Foundations of concurrent Kleene algebra

Charles Antony Richard Hoare, Bernhard Möller, Georg Struth, Ian Wehrman

Angaben zur Veröffentlichung / Publication details:

Hoare, Charles Antony Richard, Bernhard Möller, Georg Struth, and Ian Wehrman. 2009. "Foundations of concurrent Kleene algebra." Augsburg: Institut für Informatik, Universität Augsburg.

Universität Augsburg

Foundations of Concurrent Kleene Algebra

C.A.R. Hoare, B. Möller, G. Struth and I. Wehrman

Report 2009-05 April 2009

Institut für Informatik
D-86135 Augsburg

Foundations of Concurrent Kleene Algebra

C.A.R. Hoare[1], B. Möller[2], G. Struth[3], and I. Wehrman[4]

[1] Microsoft Research, Cambridge, UK
[2] Universität Augsburg, Germany
[3] University of Sheffield, UK
[4] University of Texas at Austin, USA

Abstract. A Concurrent Kleene Algebra offers two composition operators, one that stands for sequential execution and the other for concurrent execution [9]. In this paper we investigate the abstract background of this law in terms of independence relations on which a concrete trace model of the algebra is based. Moreover, we show the interdependence of the basic properties of such relations and two further laws that are essential in the application of the algebra to a Jones style rely/guarantee calculus. Finally we reconstruct the trace model in a more abstract setting based on the notion of atoms from lattice theory.

1 Introduction

A Concurrent Kleene Algebra (CKA) is one which offers two composition operators, one that stands for sequential execution and the other for concurrent execution. They are related by an inequational form of the exchange law $(a \circ b) \bullet (c \circ d) = (a \bullet c) \circ (b \bullet d)$ of two-category or bicategory theory (e.g. [16]).

The applicability of the algebra to a partially-ordered trace model of program execution semantics and to the validation familiar proof rules for sequential programs (Hoare triples) and for concurrent programming (Jones's rely/guarantee calculus) is shown in [9]. The mentioned trace model is based on a dependence relation between atomic events.

In the present paper we investigate how the laws of concurrent Kleene algebra reflect this relation; we show that two central laws are equivalent to its transitivity and acyclicity, resp. The traces obeying a generalised version of the second law are characterised in terms of convexity w.r.t the dependence relation. Moreover we introduce the notion of an event-based concurrent Kleene algebra which is a more abstract version of the concrete trace model, based on the notions of atoms and irreducible elements. We show that in such algebras the dependence relation can be recovered from the operations of sequential and concurrent composition. Most of our reasoning has been checked by computer.

Sect. 2 summarises the definitions of the trace model and its essential operators. In Sect. 3 we develop an abstract calculus of independence relations, both in formulas and diagrammatic rules. Sect. 4 presents quantales as a fundamental structure and gives the axiomatisation of CKAs in terms of quantales. Sect. 5 gives a definition of invariants as used in the mentioned rely/guarantee calculus. In Sect. 6 we establish the equivalence of two fundamental laws with (weak) acyclicity and transitivity of the basic dependence relation. Sect. 7 presents a simplified rely/guarantee calculus. Finally, Sect. 8 develops the notion of event-based CKAS and reconstructs the trace model and the dependence relation in terms of that notion. Appendix A summarises the laws characterising the various structures involved. Appendix B we show a typical input file for the automated theorem prover Prover9 [17] with which all the algebraic proofs have been reconstructed automatically.

2 Operations on Traces and Programs

In this section we present a concrete model of Concurrent Kleene Algebra which serves as a motivation of the abstract algebraic treatment in the later sections.

We assume a set EV of *event occurrences* and a *dependence relation* $\rightarrow \subseteq EV \times EV$ between them: $e \rightarrow f$ indicates occurrence of a data flow or control flow from event e to event f.

Definition 2.1 A *trace* is a set of events; the set of all traces over EV is $TR(EV) =_{df} \mathcal{P}(EV)$. A *program* is a set of traces; the set of all programs is $PR(EV) =_{df} \mathcal{P}(TR(EV))$.

We deliberately keep the definition of traces and programs so liberal to accommodate systems with very loose coupling of events; "conventional" linear traces can, e.g., be obtained by including time stamps into the events and defining the dependence relation such that it respects time.

Examples of very simple programs are the following. The program skip, which does nothing, is defined as $\{\emptyset\}$, and the program $[e]$, which does only $e \in EV$, is $\{\{e\}\}$. The program $\mathsf{false} =_{df} \emptyset$ has no traces, and therefore cannot be executed at all. It serves the rôle of the 'miracle' [18] in the development of programs by stepwise refinement.

Following [10] we will define four operators on programs P and Q:

- $P * Q$ fine-grain concurrent composition, allowing dependences between P and Q;
- $P \,;\, Q$ weak sequential composition, forbidding dependence of P on Q;
- $P \parallel Q$ disjoint parallel composition, with no dependence in either direction;
- $P \,[\,]\, Q$ alternation – exactly one of P or Q is executed, if possible.

To express the restrictions in this list we introduce the following independence relation.

Definition 2.2 For traces tp, tq we define the *independence relation* by

$$tp \not\leftarrow tq \;\Leftrightarrow_{df}\; \neg\exists\, p \in tp, q \in tq : q \rightarrow p \ .$$

Viewing tp as a set of events that should occur before all the ones in tq, one can read $tp \not\leftarrow tq$ as the requirement that tp must not depend on its "future" tq.

Now, for each operator $\circ \in \{*, ;, \parallel, [\,]\}$ we define an associated binary relation $(\circ)$ between traces such that for programs P, Q we can define generically

$$P \circ Q \;=_{df}\; \{tp \cup tq \mid tp \in P \,\wedge\, tq \in Q \,\wedge\, tp\,(\circ)\,tq\} \ . \tag{1}$$

From this definition it is immediate that $\circ$ distributes through arbitrary unions of families of programs and hence is $\subseteq$-isotone and false-strict, i.e., $\mathsf{false} \circ P = \mathsf{false} = P \circ \mathsf{false}$. Moreover, skip is a neutral element for $\circ$, i.e.,

$$\mathsf{skip} \circ P \;=\; P \;=\; P \circ \mathsf{skip} \ . \tag{2}$$

Finally, if $(\circ)$ is symmetric then $\circ$ is commutative.

Now the above informal descriptions are captured by the definitions

$$\begin{array}{rcl} tp\,(*)\,tq & \Leftrightarrow_{df} & tp \cap tq = \emptyset \ , \\ tp\,(;)\,tq & \Leftrightarrow_{df} & tp\,(*)\,tq \,\wedge\, tp \not\leftarrow tq \ , \\ tp\,(\parallel)\,tq & \Leftrightarrow_{df} & tp\,(;)\,tq \,\wedge\, tq \not\leftarrow tp \ , \\ tp\,([\,])\,tq & \Leftrightarrow_{df} & tp = \emptyset \,\vee\, tq = \emptyset \ . \end{array}$$

It is clear that $([\,]) \subseteq (\parallel) \subseteq (;) \subseteq (*)$ and that $(*)$ and $(\parallel)$ are symmetric.

Another essential operator is the union operator which again is $\subseteq$-isotone and distributes through arbitrary unions. However, it is *not* false-strict.

By the Tarski-Kleene fixpoint theorems hence all recursion equations involving only the operations mentioned have $\subseteq$-least solutions which can be approximated by the familiar fixpoint iteration starting from false. Use of union in such a recursion enables non-trivial fixpoints.

