Concurrent Kleene Algebra

C. A. R. Tony Hoare, Bernhard Möller, Georg Struth, Ian Wehrman

Angaben zur Veröffentlichung / Publication details:

Hoare, C. A. R. Tony, Bernhard Möller, Georg Struth, and Ian Wehrman. 2009. "Concurrent Kleene Algebra." Lecture Notes in Computer Science 5710: 399–414. https://doi.org/10.1007/978-3-642-04081-8_27.

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, next to choice and iteration, 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. We show the applicability of the algebra to a partially-ordered trace model of program execution semantics and demonstrate its usefulness by validating familiar proof rules for sequential programs (Hoare triples) and for concurrent programming (Jones's rely/guarantee calculus). The latter involves an algebraic notion of invariants; for these the exchange inequation strengthens to an equational distributivity law. Most of our reasoning has been checked by computer.

1 Introduction

Kleene algebra [4] has been recognised and developed [10, 11, 5] as an algebraic framework (or structural equivalence) that unifies diverse theories for conventional sequential programming. Its many familiar models include relations under relational composition, sequences under concatenation, and sequences under interleaving. This paper defines a 'double' Kleene algebra, with two composition operators, modelling sequential and concurrent composition of programs. They are related by an inequational weakening of the equational exchange law $(a \circ b) \bullet (c \circ d) = (a \bullet c) \circ (b \bullet d)$ of two-category or bicategory theory (e.g. [12]). Under certain conditions, this is strengthened to an equational law, by which concurrent composition distributes through sequential. The axioms proposed for a concurrent Kleene algebra are catalogued in Section 4.

The interest of concurrent Kleene algebra (CKA) is two-fold. Firstly, it expresses only the essential properties of program execution; indeed, it represents just those properties which are preserved even by architectures with weakly-ordered memory access, unreliable communications and massively re-ordering program optimisers. Secondly, the modelled properties, though unusually weak, are strong enough to validate the main structural laws of assertional reasoning about program correctness, both in sequential style [6] (as described in Section 5) and in concurrent style [9] (as described in Section 8).

The purpose of the paper is to introduce the basic operations and their laws, both in a concrete representation and in abstract, axiomatic form. We hope in future research to relate CKA to various familiar process algebras, such as the π-calculus or CSP, and to clarify the links between their many variants.

Before we turn to the abstract treatment, Section 2 introduces our weak semantic model which also is a concrete model of the notion of a CKA. A program is identified with the traces of all the executions it may evoke. Each trace consists of the set of events that occur during a single execution. When two subprograms are combined, say in a sequential or a concurrent combination, each event that occurs is an event in the trace of exactly one of the subprograms. Each trace of the combination is therefore the disjoint union of a trace of one of the sub-programs with a trace of the other. Our formal definitions of the program combinators identify them as a kind of separating conjunction [16].

We introduce a primitive dependence relation between the events of a trace. Its transitive closure represents a direct or indirect chain of dependence. In a sequential composition, it is obviously not allowed for an event occurring in execution of the first operand to depend (directly or indirectly) on an event occurring in execution of the second operand. We take this as our definition of a very weak form of sequential composition. Concurrent composition places no such restriction, and allows dependence in either direction. The above-mentioned exchange law seems to generally capture the interrelation between sequential and concurrent composition in adequate inequational form.

The dependence primitive is intended to model a wide range of computational phenomena, including control dependence (arising from program structure) and data dependence (arising from flow of data). There are many forms of data flow. Flow of data across time is usually mediated by computer memory, which may be private or shared, strongly or only weakly consistent. Flow of data across space is usually mediated by a real or simulated communication channel, which may be buffered or synchronised, double-ended or multiplexed, reliable or lossy, and perhaps subject to stuttering or even re-ordering of messages.

Obviously, it is only weak properties of a program that can be proved without knowing more of the properties of the memory and communication channels involved. The additional properties are conveniently specified by additional axioms, like those used by hardware architects to describe specific weak memory models (e.g. [13]). Fortunately, the addition of further axioms does not invalidate any of our fundamental theory, and they do not require fresh proofs of any of our theorems.

In this paper we focus on the basic concrete CKA model and the essential laws; further technical details can be found in the companion paper [7]. Appendix A summarises the laws characterising the various structures involved. The proofs of the various properties can be found in Appendix B, where we also show a typical input file for the automated theorem prover Prover9 [17] with which all the algebraic proofs have been reconstructed automatically.

2 Operators on Traces and Programs

We assume a set EV of *events*, i.e., occurrences of primitive actions, 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.

A *trace* is a set of events and a *program* is a set of traces. For example, the program skip, which does nothing, is defined as $\{\emptyset\}$, and the program $[e]$, which does only e, is $\{\{e\}\}$. The program false $=_{df}\ \emptyset$ has no traces, and therefore cannot be executed at all. It serves the rôle of the 'miracle' [14] in the development of programs by stepwise refinement. We have false $\subseteq P$ for all programs P.

Following [8] 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, whenever possible.

For the formal definition let $\rightarrow^+$ be the transitive closure of the dependence relation $\rightarrow$ and set, for trace tp, $dep(tp)\ =_{df}\ \{q \mid \exists p \in tp : q \rightarrow^+ p\}$. Thus, $dep(tp)$ is the set of events on which some event in tp depends. Therefore, trace tp is independent of trace tq iff $dep(tp) \cap tq = \emptyset$. The use of the transitive closure $\rightarrow^+$ seems intuitively reasonable; an algebraic justification will be discussed in Section 7.

