in λ-calculus, if we substitute weak CKAs by toposes and models of ASL^{--} by Cartesian closed categories. There, toposes form a complete class of models for simply-typed λ-calculus because of the Yoneda embedding, and they are very powerful models indeed (supporting a higher-order set theory), but there are also many interesting Cartesian closed categories that are not toposes [40]. While we cannot use our completeness result to justify ignoring the non-CKA models, it would be equally unwise to search for ASL^{--} models that can support proofs that cannot be represented at all (via embedding) in (weak) CKAs, because our completeness theorem shows that they can, if by indirect means.

As we mentioned in the introduction, Hoare and colleagues are promoting the idea of having an algebra of programs and deriving various laws (operational and logical) from the algebra. This is a very attractive point of view. We would be very much the poorer if we had, say, the Hilbert proof theory of propositional logic but not Boolean algebras, and the same is true of program logic: there is structure in the programs. However, while pretty, this paper suggests slight mismatches between the algebra (CKA and relatives) as currently formulated and program logics for concurrency (CSL and relatives), where the algebra does not accommodate some natural models. This suggests several questions for further work.

1. Is a further weakening of CKA possible which directly includes more existing models of generalizations of CSL (such as in ASL^{--} or Views), while maintaining the algebraic elegance and abstractness of CKA?
2. Might we turn the constraining nature of CKA, in that it assumes parallel composition of commands and spatial composition of assertions to be the same operation, into a virtue, by finding new concrete models of CKA which encompass the local reasoning about resources as in existing CSL models, while at the same time validating the powerful axioms of the existing or a similar CKA theory? Going further, could temporal reasoning as in [36,16,19,14,45] be accommodated?
3. Can the proof rules associated with existing CKA models which are not "resource like", such as the linear ordered model or tracelet (partially ordered) models [24], be used to prove specific concurrent programs in new ways? In particular, the linear and tracelet models fit the strongest structure, that of a Boolean CKA, which mixes Boolean algebra and commutative and non-commutative residuated monoids. Seen as a logic it is an alluring structure, mixing Boolean logic and two substructural logics together in the style of a variant of bunched logics [35,39]. Generally, if a model of this Boolean CKA logic could be found, and the logic used to prove concurrent programs represented in the model (transporting the programs as predicates viewpoint [22] to this rich logic), that would be remarkable.

Acknowledgements

We thank the anonymous referees for their comments, which helped improve the paper, and in particular uncovered an issue with a previous version of the definition of weak CKAs. We acknowledge the support of the EPSRC Programme Grant "Resource Reasoning" (grant number: EP/H008373/2). O'Hearn was supported by a Royal Academy of Engineering/Microsoft Research Chair.

Please cite this article in press as: P.W. O'Hearn et al., On the relation between Concurrent Separation Logic and Concurrent Kleene Algebra, Journal of Logical and Algebraic Methods in Programming (2014), http://dx.doi.org/10.1016/j.jlamp.2014.08.002