3 Independence Calculus and Exchange Laws

To prove the most essential laws about the interaction of our operators we now give a slightly more abstract treatment. We start by observing that an equivalent relation-algebraic formulation of the independence relation $\not\leftarrow$ is $tp \not\leftarrow tq \Leftrightarrow tp \times tq \cap \rightarrow^{\smile} = \emptyset$, where $\rightarrow^{\smile}$ is the converse of $\rightarrow$. By straightforward set theory this entails

$$(tp \cup tq) \not\leftarrow tr \Leftrightarrow tp \not\leftarrow tr \wedge tq \not\leftarrow tr \ ,$$
$$tp \not\leftarrow (tq \cup tr) \Leftrightarrow tp \not\leftarrow tq \wedge tp \not\leftarrow tr \ .$$

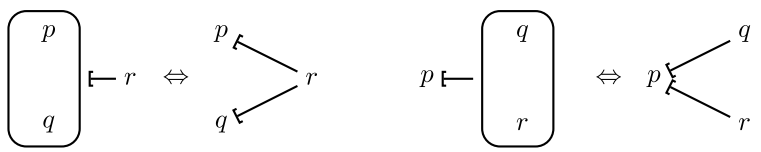
It turns out that these bilinearity properties are the essence of the characteristic laws about the interplay of our various operators. This motivates the following definition.

Definition 3.1 An *aggregation algebra*, a structure $(A, +)$ consisting of a set A and a binary operation $+ : A \times A \to A$. An *independence relation* on an aggregation algebra is a bilinear relation $R \subseteq A \times A$, i.e.,

$$(p+q)\,R\,r \Leftrightarrow p\,R\,r \wedge q\,R\,r \ ,$$
$$p\,R\,(q+r) \Leftrightarrow p\,R\,q \wedge p\,R\,r \ .$$

In our example of traces and programs, A would be the set of traces and $+$ would be trace union. For now, we will however consider an aggregation algebra as absolutely free, i.e., $+$ need not satisfy any laws. Later we will need aggregation algebras that are (commutative) semigroups or monoids. The first condition on R says that an aggregate $p+q$ is independent of r iff both its parts p and q are independent of r. The second condition says that p is independent of the aggregate $q+r$ iff it is independent of both its parts, q and r. An independence relation on $(TR(EV), \cup)$ is $\not\leftarrow$.

We can visualise the independence conditions by the following diagrams.



The ovals display aggregates. The letters in the ovals represent the entities that form their parts. In the first diagram, the oval around p and q denotes the aggregate $p+q$. The arrows denote the independence relation R, where the sign [means that any flow of dependence is blocked there. In the leftmost diagram, the arrow relates the aggregate $p+q$ to r, hence the aggregate formed by p and q is independent of r. In its neighbour diagram, there is no aggregate and both p and q are related to r, hence independent of r. The equivalence between the diagrams visualises the first bilinearity law. Analogous remarks apply to the second pair of diagrams.

An important tool for a uniform treatment of our operators from Sect. 2 is the following lemma, whose proof is straightforward.

Lemma 3.2

1. *The set of independence relations on an aggregation algebra is closed under intersection.*
2. *The relations* $(*), (;), (\|)$ *and* $([])$ *are independence relations on* $(TR(EV), \cup)$.

We now consider the interplay of two independence relations R and S on an aggregation algebra A.

Lemma 3.3 *Let R and S be independence relations on an aggregation algebra $(A, +)$ such that $R \subseteq S$. Then*

1. $(p + q)\, R\, r \wedge p\, S\, q \;\Rightarrow\; p\, S\, (q + r) \wedge q\, R\, r$.
2. $p\, R\, (q + r) \wedge q\, S\, r \;\Rightarrow\; (p + q)\, S\, r \wedge p\, R\, q$.

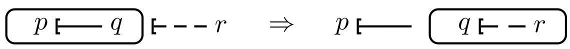
Proof.

$$\begin{aligned}
(p + q)\, R\, r \wedge p\, S\, q &\Leftrightarrow p\, R\, r \wedge q\, R\, r \wedge p\, S\, q \\
&\Rightarrow p\, S\, r \wedge q\, R\, r \wedge p\, S\, q \\
&\Leftrightarrow q\, R\, r \wedge p\, S\, (q + r).
\end{aligned}$$

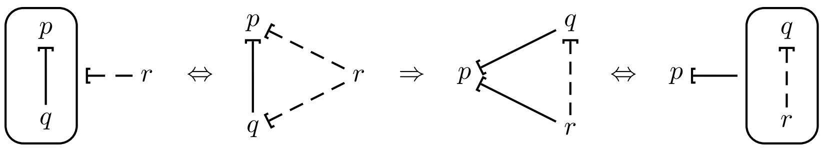
$$\begin{aligned}
p\, R\, (q + r) \wedge q\, S\, r &\Leftrightarrow p\, R\, q \wedge p\, R\, r \wedge q\, S\, r \\
&\Rightarrow p\, R\, q \wedge p\, S\, r \wedge q\, S\, r \\
&\Leftrightarrow p\, R\, q \wedge (p + q)\, S\, r.
\end{aligned}$$

□

Obviously, we now must introduce two different kinds of arrows in diagrams. The first law can be visualised as



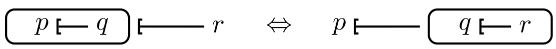
The second law looks similar. A diagrammatic proof of the first law is



A simple consequence is the following.

Corollary 3.4 *Let R be an independence relation on an aggregation algebra $(A, +)$. Then*

$$(p + q)\, R\, r \wedge p\, R\, q \Leftrightarrow q\, R\, r \wedge p\, R\, (q + r).$$



Proof. Set $S = R$ in Lemma 3.3. □

We can also prove the following *exchange laws* which are crucial for concurrent Kleene algebra. Again, we write $R^{\smile}$ for the relational converse of R.

Theorem 3.5 *Let R and S be independence relations on an aggregation algebra $(A, +)$ such that $R \subseteq S$ and S is symmetric. Then*

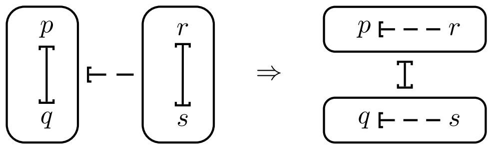
$$(p + q)\, R\, (r + s) \wedge p\, S\, q \wedge r\, S\, s \;\Rightarrow\; p\, R\, r \wedge q\, R\, s \wedge (p + r)\, S\, (q + s)\ .$$

Proof.