Definition 2.1 Consider the schematic combination function

$$\mathrm{COMB}(P, Q, C)\ =_{df}\ \{tp \cup tq \mid tp \in P\ \wedge\ tq \in Q\ \wedge\ tp \cap tq = \emptyset\ \wedge\ C(tp, tq)\}$$

with programs P, Q and a predicate C in the trace variables tp and tq. Then the above operators are given by

$$\begin{array}{lll} P * Q & =_{df} & \mathrm{COMB}(P, Q, \mathsf{TRUE})\ , \\ P\ ;\ Q & =_{df} & \mathrm{COMB}(P, Q,\ dep(tp) \cap tq = \emptyset)\ , \\ P \parallel Q & =_{df} & \mathrm{COMB}(P, Q,\ dep(tp) \cap tq = \emptyset\ \wedge\ dep(tq) \cap tp = \emptyset)\ , \\ P \,[]\, Q & =_{df} & \mathrm{COMB}(P, Q,\ tp = \emptyset\ \vee\ tq = \emptyset)\ . \end{array}$$

Example 2.2 We illustrate the operators with a mini-example. We assume a set EV of events the actions of which are simple assignments to program variables. We consider three particular events ax, ay, az associated with the assignments $x := x + 1, y := y + 2, z := x + 3$, resp. There is a dependence arrow from event e to event f iff $e \neq f$ and the variable assigned to in e occurs in the assigned expression in f. This means that for our three events we have exactly $ax \rightarrow az$. We form the corresponding single-event programs $P_x\ =_{df}\ [ax], P_y\ =_{df}\ [ay], P_z\ =_{df}\ [az]$. To describe their compositions we extend the notation for single-event programs and set $[e_1, \ldots, e_n]\ =_{df}\ \{\{e_1, \ldots, e_n\}\}$ (for uniformity we sometimes also write $[\,]$ for skip). Figure 1 lists the composition tables for our operators on these programs. They show that the operator $*$ allows forming parallel programs with race conditions, whereas ; and $\parallel$ respect dependences. □

It is straightforward from the definitions that $*, \parallel$ and $[]$ are commutative and that $[] \subseteq \parallel \subseteq\ ; \subseteq *$ where where for $\circ, \bullet \in \{*, ;, \parallel, []\}$ the formula $\circ \subseteq \bullet$ abbreviates $\forall P, Q : P \circ Q \subseteq P \bullet Q$. Further useful laws are the following.

$*$	P_x	P_y	P_z
P_x	$\emptyset$	$[ax, ay]$	$[ax, az]$
P_y	$[ax, ay]$	$\emptyset$	$[ay, az]$
P_z	$[ax, az]$	$[ay, az]$	$\emptyset$

$;$	P_x	P_y	P_z
P_x	$\emptyset$	$[ax, ay]$	$[ax, az]$
P_y	$[ax, ay]$	$\emptyset$	$[ay, az]$
P_z	$\emptyset$	$[ay, az]$	$\emptyset$

$\|\|$	P_x	P_y	P_z
P_x	$\emptyset$	$[ax, ay]$	$\emptyset$
P_y	$[ax, ay]$	$\emptyset$	$[ay, az]$
P_z	$\emptyset$	$[ay, az]$	$\emptyset$

$[]$	P_x	P_y	P_z
P_x	$\emptyset$	$\emptyset$	$\emptyset$
P_y	$\emptyset$	$\emptyset$	$\emptyset$
P_z	$\emptyset$	$\emptyset$	$\emptyset$

Fig. 1. Composition tables

Lemma 2.3 *Let* $\circ, \bullet \in \{*, ;, \|, []\}$.

1. $\circ$ *distributes through arbitrary unions; in particular,* false *is an annihilator for* $\circ$*, i.e.,* $\mathsf{false} \circ P = \mathsf{false} = P \circ \mathsf{false}$*. Moreover,* $\circ$ *is isotone w.r.t.* $\subseteq$ *in both arguments.*
2. skip *is a neutral element for* $\circ$*, i.e.,* $\mathsf{skip} \circ P = P = P \circ \mathsf{skip}$.
3. *If* $\bullet \subseteq \circ$ *and* $\circ$ *is commutative then* $(P \circ Q) \bullet (R \circ S) \subseteq (P \bullet R) \circ (Q \bullet S)$.
4. *If* $\bullet \subseteq \circ$ *then* $P \circ (Q \bullet R) \subseteq (P \circ Q) \bullet R$.
5. *If* $\circ \subseteq \bullet$ *then* $(P \circ Q) \bullet R \subseteq P \circ (Q \bullet R)$.
6. $\circ$ *is associative.*

The proofs either can be done by an easy adaptation of the corresponding ones in [8] or follow from more general results in [7]. A particularly important special case of Part 3 is the exchange law

$$(P * Q) ; (R * S) \subseteq (P ; R) * (Q ; S) \tag{1}$$

In the remainder of this paper we shall mostly concentrate on the more interesting operators $*$ and $;$.

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

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

3 Quantales, Kleene and Omega Algebras

We now abstract from the concrete case of programs and embed our model into a more general algebraic setting.

Definition 3.1 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.

The operation + denotes an abstract form of nondeterministic choice; in the concrete case of programs it will denote union (of sets of traces). This explains why + is required to be associative, commutative and idempotent. Its neutral element 0 will take the rôle of the miraculous program $\emptyset$.

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 3.2 A *quantale* [15] or *standard Kleene algebra* [4] 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 $x + y$).