References

[1] R.-J. Back, J. von Wright, Refinement Calculus – A Systematic Introduction, Graduate Texts in Computer Science, Springer, 1998.
[2] R. Backhouse, R. Crole, J. Gibbons, in: Algebraic and Coalgebraic Methods in the Mathematics of Program Construction: International Summer School and Workshop, Revised Lectures, Oxford, UK, April 10–14, 2000, in: Lecture Notes in Computer Science, Springer, 2002.
[3] R. Bornat, C. Calcagno, P.W. O'Hearn, M.J. Parkinson, Permission accounting in separation logic, in: POPL, 2005.
[4] S. Brookes, A semantics for concurrent separation logic, Theor. Comput. Sci. 375 (1–3) (2007) 227–270.
[5] C. Calcagno, P.W. O'Hearn, H. Yang, Local action and abstract separation logic, in: LICS, 2007.
[6] E. Cohen, M. Dahlweid, M.A. Hillebrand, D. Leinenbach, M. Moskal, T. Santen, W. Schulte, S. Tobies, VCC: a practical system for verifying concurrent C, in: TPHOLs, 2009.
[7] E. Cohen, M. Moskal, W. Schulte, S. Tobies, Local verification of global invariants in concurrent programs, in: CAV, 2010.
[8] H.-H. Dang, P. Höfner, B. Möller, Algebraic separation logic, J. Log. Algebr. Program. 80 (6) (2011) 221–247.
[9] B.J. Day, On closed categories of functors, in: Reports of the Midwest Category Seminar, Springer-Verlag, Berlin–New York, 1970, pp. 1–38.
[10] B.J. Day, An embedding theorem for closed categories, in: Category Seminar, Sydney, Springer-Verlag, Berlin–New York, 1974, pp. 55–64.
[11] T. Dinsdale-Young, L. Birkedal, P. Gardner, M.J. Parkinson, H. Yang, Views: compositional reasoning for concurrent programs, in: POPL, 2013.
[12] T. Dinsdale-Young, M. Dodds, P. Gardner, M.J. Parkinson, V. Vafeiadis, Concurrent abstract predicates, in: 24th ECOOP, 2010.
[13] M. Dodds, X. Feng, M.J. Parkinson, V. Vafeiadis, Deny-guarantee reasoning, in: 18th ESOP, 2009.
[14] M. Fähndrich, M. Aiken, C. Hawblitzel, O. Hodson, G.C. Hunt, J.R. Larus, S. Levi, Language support for fast and reliable message-based communication in Singularity OS, in: EuroSys, 2006.
[15] X. Feng, R. Ferreira, Z. Shao, On the relationship between concurrent separation logic and assume-guarantee reasoning, in: 16th ESOP, 2007.
[16] M. Fu, Y. Li, X. Feng, Z. Shao, Y. Zhang, Reasoning about optimistic concurrency using a program logic for history, in: CONCUR, 2010.
[17] A. Gotsman, J. Berdine, B. Cook, Precision and the conjunction rule in concurrent separation logic, Electron. Notes Theor. Comput. Sci. 276 (2011) 171–190.
[18] A. Gotsman, J. Berdine, B. Cook, M. Sagiv, Thread-modular shape analysis, in: PLDI, 2007.
[19] A. Gotsman, N. Rinetzky, H. Yang, Verifying concurrent memory reclamation algorithms with grace, in: ESOP, 2013.
[20] Y. Guo, X. Feng, Z. Shao, P. Shi, Modular verification of concurrent thread management, in: APLAS, 2012.
[21] J. Hayman, G. Winskel, Independence and concurrent separation logic, Log. Methods Comput. Sci. 4 (1) (2008).
[22] C.A.R. Hoare, I.J. Hayes, J. He, C. Morgan, A.W. Roscoe, J.W. Sanders, I.H. Sørensen, J.M. Spivey, B. Sufrin, Laws of programming, Commun. ACM 30 (8) (1987) 672–686.
[23] C.A.R. Hoare, A. Hussain, B. Möller, P.W. O'Hearn, R.L. Petersen, G. Struth, On locality and the exchange law for concurrent processes, in: CONCUR, 2011.
[24] T. Hoare, B. Möller, G. Struth, I. Wehrman, Concurrent Kleene algebra and its foundations, J. Log. Algebr. Program. 80 (6) (2011) 266–296.
[25] T. Hoare, S. van Staden, In praise of algebra, Form. Asp. Comput. 24 (4–6) (2012) 423–431.
[26] T. Hoare, S. van Staden, The laws of programming unify process calculi, in: MPC, 2012.
[27] A. Hussain, On separation logic, session types and algebra, PhD thesis, Queen Mary, University of London, 2013.
[28] S.S. Ishtiaq, P.W. O'Hearn, BI as an assertion language for mutable data structures, in: POPL, 2001.
[29] J.B. Jensen, L. Birkedal, Fictional separation logic, in: 21st ESOP, 2012.
[30] D. Kozen, Kleene algebra with tests, in: Transactions on Programming Languages and Systems, May 1997.
[31] K.R.M. Leino, P. Müller, A basis for verifying multi-threaded programs, in: ESOP, 2009.
[32] R. Ley-Wild, A. Nanevski, Subjective auxiliary state for coarse-grained concurrency, in: POPL, 2013.
[33] A. McIver, T. Rabehaja, G. Struth, Probabilistic concurrent Kleene algebra, in: Quantitative Aspects of Programming Languages and Systems, 2013.
[34] P.W. O'Hearn, Resources, concurrency, and local reasoning, Theor. Comput. Sci. 375 (1–3) (2007) 271–307, prelim version appeared in CONCUR'04.
[35] P.W. O'Hearn, D.J. Pym, The logic of bunched implications, Bull. Symb. Log. 5 (2) (1999) 215–244.
[36] P.W. O'Hearn, N. Rinetzky, M.T. Vechev, E. Yahav, G. Yorsh, Verifying linearizability with hindsight, in: PODC, 2010.
[37] D.M.R. Park, On the semantics of fair parallelism, in: Abstract Software Specifications, 1979.
[38] P. Philippaerts, J.T. Mühlberg, W. Penninckx, J. Smans, B. Jacobs, F. Piessens, Software verification with VeriFast: industrial case studies, Sci. Comput. Program. 82 (1) (February 2013) 77–97.
[39] D. Pym, P. O'Hearn, H. Yang, Possible worlds and resources: the semantics of BI, Theor. Comput. Sci. 315 (1) (2004) 257–305.
[40] D. Scott, Relating theories of the λ-calculus, in: To H.B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalisms, 1980.
[41] K. Svendsen, L. Birkedal, M.J. Parkinson, Modular reasoning about separation of concurrent data structures, in: 22nd ESOP, 2013.
[42] V. Vafeiadis, Modular fine-grained concurrency verification, PhD thesis, University of Cambridge, 2007.
[43] V. Vafeiadis, Concurrent separation logic and operational semantics, Electron. Notes Theor. Comput. Sci. 276 (2011) 335–351.
[44] V. Vafeiadis, M.J. Parkinson, A marriage of rely/guarantee and separation logic, in: CONCUR, 2007.
[45] J. Villard, É. Lozes, C. Calcagno, Proving copyless message passing, in: APLAS, 2009.
[46] H. Yang, P.W. O'Hearn, A semantic basis for local reasoning, in: FoSSaCS, 2002.

Please cite this article in press as: P.W. O'Hearn et al., On the relation between Concurrent Separation Logic and Concurrent Kleene Algebra, Journal of Logical and Algebraic Methods in Programming (2014), http://dx.doi.org/10.1016/j.jlamp.2014.08.002