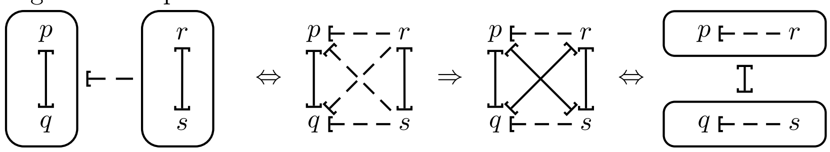
$$\begin{aligned}
&(p + q)\, R\, (r + s) \wedge p\, S\, q \wedge r\, S\, s \\
&\qquad \Leftrightarrow p\, R\, r \wedge q\, R\, r \wedge p\, R\, s \wedge q\, R\, s \wedge p\, S\, q \wedge r\, S\, s \\
&\qquad \Rightarrow p\, R\, r \wedge q\, S\, r \wedge p\, S\, s \wedge q\, R\, s \wedge p\, S\, q \wedge r\, S\, s \\
&\qquad \Rightarrow p\, R\, r \wedge q\, R\, s \wedge r\, S\, q \wedge (p + r)\, S\, (s) \wedge p\, S\, q \\
&\qquad \Rightarrow p\, R\, r \wedge q\, R\, s \wedge (p + r)\, S\, (q) \wedge (p + r)\, S\, (s) \\
&\qquad \Leftrightarrow p\, R\, r \wedge q\, R\, s \wedge (p + r)\, S\, (q + s)\ .
\end{aligned}$$

□

The diagrammatic statement of the exchange law (neglecting hypotheses) is



A diagrammatic proof is



As immediately obvious from the diagrams, the hypotheses also entail

$$(p+q)\,R\,(r+s) \wedge p\,S\,q \wedge r\,S\,s \;\Rightarrow\; p\,R\,s \wedge q\,R\,r \wedge (p+s)\,S\,(q+r)\ . \tag{3}$$

The proofs in this section, logical and diagrammatical, are only intended to give a flavour of the approach. In fact, the former ones have all been automated, hence formally verified, with Prover9 [17] and could as well have been omitted. Proofs at this level of complexity present no obstacle to ATP systems.

We now apply our results to our special aggregation algebra $(TR(EV), \cup)$.

Lemma 3.6 *Let* $\circ, \bullet \in \{*, ;, \|, []\}$.
1. *If* $(\bullet) \subseteq (\circ)$ *and* $(\circ)$ *is symmetric then* $(P \circ Q) \bullet (R \circ S) \subseteq (P \bullet R) \circ (Q \bullet S)$.
2. *If* $(\bullet) \subseteq (\circ)$ *then* $(P \circ Q) \bullet R \subseteq P \circ (Q \bullet R)$ *and* $P \bullet (Q \circ R) \subseteq (P \bullet Q) \circ R$.
3. $\circ$ *is associative.*

Proof.
1. For traces $tp \in P, tq \in Q, tr \in R, ts \in S$ we have by (1) and Theorem 3.5

$$\begin{array}{ll} & (tp \cup tq) \cup (tr \cup ts) \in (P \circ Q) \bullet (R \circ S) \\ \Leftrightarrow & tp(\circ)tq \;\wedge\; tr(\circ)ts \;\wedge\; (tp \cup tq))(\bullet)(tr \cup ts) \\ \Rightarrow & tp(\bullet)tr \;\wedge\; tq(\bullet)ts \;\wedge\; (tp \cup tr))(\circ)(tq \cup ts) \\ \Leftrightarrow & (tp \cup tr) \cup (tq \cup ts) \in (P \bullet R) \circ (Q \bullet S)\ . \end{array}$$

   Since $(tp \cup tq) \cup (tr \cup ts) = (tp \cup tr) \cup (tq \cup ts)$, we are done.
2. Similar to the proof of Part 1, using Lemma 3.3.
3. Use the two previous laws with $\bullet = \circ$. □

A particularly important special case of Part 1 is the exchange law

$$(P * Q)\,;(R * S) \subseteq (P\,;R) * (Q\,;S)\ . \tag{4}$$

In the remainder of this paper we shall no longer deal with the less interesting operators $\|$ and $[]$.

4 Quantales and Concurrent Kleene Algebras

We now abstract further from the concrete example of traces and programs.

Definition 4.1 A *semiring* is a structure $(S, +, 0, \cdot, 1)$ such that $(S, +, 0)$ is a commutative monoid, $(S, \cdot, 1)$ is a monoid, multiplication distributes over addition in both arguments and 0 is a left and right annihilator with respect to multiplication ($a \cdot 0 = 0 = 0 \cdot a$). A semiring is *idempotent* if its addition is.

In an idempotent semiring, the relation $\leq$ defined by $a \leq b \Leftrightarrow_{df} a + b = b$ is a partial ordering, in fact the only partial ordering on S for which 0 is the least element and for which addition and multiplication are isotone in both arguments. It is therefore called the *natural ordering* on S. This makes S into a semilattice with addition as join and least element 0.

Definition 4.2 A *quantale* [19] or *standard Kleene algebra* [5] is an idempotent semiring that is a complete lattice under the natural order and in which composition distributes over arbitrary suprema.

Let now $PR(EV)$ denote the set of all programs over the event set EV. From the observations in Sect. 2 the following is immediate:

Lemma 4.3 $(PR(EV), \cup, \mathsf{false}, *, \mathsf{skip})$ *and* $(PR(EV), \cup, \mathsf{false}, ;, \mathsf{skip})$ *both are quantales.*

In a quantale S, finite and infinite iteration * and $^\omega$ are defined by

$$a^* = \mu x \,.\, 1 + a \cdot x \ , \qquad\qquad a^\omega = \nu x \,.\, a \cdot x \ ,$$

where μ and ν denote the least and greatest fixpoint operators. The star used here is not to be confused with the separation operator $*$ above; it should also be noted that a^ω in [1] corresponds to $a^* + a^\omega$ in the quantale setting.

It is well known that then $(S, +, \cdot, 0, 1, ^*)$ forms a Kleene algebra [13]. From this we obtain many useful laws for free. As instances we mention

$$a^* \cdot a^* = (a^*)^* = a^* \ , \qquad (a \cdot b)^* \cdot a = a \cdot (b \cdot a)^* \ , \qquad (a + b)^* = a^* \cdot (b \cdot a^*)^* \ .$$

Since in a quantale the function defining star is continuous, Kleene's fixpoint theorem shows that $a^* = \bigsqcup_{i \in \mathbb{N}} a^i$. Moreover, we have the star induction rules

$$b + a \cdot x \le x \ \Rightarrow\ a^* \cdot b \le x \ , \qquad b + x \cdot a \le x \ \Rightarrow\ b \cdot a^* \le x \ . \tag{5}$$

Hence in $(PR(EV), \cup, \mathsf{false}, *, \mathsf{skip})$ and $(PR(EV), \cup, \mathsf{false}, ;, \mathsf{skip})$ the program P^* consists of all finite disjoint unions and all finite sequential compositions of traces in P, resp. In the latter case P^* is denoted by P^∞ in [10].

If, in addition, the complete lattice $(S, \le)$ in a quantale is completely distributive, i.e., if $+$ distributes over arbitrary infima, then $(S, +, \cdot, 0, 1, ^*, ^\omega)$ forms an omega algebra [4]. Again this entails many useful laws, e.g.,

$$(a \cdot b)^\omega = a \cdot (b \cdot a)^\omega \ , \qquad (a + b)^\omega = a^\omega + a^* \cdot b \cdot (a + b)^\omega \ .$$

Since $PR(EV)$ is a power set lattice, it is completely distributive. Hence both program quantales also admit infinite iteration with all its laws. The infinite iteration $^\omega$ in $(PR(EV), \cup, \mathsf{false}, *, \mathsf{skip})$ is similar to the unbounded parallel spawning $!P$ in the π-calculus [21]. The abstract combination of two quantales leads to the following definition [9].

Definition 4.4 A *concurrent Kleene algebra (CKA)* is a structure $(S, +, 0, *, ;, 1)$ such that $(S, +, *, 0, 1)$ and $(S, +, ;, 0, 1)$ are quantales linked by the exchange axiom

$$(a * b) \,;\, (c * d) \le (b \,;\, c) * (a \,;\, d) \ .$$

Compared with the original exchange law (4) this one has its free variables in a different order. This does no harm, since the concrete $*$ operator on programs is commutative and hence satisfies the above law as well. Hence we have

Corollary 4.5 $(PR(EV), \cup, \mathsf{false}, *, ;, \mathsf{skip})$ *is a CKA.*

The reason for our formulation of the exchange axiom here is that this form of the law implies commutativity of $*$ as well as $a \,;\, b \le a * b$ and hence saves two axioms. We list some important consequences of this axiomatisation; the proofs are given in [9].