In particular, quantale composition is *continuous*, i.e., distributes through suprema of arbitrary, not just countable, chains. As an idempotent semiring, every quantale has 0 as its least element. As a complete lattice, it also has a greatest element $\top$.

Let now $PR(EV) =_{df} \mathcal{P}(\mathcal{P}(EV))$ denote the set of all programs over the event set EV. From the observations in Section 2 the following is immediate:

Lemma 3.3 $(PR(EV), \cup, \mathsf{false}, *, \mathsf{skip})$ *and* $(PR(EV), \cup, \mathsf{false}, ;, \mathsf{skip})$ *are quantales. In each of them* $\top = \mathcal{P}(EV)$ *is the most general program over* EV*.*

Definition 3.4 In a quantale S, the finite and infinite iterations a^* and a^ω of an element $a \in S$ are defined by $a^* = \mu x \,.\, 1 + a \cdot x$ and $a^\omega = \nu x \,.\, a \cdot x$, where μ and ν denote the least and greatest fixpoint operators.

The star used here should not 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 $(S, +, 0, \cdot, 1, ^*)$ forms a Kleene algebra [10]. From this we obtain many useful laws for free. As examples we mention

$$1 \leq a^* \,, \quad a \leq a^* \,, \quad a^* \cdot a^* = (a^*)^* = a^* \,, \quad (a + b)^* = a^* \cdot (b \cdot a^*)^* \,. \quad \text{(KA)}$$

The finite non-empty iteration of a is defined as $a^+ =_{df} a \cdot a^* = a^* \cdot a$. Again, the plus in a^+ should not be confused with the plus of semiring addition.

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

$$b + a \cdot x \leq x \Rightarrow a^* \cdot b \leq x \,, \qquad b + x \cdot a \leq x \Rightarrow b \cdot a^* \leq x \,. \qquad (2)$$

We now study the behaviour of iteration in our program quantales. For a program P, the program P^* taken in the quantale $(PR(EV), \cup, \mathsf{false}, ;, \mathsf{skip})$ consists of all sequential compositions of finitely many traces in P; it is denoted

by P^∞ in [8]. The program P^* taken in $(PR(EV), \cup, \mathsf{false}, *, \mathsf{skip})$ consists of all disjoint unions of finitely many traces in P; it may be considered as describing all finite parallel spawnings of traces in P.

Example 3.5 With the notation of Example 2.2 let $P =_{df} P_x \cup P_y \cup P_z$. We first look at its powers w.r.t. $*$ composition:

$$P^2 = P * P = [ax, ay] \cup [ax, az] \cup [ay, az] \ ,$$
$$P^3 = P * P * P = [ax, ay, az] \ .$$

Hence P^2 and P^3 consist of all programs with exactly two and three events from $\{ax, ay, az\}$, respectively. Since none of the traces in P is disjoint from the one in P^3, we have $P^4 = P^3 * P = \emptyset$, and hence strictness of $*$ w.r.t. $\emptyset$ implies $P^n = \emptyset$ for all $n \geq 4$. Therefore P^* consists of all traces with at most three events from $\{ax, ay, az\}$ (the empty trace is there, too, since by definition skip is contained in every program of the form Q^*). It coincides with the set of all possible traces over the three events; this connection will be taken up again in Section 6.

It turns out that for the powers of P w.r.t. the ; operator we obtain exactly the same expressions, since for every program $Q = [e] \cup [f]$ with $e \neq f$ we have

$$Q\,;Q = ([e] \cup [f])\,;([e] \cup [f]) = [e]\,;[e] \cup [e]\,;[f] \cup [f]\,;[e] \cup [f]\,;[f] = [e, f] = Q * Q \ ,$$

provided $e \not\rightarrow^+ f$ or $f \not\rightarrow^+ e$, i.e., provided the trace $[e, f]$ is consistent with the dependence relation. Only in case of a cyclic dependence $e \rightarrow^+ f \rightarrow^+ e$ we have $Q\,;Q = \emptyset$, whereas still $Q * Q = [e, f]$. □

If the complete lattice $(S, \leq)$ in a quantale is completely distributive, i.e., if $+$ distributes over arbitrary infima, then $(S, +, 0, \cdot, 1, ^*, ^\omega)$ forms an omega algebra in the sense of [3]. Again this entails many useful laws, e.g.,

$$1^\omega = \top \ , \qquad (a \cdot b)^\omega = a \cdot (b \cdot a)^\omega \ , \qquad (a + b)^\omega = a^\omega + a^* \cdot b \cdot (a + b)^\omega \ .$$

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 P^ω w.r.t. the composition operator $*$ is similar to the unbounded parallel spawning $!P$ of traces in P in the π-calculus (e.g. [18]).

4 Concurrent Kleene Algebras

The fact that $PR(EV)$ is a double quantale motivates the following abstract definition.

Definition 4.1 By a *concurrent Kleene algebra* (CKA) we mean a structure $(S, +, 0, *, ;, 1)$ such that $(S, +, 0, *, 1)$ and $(S, +, 0, ;, 1)$ are quantales linked by the exchange axiom

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

This implies, in particular, that $*$ and ; are isotone w.r.t. $\leq$ in both arguments.

Compared to the original exchange law (1) 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 4.2 $(PR(EV), \cup, \mathsf{false}, *, ;, \mathsf{skip})$ *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 $a\,;b \leq a * b$ and hence saves two axioms. This is shown by the following

Lemma 4.3 *In a CKA the following laws hold.*

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

The notion of a CKA abstracts completely from traces and events; in the companion paper [7] we show how to retrieve these notions algebraically using the lattice-theoretic concept of atoms.

5 Hoare Triples

In [8] Hoare triples relating programs are defined by $P\,\{Q\}\,R \Leftrightarrow_{df} P\,;Q \subseteq R$. Again, it is beneficial to abstract from the concrete case of programs.

Definition 5.1 An *ordered monoid* is a structure $(S, \leq, \cdot, 1)$ such that $(S, \cdot, 1)$ is a monoid with a partial order $\leq$ and $\cdot$ is isotone in both arguments. In this case we define the *Hoare triple* $a\,\{b\}\,c$ by

$$a\,\{b\}\,c \Leftrightarrow_{df} a \cdot b \leq c\ .$$

This definition entails the following laws:

Lemma 5.2 *Assume an ordered monoid* $(S, \leq, \cdot, 1)$.

1. $a\,\{1\}\,c \Leftrightarrow a \leq c$; *in particular,* $a\,\{1\}\,a \Leftrightarrow \mathsf{TRUE}$. *(skip)*
2. $(\forall\, a, c : a\,\{b\}\,c \Rightarrow a\,\{b'\}\,c) \Leftrightarrow b' \leq b$. *(antitony)*
3. $(\forall\, a, c : a\,\{b\}\,c \Leftrightarrow a\,\{b'\}\,c) \Leftrightarrow b = b'$. *(extensionality)*
4. $a\,\{b \cdot b'\}\,c \Leftrightarrow \exists\, d : a\,\{b\}\,d \wedge d\,\{b'\}\,c$. *(composition)*
5. $a \leq d \wedge d\,\{b\}\,e \wedge e \leq c \Rightarrow a\,\{b\}\,c$. *(weakening)*

If $(S, \cdot, 1)$ *is the multiplicative reduct of an idempotent semiring* $(S, +, 0, \cdot, 1)$ *and the order used in the definition of Hoare triples is the natural semiring order then we have in addition*

6. $a\,\{0\}\,c \Leftrightarrow \mathsf{TRUE}$, *(failure)*
7. $a\,\{b + b'\}\,c \Leftrightarrow a\,\{b\}\,c \wedge a\,\{b'\}\,c$. *(choice)*

If that semiring is a quantale then we have in addition

8. $a \,\{\!|b|\!\}\, a \;\Leftrightarrow\; a \,\{\!|b^+|\!\}\, a \;\Leftrightarrow\; a \,\{\!|b^*|\!\}\, a.$ *(iteration)*

Lemma 5.2 can be expressed more concisely in relational notation. Define for $b \in S$ the relation $\{\!|b|\!\} \subseteq S \times S$ between preconditions a and postconditions c by

$$\forall\, a, c : a \,\{\!|b|\!\}\, c \;\Leftrightarrow_{df}\; a \cdot b \leq c \;.$$

Then the above properties rewrite into

1. $\{\!|1|\!\} \;=\; \leq$.
2. $\{\!|b|\!\} \subseteq \{\!|b'|\!\} \;\Leftrightarrow\; b' \leq b$.
3. $\{\!|b|\!\} = \{\!|b'|\!\} \;\;\Leftrightarrow\;\; b = b'$.
4. $\{\!|b \cdot b'|\!\} = \{\!|b|\!\} \circ \{\!|b'|\!\}$ where $\circ$ means relational composition.
5. $\leq \circ \{\!|b|\!\} \circ \leq \;\subseteq\; \{\!|b|\!\}$.
6. $\{\!|0|\!\} = \mathbb{T}$ where $\mathbb{T}$ is the universal relation.
7. $\{\!|b + b'|\!\} = \{\!|b|\!\} \cap \{\!|b'|\!\}$.
8. $\{\!|b|\!\} \cap I = \{\!|b^+|\!\} \cap I = \{\!|b^*|\!\} \cap I$ where I is the identity relation.

Next we determine the weakest premise ensuring that two composable Hoare triples establish a third one.

Lemma 5.3 *Assume again an ordered monoid $(S, \leq, \cdot, 1)$. Then*

$$(\forall\, a, d, c : a \,\{\!|b|\!\}\, d \;\wedge\; d \,\{\!|b'|\!\}\, c \;\Rightarrow\; a \,\{\!|e|\!\}\, c) \;\;\Leftrightarrow\;\; e \leq b \cdot b' \;.$$

Proof. By straightforward predicate logic the claim is equivalent to the the relational statement $\{\!|b|\!\} \circ \{\!|b'|\!\} \subseteq \{\!|e|\!\} \;\Leftrightarrow\; e \leq b \cdot b'$ which holds by Properties 4 and 2 above. □

Next we present two further rules that are valid in CKAs when the above monoid operation is specialised to sequential composition:

Lemma 5.4 *Let $S = (S, +, 0, *, ;, 1)$ be a CKA and $a, a', b, b', c, c', d \in S$ with $a \,\{\!|b|\!\}\, c$ interpreted as $a \,;\, b \leq c$.*

1. $a \,\{\!|b|\!\}\, c \;\wedge\; a' \,\{\!|b'|\!\}\, c' \;\Rightarrow\; (a * a') \,\{\!|b * b'|\!\}\, (c * c')$. *(concurrency)*
2. $a \,\{\!|b|\!\}\, c \;\Rightarrow\; (d * a) \,\{\!|b|\!\}\, (d * c)$. *(frame rule)*

Let us now interpret these results in our concrete CKA of programs. It may seem surprising that so many of the standard basic laws of Hoare logic should be valid for such a weak semantic model of programs. For example the definition of weak sequential composition allows all standard optimisations by compilers which shift independent commands between the operands of a semicolon. What is worse, weak composition does not require any data to flow from an assignment command to an immediately following read of the assigned variable. The data may flow to a different thread, which assigns a different value to the variable. In fact, weak sequential composition is required for any model of modern architectures, which allow arbitrary race conditions between fine-grain concurrent threads.