Lemma 4.6 *In a CKA the following laws hold.*

1. $a * b = b * a$.
2. $(a * b) \,;\, (c * d) \le (a \,;\, c) * (b \,;\, d)$.
3. $a \,;\, b \ \le\ a * b$.
4. $(a * b) \,;\, c \ \le\ a * (b \,;\, c)$.
5. $a \,;\, (b * c) \ \le\ (a \,;\, b) * c$.

5 Invariants

For the further development we need to deal with the set of events a program may use.

Definition 5.1 A *power invariant* is a program R of the form $R = \mathcal{P}(E)$ for a set $E \subseteq EV$ of events.

It consists of all possible traces that can be formed from events in E and hence is the most general program using only those events. The smallest power invariant is $\mathsf{skip} = \mathcal{P}(\emptyset)$. The term "invariant" expresses that often a program relies (whence the name R) on the assumption that its environment only uses events from a particular subset, i.e., preserves the invariant of staying in that set. We will now investigate the properties of power invariants.

To this end we want to define a function that forms from a program the smallest invariant containing it. We denote, for a program P, by $|P| =_{df} \bigcup P$ the set of all events occurring in traces of P; when convenient, $|P|$ can also be considered as a trace.

It is straightforward to check that $|\_|$ distributes through arbitrary unions. Hence it has an upper adjoint F, defined by the Galois connection

$$P \subseteq F(X) \Leftrightarrow |P| \subseteq X \ .$$

This entails $F(X) = \mathcal{P}(X)$ and $|\mathcal{P}(X)| = X$. Moreover, as adjoints of a Galois connection, $\mathcal{P}(\_)$ and $|\_|$ are $\subseteq$-isotone. Setting $X = |P|$ we obtain $P \subseteq \mathcal{P}(|P|)$. Finally, $\mathcal{P}(X) \subseteq \mathcal{P}(Y) \Leftrightarrow X \subseteq Y$.

Motivated by the above remarks we now define $\mathsf{INV}(P) =_{df} \mathcal{P}(|P|)$. Then $\mathsf{INV}(P)$ is the most general program that can be formed from the events of P. As a composition of isotone functions, INV is isotone again.

An *invariant* is a program R with $R = \mathsf{INV}(R)$. This means that invariants are fixpoints of an isotone function and hence form a complete lattice under the inclusion order.

The operation ∇ from [10] and INV are interdefinable. To this end we set $\mathsf{SINGLES}(P) =_{df} \{\{e\} : \{e\} \in P\}$. Then

$$\mathsf{INV}(\mathsf{SINGLES}(Q)) \ = \ Q \nabla Q \ , \qquad Q \nabla R \ = \ \mathsf{INV}(\mathsf{SINGLES}(Q \cup R)) \ .$$

We shall use INV, since it leads to simpler and more intuitive formulations.

We give a few useful properties of INV.

Theorem 5.2

1. $\mathsf{INV}(P)$ *is the smallest invariant containing* P.
2. $\mathsf{INV}(\mathsf{INV}(P)) = \mathsf{INV}(P)$*; hence* $\mathsf{INV}(P)$ *is an invariant.*
3. INV *is a closure operator.*
4. $\mathsf{skip} \subseteq \mathsf{INV}(P)$.

Proof. 1. We have already seen above that $P \subseteq \mathsf{INV}(P)$. Let S be another invariant with $P \subseteq S$. Then, by isotony of INV and the definition of invariants, $\mathsf{INV}(P) \subseteq \mathsf{INV}(S) = S$.

2. Since, as remarked above, $|\mathcal{P}(X)| = X$, we have

$$\mathsf{INV}(\mathsf{INV}(P)) \ = \ \mathcal{P}(|\mathcal{P}(|P|)|) \ = \ \mathcal{P}(|P|) \ = \ \mathsf{INV}(P) \ .$$

3. By Part 1 INV is extensive. By the Galois connection it is isotone and by Part 2 it is idempotent.
4. Immediate from the definition of INV. □

Since INV is a closure operator we have the following (see e.g. [3]).

Corollary 5.3 *For set $\mathcal{R}$ of power invariants, $\bigcap \mathcal{R}$ and $\mathsf{INV}(\bigcup \mathcal{R})$ are the meet and join of $\mathcal{R}$ in the complete lattice of invariants, resp.*

We now again abstract from the concrete case of programs.

Definition 5.4 A *CKA with invariants* is a structure $(S, +, 0, *, ;, 1, \iota)$ such that $(S, +, 0, *, ;, 1)$ is a CKA and $\iota : S \to S$ is a closure operator that additionally satisfies, for all $a, b \in S$,

$$1 \leq \iota\, a \;, \qquad\qquad \iota\,(a * b) \leq \iota\,(a + b) \;.$$

An *invariant* is an element $a \in S$ with $\iota\, a = a$.

In [9] a more specific view of invariants is taken: there an invariant is an element r with $1 \leq r$ and $r * r \leq r$. This entails that the invariants are precisely the fixpoints of the finite iteration operator * w.r.t. concurrent composition $*$. This still allows proving many of the above properties, but does not characterise power invariants and hence is not adequate for all purposes. However, we have the following connection.

Lemma 5.5 *Defining in a CKA $\iota\, a =_{df} a^*$ makes it a CKA with invariants.*

Proof. By standard Kleene algebra * is a closure operator with $1 \leq a^*$. The remaining axiom is shown by star induction (5) and isotony as follows:

$$\begin{array}{l} (a * b)^* \leq (a + b)^* \;\Leftarrow\; 1 + a * b * (a + b)^* \leq (a + b)^* \;\Leftarrow \\ \quad 1 \leq (a + b)^* \;\wedge\; (a + b) * (a + b) * (a + b)^* \leq (a + b)^* \;\Leftrightarrow\; \mathsf{TRUE}\;. \end{array} \qquad \square$$

Again it is clear that the invariants in the abstract sense form a complete lattice with properties analogous to those of Corollary 5.3. Moreover, one has the usual Galois connection for closures (e.g. [7]):

$$a \leq \iota\, b \;\Leftrightarrow\; \iota\, a \leq \iota\, b \;. \tag{6}$$

With this definition we can give a uniform abstract proof of idempotence of operators on invariants.

Theorem 5.6 *Consider a CKA s with invariants. Let $\circ$ be an isotone binary operation on s that has 1 as neutral element and satisfies $\forall\, a, b : \iota\,(a \circ b) \subseteq \iota\,(a + b)$. Then for invariant r we have $r \circ r = r$.*

Proof. We first show $r \circ r \leq r$. By extensivity of ι, the assumption and $r + r = r$ as well as invaraince of r we have $r \circ r \subseteq \iota\,(r \circ r) \subseteq \iota\, r = r$. The converse inclusion is shown by $r = r \circ 1 \leq r \circ r$, using neutrality of 1, the axiom $1 \leq \iota\, a$ and isotony of $\circ$. $\square$

Next we define the *guarantee relation* slightly more liberally than [10] by

$$a \;\mathsf{guar}\; b \;\Leftrightarrow_{df}\; \iota\, a \leq \iota\, b \;.$$

If b is an invariant, i.e., $b = \iota\, b$, we obtain by (6)

$$a \;\mathsf{guar}\; b \;\Leftrightarrow\; \iota\, a \leq \iota\, b \;\Leftrightarrow\; a \leq \iota\, b \;\Leftrightarrow\; a \leq b \;.$$

We have the following properties.