The validity of Hoare logic in this weak model is entirely due to a cheat: that we use the same model for our assertions as for our programs. Thus any weakness of the programming model is immediately reflected in the weakness of the assertion language and its logic. In fact, conventional assertions mention the current values of single-valued program variables; and this is not adequate for reasoning about general fine-grain concurrency. To improve precision here, assertions about the history of assigned values would seem to be required.

6 Invariants

We now deal with the set of events a program may use.

Definition 6.1 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 $\mathsf{skip} = \mathcal{P}(\emptyset) = \{\emptyset\}$. The term "invariant" expresses that often a program relies on the assumption that its environment only uses events from a particular subset, i.e., preserves the invariant of staying in that set.

Example 6.2 Consider again the event set EV form Example 2.2. Let V be a certain subset of the variables involved and let E be the set of all events that assign to variables in V. Then the environment Q of a given program P can be constrained to assign at most to the variables in V by requiring $Q \subseteq R$ with the power invariant $R =_{df} \mathcal{P}(E)$. The fact that we want P to be executed only in such environments is expressed by forming the parallel composition $P * R$. □

If E is considered to characterise the events that are admissible in a certain context, a program P can be confined to using only admissible events by requiring $P \subseteq R$ for $R = \mathcal{P}(E)$. In the rely/guarantee calculus of Section 8 invariants will be used to express properties of the environment on which a program wants to rely (whence the name R).

Power invariants satisfy a rich number of useful laws (see [7] for details). The most essential ones for the purposes of the present paper are the following straightforward ones for arbitrary invariant R:

$$\mathsf{skip} \subseteq R \;, \qquad R * R \subseteq R \;. \tag{3}$$

We now again abstract from the concrete case of programs. It turns out that the properties in (3) largely suffice for characterising invariants.

Definition 6.3 An *invariant* in a CKA S is an element $r \in S$ satisfying $1 \leq r$ and $r * r \leq r$. The set of all invariants of S is denoted by $I(S)$.

The two axioms for invariants can be combined into the equivalent formula $1 + r * r \leq r$.

We now first show a number of algebraic properties of invariants that are useful in proving the soundness of the rely/guarantee-calculus in Section 8.

Theorem 6.4 *Assume a CKA S, an $r \in I(S)$ and arbitrary $a, b \in S$.*

1. $a \leq r \circ a$ *and* $a \leq a \circ r$.
2. $r \,;\, r \leq r$.
3. $r * r = r = r \,;\, r$.
4. $r \,;\, (a * b) \leq (r \,;\, a) * (r \,;\, b)$ *and* $(a * b) \,;\, r \leq (a \,;\, r) * (b \,;\, r)$.
5. $r \,;\, a \,;\, r \leq r * a$.
6. $a \in I(S) \Leftrightarrow a = a^*$, *where* * *is taken w.r.t.* $*$ *composition.*
7. *The least invariant comprising* $a \in S$ *is* a^* *where* * *is taken w.r.t.* $*$ *composition.*

Next we discuss the lattice structure of the set $I(S)$ of invariants.

Theorem 6.5 *Assume again a CKA S.*

1. $(I(S), \leq)$ *is a complete lattice with least element* 1 *and greatest element* $\top$.
2. *For* $r, r' \in I(S)$ *we have* $r \leq r' \Leftrightarrow r * r' = r'$. *This means that* $\leq$ *coincides with the natural order induced by the associative, commutative and idempotent operation* $*$ *on* $I(S)$.
3. *For* $r, r' \in I(S)$ *the infimum* $r \sqcap r'$ *in* S *coincides with the infimum of* r *and* r' *in* $I(S)$.
4. $r * r'$ *is the supremum of* r *and* r' *in* $I(S)$. *In particular,* $r \leq r'' \wedge r' \leq r'' \Leftrightarrow r * r' \leq r''$ *and* $r' \sqcap (r * r') = r'$.
5. *Invariants are downward closed:* $r * r' \leq r'' \Rightarrow r \leq r''$.
6. $I(S)$ *is even closed under arbitrary infima, i.e., for a subset* $U \subseteq I(S)$ *the infimum* $\sqcap U$ *taken in* S *coincides with the infimum of* U *in* $I(S)$.

We conclude this section with two laws about iteration.

Lemma 6.6 *Assume a CKA S and let $r \in I(S)$ be an invariant and $a \in S$ be arbitrary. Let the finite iteration * be taken w.r.t. $*$ composition. Then*

1. $(r * a)^* \leq r * a^*$.
2. $r * a^* = r * (r * a)^*$.

7 Single-Event Programs and Rely/Guarantee-CKAs

We will now show that our definitions of $*$ and $;$ for concrete programs in terms of transitive closure of the dependence relation $\rightarrow$ entail two important further laws that are essential for the rely/guarantee calculus to be defined below. In the following theorem they are presented as inclusions; the reverse inclusions already follow from Theorem 6.4.4 for Part 1 and from Lemma 4.3.5, 4.3.4, 4.3.1 and Theorem 6.4.3 for Part 2. Informally, Part 1 means that for acyclic $\rightarrow$ parallel composition of an invariant with a singleton program can be always sequentialised. Part 2 means that for invariants a kind of converse to the exchange law of Lemma 4.3.2 holds.