Theorem 5.7

1. *If g is an invariant then $1 \;\mathsf{guar}\; g$.*
2. *If g, g' are invariants and $\circ$ is again an isotone binary operation satisfying $\forall\, a, b : \iota\,(a \circ b) \leq \iota\,(a + b)$ then*

$$b \;\mathsf{guar}\; g \;\;\wedge\;\; b' \;\mathsf{guar}\; g' \;\;\Rightarrow\;\; (b \circ b') \;\mathsf{guar}\; (g + g') \;.$$

3. *For the concrete case of programs,* $[e]\ \mathsf{guar}\ G \Leftrightarrow e \in |G|$*.*

Proof. 1. Immediate from the axioms and the above remark on guar.

2.
$$
\begin{array}{ll}
 & b\ \mathsf{guar}\ g \wedge b'\ \mathsf{guar}\ g' \\
\Leftrightarrow & \quad \{\!\!\{ \text{ above remark on } \mathsf{guar} \}\!\!\} \\
 & \iota\, b \le g \wedge \iota\, b' \le g' \\
\Rightarrow & \quad \{\!\!\{ \text{ isotony of } + \}\!\!\} \\
 & \iota\, b + \iota\, b' \le g + g' \\
\Rightarrow & \quad \{\!\!\{ \text{ isotony of } \iota \}\!\!\} \\
 & \iota\,(b + b') \le g + g' \\
\Rightarrow & \quad \{\!\!\{ \text{ assumption about } \circ \}\!\!\} \\
 & \iota\,(b \circ b') \le g + g' \\
\Leftrightarrow & \quad \{\!\!\{ \text{ extensivity of } \iota \}\!\!\} \\
 & \iota\,(b \circ b') \le \iota\,(g + g') \\
\Leftrightarrow & \quad \{\!\!\{ \text{ definition } \}\!\!\} \\
 & (b \circ b')\ \ \mathsf{guar}\ \ (g + g')\ .
\end{array}
$$

3. By the definitions and the Galois connection for $|\_|$,

$$[e]\ \mathsf{guar}\ G \Leftrightarrow \mathsf{INV}([e]) \subseteq \mathsf{INV}(G) \Leftrightarrow \{e\} \subseteq \mathsf{INV}(G) \Leftrightarrow e \in |G|\ . \qquad \square$$

6 Characterising Dependence

In [9] it is shown that the definitions of $*$ and ; for concrete programs in terms of the transitive closure $\rightarrow^+$ of the dependence relation $\rightarrow$ entail two important further laws that are essential for the rely/guarantee calculus to be presented below:

Theorem 6.1 *Let $R = \mathcal{P}(E)$ be a power invariant in $PR(EV)$.*

1. *If $\rightarrow$ is acyclic and $e \in EV$ then*

$$R * [e] \subseteq R\,;[e]\,;R\ .$$

2. *If $\rightarrow$ is transitive then for all $P, Q \in PR(EV)$ we have*

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

But in fact, in a sense also the reverse implications hold. To formulate them we need a further notion.

Definition 6.2 We call $\rightarrow$ *weakly acyclic* if for all events e, f,

$$e \rightarrow^+ f \rightarrow^+ e \ \Rightarrow\ f = e\ ,$$

and *weakly transitive* if

$$e \rightarrow f \rightarrow g \ \Rightarrow\ (e = g \vee e \rightarrow g)\ .$$

Weak acaclicity means that $\rightarrow$ may at most have immediate self-loops (which cannot be "detected" by the ; operator, since it is defined in terms of distinct events only).

Theorem 6.3

1. *If $R * [e] \subseteq R\,;[e]\,;R$ is valid for all power invariants R and events e then $\rightarrow$ is weakly acyclic.*
2. *If $R * (P\,;Q) \subseteq (R * P)\,;(R * Q)$ is valid for all power invariants R and programs P, Q then $\rightarrow$ is weakly transitive.*

Proof of Part 2.
Assume events p, q, r with $q \rightarrow r$ and $r \rightarrow p$ but $q \not\rightarrow p$. This implies $q \neq r$ and $r \neq p$. Assume now $p \neq q$ and set $P =_{df} [p]$, $Q =_{df} [q]$ and $R =_{df} [\,] \cup [r]$. Then $P\,;Q = [p, q]$ and $R * (P\,;Q) = [p, q] \cup [r, p, q]$. Moreover, $R * P = [p] \cup [r, p]$ and $R * Q = [q] \cup [r, q]$, hence $(R * P)\,;(R * Q) = [p, q]$ contradicting the assumed property. Therefore we must have $p \leftarrow q$. $\square$

We abstract this into the following

Definition 6.4 A CKA S with invariants is $*$-*distributive* if all invariants r and all $a, b \in S$ satisfy

$$r * (a\,;b) \leq (r * a)\,;(r * b)\ .$$

We still have to prove Part 1. Rather than doing this directly we investigate a slightly more general property which is equivalent to an interesting property of traces more general than single-event ones.

Definition 6.5 A trace tp is *convex* if for all events $p, q \in tp$ and arbitrary event f we have

$$p \rightarrow^+ f \rightarrow^+ q \;\Rightarrow\; f \in tp\ .$$

A convex trace can be considered "closed" under dependence.

For the following lemma we introduce the auxiliary function $dep(tp) =_{df} \{q \mid \exists\, p \in tp : q \rightarrow^+ p\}$ on traces tp. Hence $dep(tp)$ consists of all events on which some event of tp depends. Then we have

Lemma 6.6 Let tp be a trace and assume that $R * \{tp\} \subseteq R\,;\{tp\}\,;R$ holds for all power invariants R.

1. Dependence between a trace and any event outside occurs at most in one direction, i.e., for any event $f \notin tp$ we have

$$tp \cap dep(\{f\}) = \emptyset \;\;\vee\;\; \{f\} \cap dep(tp) = \emptyset\ .$$

2. As a consequence, tp is convex.

Proof. 1. Set $R =_{df} \mathcal{P}(\{f\})$. By assumption the trace $tr = \{f\} \in R$ can be split as $tr = tr'\,;tr''$ such that $tr * tp = tr'\,;tp\,;tr''$.
Case 1: $tr' = \{f\} \wedge tr'' = \emptyset$. Hence $tr * tp = \{f\}\,;tp$. This implies $\{f\} \cap dep(tp) = \emptyset$.
Case 2: $tr' = \emptyset \wedge tr'' = \{f\}$. Hence $tr * tp = tp\,;\{f\}$. This implies $tp \cap dep(\{f\}) = \emptyset$.
2. Suppose $f \notin tp$. The premise $p \rightarrow^+ f$ implies $p \in tp \cap dep(\{f\})$ while $f \rightarrow^+ q$ implies $f \in \{f\} \cap dep(tp)$. In particular, both sets are non-empty, contradicting Part 1. $\square$

The case of singleton traces is covered as follows:

Lemma 6.7 All traces $\{e\}$ are convex iff $\rightarrow$ is weakly acyclic.