Theorem 7.1 *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. *For all $P, Q \in PR(EV)$ we have*

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

In the companion paper [7] we define the composition operators ; and $\|$ in terms of $\rightarrow$ rather than $\rightarrow^+$ and show a converse of Theorem 7.1:

- If Part 1 is valid then $\rightarrow$ is weakly acyclic, viz.

$$\forall e, f \in EV : e \rightarrow^+ f \rightarrow^+ e \;\Rightarrow\; e = f .$$

  This means that $\rightarrow$ allows at most immediate self-loops which cannot be "detected" by our definitions of the operators that require disjointness of the operands. It is easy to see that $\rightarrow$ is weakly acyclic iff its reflexive-transitive closure $\rightarrow^*$ is a partial order.
- If Part 2 is valid then $\rightarrow$ is weakly transitive, i.e.,

$$e \rightarrow f \rightarrow g \;\Rightarrow\; e = g \vee e \rightarrow g .$$

This provides the formal justification why in the present paper we right away defined our composition operators in terms of $\rightarrow^+$ rather than just $\rightarrow$.

As before we abstract the above results into general algebraic terms. The terminology stems from the applications in the next section.

Definition 7.2 A *rely/guarantee-CKA* is a pair (S, I) such that S is a CKA and $I \subseteq I(S)$ is a set of invariants such that $1 \in I$ and for all $r, r' \in I$ also $r \sqcap r' \in I$ and $r * r' \in I$, in other words, I is a sublattice of $I(S)$. Moreover, all $r \in I$ and $a, b \in S$ have to satisfy

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

Together with the exchange law in Lemma 4.3.2, $\circ$-idempotence of r and commutativity of $*$ this implies

$$r * (b \circ c) = (r * b) \circ (r * c) \qquad (*\text{-}\textit{distributivity})$$

for all invariants $r \in I$ and operators $\circ \in \{*, ;\}$.

The restriction that $I(S)$ be a sublattice of $I(S)$ is motivated by the rely/guarantee-calculus in Section 8 below.

Using Theorem 7.1 we can prove

Lemma 7.3 *Let $I =_{df} \{\mathcal{P}(E) \mid E \subseteq EV\}$ be the set of all power invariants over EV. Then $(PR(EV), I)$ is a rely-guarantee-algebra.*

Proof. We only need to establish closure of $\mathcal{P}(\mathcal{P}(EV))$ under $*$ and $\cap$. But straightforward calculations show that $\mathcal{P}(E) * \mathcal{P}(F) = \mathcal{P}(E \cup F)$ and $\mathcal{P}(E) \cap \mathcal{P}(F) = \mathcal{P}(E \cap F)$ for $E, F \subseteq EV$. □

We now can explain why it was necessary to introduce the subset I of invariants in a rely/guarantee-CKA. Our proof of $*$-distributivity used downward closure of power invariants. Other invariants in $PR(EV)$ need not be downward closed and hence $*$-distributivity need not hold for them.

Example 7.4 Assume an event set EV with three different events $e, f, g \in EV$ and dependences $e \to g \to f$. Set $P =_{df} [e, f]$. Then $P * P = \emptyset$ and hence $P^i = \emptyset$ for all $i > 1$. This means that the invariant $R =_{df} P^* = \mathsf{skip} \cup P = [] \cup [e, f]$ is not downward closed. Indeed, $*$-distributivity does not hold for it: we have $R * [g] = [g] \cup [e, f, g]$, but $R\,;[g]\,;R = [g]$. □

The property of $*$-distributivity implies further iteration laws.

Lemma 7.5 *Assume a rely/guarantee-CKA (S, I), an invariant $r \in I$ and an arbitrary $a \in S$ and let the finite iteration * be taken w.r.t. $\circ \in \{*, ;\}$.*

1. $r * a^* = (r * a)^* \circ r = r \circ (r * a)^*$.
2. $(r * a)^+ = r * a^+$.

8 Jones's Rely/Guarantee-Calculus

In [9] Jones has presented a calculus that considers properties of the environment on which a program wants to rely and the ones it, in turn, guarantees for the environment. We now provide an abstract algebraic treatment of this calculus.

Definition 8.1 We define, abstracting from [8], the *guarantee relation* by setting for arbitrary element b and invariant g

$$b \;\mathsf{guar}\; g \;\Leftrightarrow_{df}\; b \leq g \;.$$

A slightly more liberal formulation is discussed in [7].

Example 8.2 With the notation $P_u =_{df} [au]$ for $u \in \{x, y, z\}$ of Example 2.2 we have $P_u \;\mathsf{guar}\; G_u$ where $G_u =_{df} P_u \cup \mathsf{skip} = [au] \cup []$. □

We have the following properties.

Theorem 8.3 *Let b, b' be arbitrary elements and g, g' be invariants of a CKA.*

1. $1 \;\mathsf{guar}\; g$.
2. *If* $\circ \in \{*, ;\}$ *then* $b \;\mathsf{guar}\; g \wedge b' \;\mathsf{guar}\; g' \;\Rightarrow\; (b \circ b') \;\mathsf{guar}\; (g * g')$.
3. *For* $\circ \in \{*, ;\}$ *we have* $a \;\mathsf{guar}\; g \Rightarrow a^* \;\mathsf{guar}\; g$.
4. *For the concrete case of programs let $G = \mathcal{P}(E)$ for some set $E \subseteq EV$ and $e \in EV$. Then* $[e] \;\mathsf{guar}\; G \Leftrightarrow e \in E$.