Proof. ($\Rightarrow$) Assume $e \rightarrow^+ f \rightarrow^+ e$. Then by the assumed convexity of $\{e\}$ we get $f \in \{e\}$, i.e., $f = e$.
($\Leftarrow$) Assume $p \rightarrow^+ f \rightarrow^+ q$ for $p, q \in \{e\}$, i.e., $e \rightarrow^+ f \rightarrow^+ e$. Then by the assumed weak acyclicity $f = e$, i.e., $f \in \{e\}$. $\square$

We now want to show that also the reverse of Lemma 6.6 holds.

Lemma 6.8 Let tp be convex. Then for all power invariants R the formula $R * \{tp\} \subseteq R\,;\{tp\}\,;R$ is valid.

Proof. Consider some $tr \in R$. We need to show $\{tr\} * \{tp\} \subseteq R\,;\{tp\}\,;R$. The claim holds vacuously if $tp \cap tr \neq \emptyset$. Hence assume that $tp \cap tr = \emptyset$ and set

$$tr' =_{df} tr \cap dep(tp)\ , \qquad tr'' =_{df} tr - dep(tp)\ .$$

In particular, $tp \cap tr' = \emptyset$. From Lemma 6.3 of [9] we know

$$tr'' \cap dep(tp) \;=\; tr'' \cap dep(tr') \;=\; \emptyset \;.$$

If we can show that also $tp \cap dep(tr') = \emptyset$ we have $\{tr\} * \{tp\} = \{tr'\}\,;\{tp\}\,;\{tr''\}$ and are done. Therefore, suppose $p \in tp \cap dep(tr')$, say $p \to^{+} r$ for some $r \in tr'$. By definition of tr' there is a $q \in tp$ with $r \to^{+} q$. Since tp is assumed to be convex, this implies $r \in tp$, a contradiction to $r \in tr'$ and $tp \cap tr' = \emptyset$. $\square$

Next, we consider general programs.

Definition 6.9 A program is *convex* if all its traces are.

Lemma 6.10 *P is convex iff it satisfies for all power invariants R*

$$R * P \subseteq R\,;P\,;R \;.$$

Proof. ($\Rightarrow$) Immediate from the definition and Lemma 6.8.
($\Leftarrow$) Consider traces $tp \in P$ and $tr \in R$. We need to show $\{tr\}*\{tp\} \subseteq R;\{tp\};R$. The claim holds vacuously if $tp \cap tr \neq \emptyset$. Hence let $tp \cap tr = \emptyset$. By the assumption there are traces $tp' \in P$ and $tr', tr'' \in tr$ with $tp' \cap tr' = tp' \cap tr'' = tr' \cap tr'' = \emptyset$ and $tr' \not\leftarrow tp' \;\wedge\; tp' \not\leftarrow tr'' \;\wedge\; tr' \not\leftarrow tr''$ such that $tp \cup tr = tr' \cup tp' \cup tr''$. But by disjointness this implies $tp' = tp$ and we are done. $\square$

These results motivate the following abstraction.

Definition 6.11 An element a of a CKA with invariants is called *convex* iff for all invariants r we have $r * a \leq r\,;a\,;r$.

By $b\,;c \leq b * c$, commutativity of $*$ and idempotence of invariants (Theorem 5.6) this inequation strengthens to an equality. This means that convex elements behave like "atoms" w.r.t. sequentialisation. Convexity will be important for one of the rules presented in the next section.

7 A Simplified Rely/Guarantee-Calculus

Jones quintuples [12] can be defined, as in [10], by

$$a \; r \,\{b\}\, s \, g \quad \Leftrightarrow_{df} \quad a \,\{r * b\}\, s \;\;\wedge\;\; b \;\mathsf{guar}\; g \;.$$

when r and g are invariants. They are based on the following Hoare triples:

$$c \,\{d\}\, e \;\Leftrightarrow_{df}\; c\,;d \leq e \;.$$

In [9] it is shown that all the standard rules for Hoare triples also hold for this abstract version.

However, in the setting of the present paper the following type of quadruples with an invariant r works just as well:

$$a \; r \,\{b\}\, s \;\Leftrightarrow_{df}\; a \,\{r * b\}\, s \;.$$

If information about the events of a program b is needed (the rôle of g in the original quintuples of the Jones calculus is, to a certain extent, to carry this information), one can use the smallest invariant $\iota\, b$ containing b, since b guar $\iota\, b$.

We give the simplified versions of the original rely/guarantee-properties; the proofs result in a straightforward way from the ones shown in [9] by omitting the guarantee parts.

For parallel composition we obtain

Theorem 7.1 *For invariants* r, r',

$$a\ r\,\{b\}\,s \ \wedge\ a'\ r'\,\{b'\}\,s' \ \wedge\ b' \le r \ \wedge b \le r' \ \Rightarrow \\ (a \sqcap a')\ (r \sqcap r')\,\{b * b'\}\,(s \sqcap s')\ .$$

For sequential composition one has

Theorem 7.2 *For invariants* r, r',

$$a\ r\,\{b\}\,s \wedge s\ r'\,\{b'\}\,s' \Rightarrow a\ (r \sqcap r')\,\{b\,;b'\}\,s'$$

provided $b\,;b'$ *is protected from* $r \sqcap r'$, *i.e.,*

$$(r \sqcap r') * (b\,;b') \le (r * b)\,;(r' * b')\ .$$

The protectedness assumption holds in particular if the underlying CKA is $*$-distributive, since $r \sqcap r'$ is again an invariant and hence

$$(r \sqcap r') * (b\,;b') \le ((r \sqcap r') * b)\,;(r \sqcap r') * b') \le (r * b)\,;(r' * b')\ .$$

Next we give rules for 1, union and convex programs.

Theorem 7.3

1. $a\ r\,\{1\}\,s \ \Leftrightarrow\ a\,\{r\}\,s$.
2. $a\ r\,\{b + b'\}\,s \ \Leftrightarrow\ a\ r\,\{b\}\,s \wedge a\ r\,\{b'\}\,s$.
3. *If* b *is convex then* $a\ r\,\{b\}\,s \ \Leftrightarrow\ a\,\{r\,;b\,;r\}\,s$.

Part 3 has only been given for concrete single-event programs in [9]; therefore we give a quick proof for the abstract form here:

$$a\ r\,\{b\}\,s \Leftrightarrow a\,;(r * b) \subseteq s \Leftrightarrow a\,;(r\,;b\,;r) \subseteq s \Leftrightarrow a\,\{r\,;b\,;r\}\,s\ . \qquad \square$$

8 Event-Based Algebras

The definition of a CKA does not mention the dependence relation anymore. However, in the next section, when we establish a sufficient condition for protectedness, we shall need it, even at the level of single events. Therefore we now give algebraic characterisations of traces and events.

Throughout this section we assume a CKA S with $1 \ne 0$. We denote the supremum of a subset $T \subseteq S$ by ΣT.

Definition 8.1 An element $t \in S$ is called a *trace* if it is subatomic and join-prime, i.e., if

$$\forall a \in S : a \le t \Rightarrow a = 0 \vee a = t\ , \\ \forall T \subseteq S : T \ne \emptyset \wedge t \le \Sigma T \Rightarrow \exists a \in T : t \le a\ .$$

The set of all traces is denoted by $TR(S)$. For b in S the set of traces of b is

$$TR(b) =_{df} \{a \in TR(S)a^* \le b\}\ .$$

By this definition 0 is a trace. The traces different from 0 would be called atoms in lattice theory (e.g. [3]). Admitting also 0 as a trace saves a number of case distinctions. It is immediate that every trace a is $+$-irreducible, i.e.,

$$a = b + c \Rightarrow b = a \vee c = a\ .$$

Moreover, if a is a trace and $b \le a$ then b is a trace again. In particular, if $a * b$ is a trace then by Lemma 4.6(3) also $a\,;b$ is a trace.