Using the guarantee relation, Jones quintuples can be defined, as in [8], by

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

where r and g are invariants and Hoare triples are again interpreted in terms of sequential composition ; .ed in terms of sequential composition ; .

The first rule of the rely/guarantee calculus concerns parallel composition.

Theorem 8.4 *Consider a CKA S. For invariants $r, r', g, g' \in I(S)$ and arbitrary $a, a', b, b', c, c' \in S$,*

$$a\ r\ \{b\}\ g\ c\ \ \wedge\ \ a'\ r'\ \{b'\}\ g'\ c'\ \ \wedge\ \ g'\ \mathsf{guar}\ r\ \ \wedge\ g\ \mathsf{guar}\ r'\ \ \Rightarrow \\ (a \sqcap a')\ (r \sqcap r')\ \{b * b'\}\ (g * g')\ (c \sqcap c')\ .$$

Note that $r \sqcap r'$ and $g * g'$ are again invariants by Lemma 6.5.3 and 6.5.4. For sequential composition one has

Theorem 8.5 *Assume a rely/guarantee-CKA (S, I). Then for invariants r, r', $g, g' \in I$ and arbitrary a, b, b', c, c',*

$$a\ r\ \{b\}\ g\ c\ \wedge\ c\ r'\ \{b'\}\ g'\ c'\ \Rightarrow\ a\ (r \sqcap r')\ \{b\,;b'\}\ (g * g')\ c'$$

Next we give rules for 1, union and singleton event programs.

Theorem 8.6 *Assume a rely/guarantee-CKA (S, I). Then for invariants $r, g \in I$ and arbitrary $s \in S$,*

1. $a\ r\ \{1\}\ g\ s\ \ \Leftrightarrow\ \ a\ \{r\}\ s$.
2. $a\ r\ \{b + b'\}\ g\ s\ \ \Leftrightarrow\ \ a\ r\ \{b\}\ g\ s\ \wedge\ a\ r\ \{b'\}\ g\ s$.
3. *Assume power invariants $R = \mathcal{P}(E)$, $G = \mathcal{P}(F)$ for $E, F \subseteq EV$, event $e \notin E$ and let $\rightarrow$ be acyclic. Then $P\ R\ \{[e]\}\ G\ S\ \Leftrightarrow\ P\ \{R\,;[e]\,;R\}\ S\ \wedge\ [e]\ \mathsf{guar}\ G$.*

Finally we give rely/guarantee rules for iteration.

Theorem 8.7 *Assume a rely/guarantee-CKA (S, I) and let * be finite iteration w.r.t. $\circ \in \{*, ;\}$. Then for invariants $r, g \in I$ and arbitrary elements $a, b \in S$,*

$$a\ r\ \{b\}\ g\ a\ \Rightarrow\ a\ r\ \{b^+\}\ g\ a\ , \\ a\ \{r\}\ a\ \wedge\ a\ r\ \{b\}\ g\ a\ \Rightarrow\ a\ r\ \{b^*\}\ g\ a\ .$$

We conclude this section with a small example of the use of our rules.

Example 8.8 We consider again the programs $P_u = [au]$ and invariants $G_u = P_u \cup \mathsf{skip}$ $(u \in \{x, y\})$ from Example 8.2. Moreover, we assume an event av with $v \neq x, y$, $ax \not\rightarrow av$ and $ay \not\rightarrow av$ and set $P_v =_{df} [av]$. We will show that the quintuple

$$P_v\ \mathsf{skip}\ \{P_x * P_y\}\ (G_x * G_y)\ [av, ax, ay]$$

holds. In particular, the parallel execution of the assignments $x := x + 1$ and $y := y + 2$ guarantees that at most x and y are changed. We set $R_x =_{df} G_y$ and $R_y =_{df} G_x$. Then

(a) $P_x\ \mathsf{guar}\ G_x\ \mathsf{guar}\ R_y$, (b) $P_y\ \mathsf{guar}\ G_y\ \mathsf{guar}\ R_x$.

Define the postconditions

$$S_x =_{df} [av, ax] \cup [av, ax, ay] \quad \text{and} \quad S_y =_{df} [av, ay] \cup [av, ax, ay]\ .$$

Then

(c) $S_x \cap S_y = [av, ax, ay]$, (d) $R_x \cap R_y = \mathsf{skip}$.

From the definition of Hoare triples we calculate

$$P_v \,\{\!|R_x|\!\}\, ([av] \cup [av, ay]) \qquad ([av] \cup [av, ay])\,\{\!|P_x|\!\}\, S_x \qquad S_x \,\{\!|R_x|\!\}\, S_x \ ,$$

since $[av, ax, ay] * [ay] = \emptyset$. Combining the three clauses by Lemma 5.2.4 we obtain

$$P_v \,\{\!|R_x \,;\, P_x \,;\, R_x|\!\}\, S_x \ .$$

By Theorem 8.6.3 we obtain $P_v \; R_y \,\{\!|P_x|\!\}\, G_x \; S_x$ and, similarly, $P_v \; R_x \,\{\!|P_y|\!\}\, G_y \; S_y$. Now the claim follows from the clauses (a),(b),(c),(d) and Theorem 8.4. □

In a practical application of the theory of Kleene algebras to program correctness, the model of a program trace will be much richer than ours. It will certainly include labels on each event, indicating which atomic command of the program is responsible for execution of the event. It will include labels on each data flow arrow, indicating the value which is 'passed along' the arrow, and the identity of the variable or communication channel which mediated the flow.