In our concrete model the abstract traces different from 0 correspond to singleton programs.

Definition 8.2 In a CKA S we define a relation $\sqsubseteq$ by

$$a \sqsubseteq b \Leftrightarrow_{df} \exists\, c : b = a * c \ .$$

To investigate its properties we need

Definition 8.3 A subset $E \subseteq S$ is *well behaved* if the following conditions hold (for $a, b, c \in E$):

(a) $1 \in E$.
(b) $E * E \subseteq E$.
(c) $*$ is cancellative on E, i.e., $a * b \neq 0 \wedge a * b = a * c \;\Rightarrow\; b = c$.
(d) 1 is $*$-irreducible in E, i.e., $1 = a * b \Rightarrow a = 1 \vee b = 1$.

Lemma 8.4

1. $\sqsubseteq$ is a preorder, i.e., reflexive and transitive.

Assume now that $E \subseteq S$ is well behaved. Then we have the following additional properties.

2. $\sqsubseteq$ is antisymmetric on E.
3. 1 is the $\sqsubseteq$-least element of E.
4. If $0 \in E$ then it is the $\sqsubseteq$-greatest element of E.

Proof. 1. Reflexivity follows by choosing $c = 1$ in the definition of $\sqsubseteq$.
For transitivity assume $a \sqsubseteq b$ and $b \sqsubseteq c$, say $b = a * d$ and $c = b * e$. Then $c = (a * d) * e = a * (d * e)$.
2. Assume $a \sqsubseteq b$ and $b \sqsubseteq a$. If $a = 0$ then $b = 0$ follows form the definition of $a \sqsubseteq b$, since 0 is an annihilator for $*$. Otherwise let $b = a * c$ and $a = b * d$. Then $a * 1 = a = b * d = a * c * d$, hence $1 = c * d$ by cancellativity. Now, irreducibility of 1 implies $c = 1 \vee d = 1$ and hence $c = 1 = d$, showing $a = b$.
3. and (4) are straightforward from the definition of $\sqsubseteq$, neutrality of 1 and annihilation of 0. □

In our concrete model, the set E of singleton programs is well behaved and $\sqsubseteq$ is isomorphic to the subset relation on concrete traces.

Assume now that E is well behaved and hence $\sqsubseteq$ is a partial order on E. The supremum of a subset $D \subseteq E$ w.r.t. $\sqsubseteq$, if existent, is denoted by $\mathit{\Pi}\, D$.

Lemma 8.5 *If $0 \in D \subseteq E$ then $0 = \mathit{\Pi}\, D$.*

This immediate from the definition of $\sqsubseteq$ and suprema.

Definition 8.6 Assume that E is well behaved. Then $e \in E$ is called an *E-event* if it is subatomic and join-prime w.r.t. $\sqsubseteq$, i.e., if

$$\forall\, d \in E : d \sqsubseteq e \;\Rightarrow\; d = 1 \,\vee\, d = e \ ,$$
$$\forall\, D \subseteq E : D \neq \emptyset \,\wedge\, \mathit{\Pi}\, D \text{ exists} \;\Rightarrow\; (t \sqsubseteq \mathit{\Pi}\, D \;\Rightarrow\; \exists\, d \in D : t \sqsubseteq d) \ .$$

By this definition 1 is an E-event, as is 0 if $0 \in E$. The E-events different from $0, 1$ are atoms w.r.t. $\sqsubseteq$ in E. Clearly, every E-event a is $*$-irreducible in E:

$$a = b * c \;\Rightarrow\; b = a \,\vee\, c = a \ .$$

To put things into perspective, we note that the order $\sqsubseteq$ corresponds to the well-known divisibility order on the natural numbers and E-events play the same rôle as the prime numbers.

Definition 8.7 A CKA S is *event-based* if the following properties hold:

(a) 1 is a trace.
(b) Every element is the supremum of its traces, i.e., for all $a \in S$ we have $a = \mathit{\Sigma}\; \mathit{TR}(a)$.

(c) The set $TR(S)$ of traces is well behaved. By $EV(S)$ we denote the set of $TR(S)$-events and call them the *events* of S. The set of events of trace t is

$$EV(t) =_{df} \{e \in EV(S)e^* \leq t\} \ .$$

(d) The set $TR(S)$ of traces is a complete lattice w.r.t. $\sqsubseteq$ and every trace is the supremum of its events, i.e., for all $t \in TR(S)$ we have $t = \mathit{\Pi}\, EV(t)$.
(e) For all events e we have $e * e = 0$ and hence $e \,;\, e = 0$.

For an arbitrary $a \in S$ we then set $EV(a) =_{df} \bigcup_{t \in TR(a)} EV(t)$.

Hence our concrete model of programs forms an event-based CKA. Event-based CKAs are quite similar to the feature algebras developed in [11] for the description of product families.

The definition of an event-based CKA S immediately yields

Lemma 8.8

1. *$EV(0) = EV(S)$.*
2. *$EV(1) = \{1\}$.*
3. *For traces a, b with $a * b \neq 0$ we have $EV(a * b) = EV(a) \cup EV(b)$ and hence $a * b = \mathit{\Pi}\, \{a, b\}$.*

8.1 Abstract Dependence and Protection

In this section we define an abstract counterpart to the dependence relation and use it to give an intuitive sufficient criterion for protectedness. For it we need an abstract formulation of the dependence relation.

Definition 8.9 We call element a *sequentially independent* of element b, in signs $a \not\leftarrow b$, if $a * b \leq a \,;\, b$.

The following properties are shown by straightforward calculation and, in the last case, by Theorem 5.6:

Lemma 8.10

1. *$0 \not\leftarrow a$ and $a \not\leftarrow 0$.*
2. *$1 \not\leftarrow a$ and $a \not\leftarrow 1$.*
3. *$a \not\leftarrow c \wedge b \not\leftarrow c \Rightarrow (a + b) \not\leftarrow c$.*
4. *$a \not\leftarrow b \wedge a \not\leftarrow c \Rightarrow a \not\leftarrow (b + c)$.*
5. *If r is an invariant then $r \not\leftarrow r$.*

Part 5 shows that for general programs this notion behaves in an unexpected way. However, in our concrete model it works fine for singleton programs:

$$\{tp\} \not\leftarrow \{tq\} \ \Leftrightarrow\ \forall\, p \in tp, q \in tq : \neg(p \leftarrow q) \ .$$

In particular, $[p] \not\leftarrow [q] \Leftrightarrow \neg(p \leftarrow q)$. This motivates the following

Definition 8.11 In an event-based CKA we define the dependence relation between events e, f by

$$e \rightarrow f \ \Leftrightarrow_{df}\ \neg(f \not\leftarrow e) \ \Leftrightarrow\ f \,;\, e \neq f * e \ .$$

Again, we denote the converse of $\rightarrow$ by $\leftarrow$. We say that the algebra *respects dependence* if $e \leftarrow f \ \Rightarrow\ e \,;\, f = 0$.

Lemma 8.12 *Consider traces tp, tq of an event-based CKA that respects dependence.*

1. *If $p \rightarrow q$ for some $p \in EV(tp)$ and $q \in EV(tq)$ then $tp \,;\, tq = 0$.*
2. *If $tp * tq \neq 0$ then*

$$tp \not\leftarrow tq \ \Leftrightarrow\ \forall\, p \in EV(tp), q \in EV(tq) : p \not\leftarrow q \ .$$

Proof. 1. By additivity of ; we have $tp\,;tq = \Pi\,\{u\,;vu^* \in EV(tp), v \in EV(tq)\}$ and the claim follows from Lemma 8.5.

2. ($\Leftarrow$) Immediate from event-basedness and additivity of $*$ and ; .
($\Rightarrow$) By Part 1 we have $p\,;q \neq 0$ for all $p \in EV(tp)$ and $q \in EV(tq)$. Since $TR(S)$ is assumed to be well behaved, also $p * q$ is a trace, and from $p\,;q \leq p * q$ it follows that $p\,;q = p * q$. □

With these prerequisites it is now possible to completely replay the proof of Theorem 6.1 in the abstract setting of event-based CKAs; we omit the details.

References

1. R. Back, J. von Wright: Refinement calculus — a systematic introduction. Springer 1998
2. J. Benabou: Introduction to bicategories. In: Reports of the Midwest Category Seminar. Lecture Notes in Math. 47. Springer 1967, 1-77
3. Birkhoff, G. Lattice Theory, 3rd ed. Amer. Math. Soc. 1967
4. E. Cohen: Separation and reduction. In: R. Backhouse, J. Oliveira (eds.): Mathematics of Program Construction (MPC'00). LNCS 1837. Springer 2000, 45–59
5. J. Conway: Regular Algebra and Finite Machines. Chapman&Hall 1971
6. J. Desharnais, B. Möller, G. Struth: Kleene Algebra with domain. Trans. Computational Logic 7, 798–833 (2006)
7. M. Erné, J. Koslowski, A. Melton, G. E. Strecker: A primer on Galois connections. In: Proc. 1991 Summer Conference on General Topology and Applications in Honor of Mary Ellen Rudin and Her Work. Annals of the New York Academy of Sciences 704, 103–125 (1993)
8. C.A.R. Hoare: An axiomatic basis for computer programming. Commun. ACM. 12, 576–585 (1969)
9. C.A.R. Hoare, B. Möller, G. Struth, I. Wehrman: Concurrent Kleene Algebra.Institut für Informatik, Universität Augsburg, Technical Report 2009-04, April 2009
10. C.A.R. Hoare, I. Wehrman, P. O'Hearn: Graphical models of separation logic. Proc. Marktoberdorf Summer School 2008 (forthcoming)
11. P. Höfner, R. Khedri, B. Möller: Feature algebra. In J. Misra, T. Nipkow, E. Sekerinski (eds): Formal Methods (FM 2006). LNCS 4085. Springer 2006, 300–315
12. C. Jones: Development methods for computer programs including a notion of interference. PhD Thesis, University of Oxford. Programming Research Group, Technical Monograph 25, 1981
13. D. Kozen: A completeness theorem for Kleene algebras and the algebra of regular events. Information and Computation 110, 366–390 (1994)
14. D. Kozen: Kleene algebra with tests. Trans. Programming Languages and Systems 19, 427–443 (1997)
15. P. Lescanne: Modèles non déterministes de types abstraits. R.A.I.R.O. Informatique théorique 16, 225–244 (1982)
16. S. Mac Lane: Categories for the working mathematician (2nd ed.). Springer 1998
17. W. McCune: Prover9 and Mace4. http://www.prover9.org/ (accessed March 1, 2009)
18. C. Morgan: Programming from Specifications. Prentice Hall 1990
19. C. Mulvey: &. Rendiconti del Circolo Matematico di Palermo 12, 99–104 (1986)
20. P. O'Hearn: Resources, concurrency, and local reasoning. Theor. Comput. Sci. 375, 271–307 (2007)
21. D. Sangiorgi, D. Walker: The π-calculus — A theory of mobile processes. Cambridge University Press 2001

A Axiom Systems

For ease of reference we summarise the algebraic structures employed in the paper.

1. A *semiring* is a structure $(S, +, 0, \cdot, 1)$ such that $(S, +, 0)$ is a commutative monoid, $(S, \cdot, 1)$ is a monoid, multiplication distributes over addition in both arguments and 0 is a left and right annihilator with respect to multiplication ($a \cdot 0 = 0 = 0 \cdot a$). A semiring is *idempotent* if its addition is.
2. A *quantale* [19] or *standard Kleene algebra* [5] is an idempotent semiring that is a complete lattice under the natural order and in which composition distributes over arbitrary suprema. The infimum and the supremum of a subset T are denoted by $\sqcap T$ and $\sqcup T$, respectively. Their binary variants are $x \sqcap y$ and $x \sqcup y$ (the latter coinciding with $x + y$).
3. A *concurrent Kleene algebra* (CKA) is a structure $(S,+, 0,*, ;, 1)$ such that $(S, +, 0, *, 1)$ and $(S, +, 0, ;, 1)$ are quantales linked by the exchange axiom
$$(a * b) ; (c * d) \leq (b ; c) * (a ; d) \ .$$
4. A *CKA with invariants* $(S, +, 0, *, ;, 1, \iota)$ consists of a CKA $(S, +, 0, *, ;, 1)$ and a closure operator $\iota : S \to S$ that additionally satisfies, for all $a, b \in S$,
$$1 \leq \iota\, a \ , \qquad\qquad \iota\,(a * b) \leq \iota\,(a + b) \ .$$
An *invariant* is an element $a \in S$ with $\iota\, a = a$.
5. A *rely/guarantee-CKA* [9] is a pair (S, I) such that S is a CKA and $I \subseteq I(S)$ is a set of invariants, i.e. of elements r satisfying $r = r^*$, such that $1 \in I$ and for all $r, r' \in I$ also $r \sqcap r' \in I$ and $r * r' \in I$. Moreover, all $r \in I$ and $a, b \in S$ have to satisfy
$$r * (a ; b) \leq (r * a) ; (r * b) \ .$$

B Sample Proof Script

As a sample input file for Prover9 we show one for proving the sequential composition rule from from Sec. 7. One sees that the axioms and the proof goals can be stated almost in the same syntax as we have used in our definitions. A collection of further input files and proofs can be found under
`http://www.dcs.shef.ac.uk/~georg/ka/`.

```
formulas(assumptions).

% meet
  x^x = x.
  x^y = y^x.
  x^(y^z) = (x^y)^z.

% partial order
  x <= y <-> x = x^y.

% meet split
  z <= x^y <-> z <= x & z <= y.

% isotone commutative semigroup
  (x^y)*z <= x*z.
  x*(y^z) <= x*z.
  x*y = y*x.
```

```
  x*(y*z) = (x*y)*z.

% isotone semigroup
  (x^y);z <= x;z.
  x;(y^z) <= x;z.
  x;(y;z) = (x;y);z.

% Jones quadruple
  all p all q all r all s (quad(p,r,q,s) <-> p;(r*q) <= s).

% protection
 all q1 all q2 all r (protected(q1,q2,r) <->
                    r * (q1 ; q2) <= (r * q1) ; (r * q2)).

end_of_list.

formulas(goals).

% sequential composition rule
 all p1 all q1 all r1 all s1 all p2 all q2 all r2 all s2
 (quad(p1,r1,q1,s1) & quad(s1,r2,q2,s2) & protected(q1,q2,r1^r2) ->
    quad(p1,(r1^r2),(q1 ; q2),s2)).

end_of_list.
```