9 Conclusion and Outlook

The study in this paper has shown that even with the extremely weak assumptions of our trace model many of the important programming laws can be shown, mostly by very concise and simple algebraic calculations. Indeed, the rôle of the axiomatisation was precisely to facilitate these calculations: rather than verifying the laws laboriously in the concrete trace model, we can do so much more easily in the algebraic setting of Concurrent Kleene Algebras. This way many new properties of the trace model have been shown in the present paper. Hence, although currently we know of no other interesting model of CKA than the trace model, the introduction of that structure has already been very useful.

It is not easy to relate the CKA operators to those of the more familiar process algebras. The closest analogies seem to be the following ones.

CKA operator	corresponding operator
$+$	non-deterministic choice in CSP
$*$	parallel composition $\mid$ in ACP, π-calculus and CCS
$\parallel$	interleaving $\mid\mid\mid$ in CSP
;	sequential composition ; in CSP and $\cdot$ in ACP
$[]$	choice $+$ in CCS and internal choice $\Box$ in CSP
1	`SKIP` in CSP
0	this is the miracle and cannot be represented in any implementable calculus

However, there are a number of laws which show the inaccuracy of this table. For instance, in CSP we have $\texttt{SKIP} \,\Box\, P \neq P$, whereas CKA satisfies $1 \,[]\, P = P$. A similarly different behaviour arises in CCS, ACP and the π-calculus concerning distributivity of composition over choice.

This indicates that CKA is not a direct abstraction of these concurrency calculi. Rather, we envisage that the trace model and its abstraction CKA can serve as a basic setting into which many of the existing other calculi can be mapped so that then their essential laws can be proved using the CKA laws. A first experiment along these lines is a trace model of a core subset of the π-calculus in [8]. An elaboration of these ideas will be the subject of further studies.

Acknowledgement We are grateful for valuable comments by J. Desharnais, H.-H. Dang, R. Glück, W. Guttmann, P. Höfner, P. O'Hearn and H. Yang.

References

1. R. Back, J. von Wright: Refinement calculus — A systematic introduction. Springer 1998
2. Birkhoff, G. Lattice Theory, 3rd ed. Amer. Math. Soc. 1967
3. E. Cohen: Separation and reduction. In: R. Backhouse, J. Oliveira (eds.): Mathematics of Program Construction (MPC'00). LNCS 1837. Springer 2000, 45–59
4. J. Conway: Regular Algebra and Finite Machines. Chapman&Hall 1971
5. J. Desharnais, B. Möller, G. Struth: Kleene Algebra with domain. Trans. Computational Logic 7, 798–833 (2006)
6. C.A.R. Hoare: An axiomatic basis for computer programming. Commun. ACM. 12, 576–585 (1969)
7. C.A.R. Hoare, B. Möller, G. Struth, I. Wehrman: Foundations of Concurrent Kleene Algebra (forthcoming)
8. C.A.R. Hoare, I. Wehrman, P. O'Hearn: Graphical models of separation logic. Proc. Marktoberdorf Summer School 2008 (forthcoming)
9. C. Jones: Development methods for computer programs including a notion of interference. PhD Thesis, University of Oxford. Programming Research Group, Technical Monograph 25, 1981
10. D. Kozen: A completeness theorem for Kleene algebras and the algebra of regular events. Information and Computation 110, 366–390 (1994)
11. D. Kozen: Kleene algebra with tests. Trans. Programming Languages and Systems 19, 427–443 (1997)
12. S. Mac Lane: Categories for the working mathematician (2nd ed.). Springer 1998
13. J. Misra: Axioms for memory access in asynchronous hardware systems. ACM Trans. Program. Lang. Syst. 8, 142–153 (1986)
14. C. Morgan: Programming from Specifications. Prentice Hall 1990
15. C. Mulvey: &. Rendiconti del Circolo Matematico di Palermo 12, 99–104 (1986)
16. P. O'Hearn: Resources, concurrency, and local reasoning. Theor. Comput. Sci. 375, 271–307 (2007)
17. W. McCune: Prover9 and Mace4. `http://www.prover9.org/` (accessed March 1, 2009)
18. 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* [15] or *standard Kleene algebra* [4] 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 *rely/guarantee-CKA* 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 Proofs

We mention two important proof techniques.

First, one has the principle of *indirect inequality*, i.e.,

$$x \leq y \;\Leftrightarrow\; (\forall\, z : y \leq z \;\Rightarrow\; x \leq z) \;\Leftrightarrow\; (\forall\, z : z \leq x \;\Rightarrow\; z \leq y)\ . \tag{4}$$

For the $(\Rightarrow)$ parts use transitivity of $\leq$; for $(\Leftarrow)$ set $z = y$ and $z = x$, resp., and use reflexivity of $\leq$. Hence this principle applies also to preorders, not just to partial orders. One can even show that a relation $\leq$ satisfies this principle iff it is a preorder. For partial orders one obtains, via antisymmetry, as a corollary the principle of *indirect equality*

$$x = y \;\Leftrightarrow\; (\forall\, z : y \leq z \;\Leftrightarrow\; x \leq z) \;\Leftrightarrow\; (\forall\, z : z \leq x \;\Leftrightarrow\; z \leq y)\ . \tag{5}$$

Second, in a quantale we can use the following fusion rules for least fixpoints. Let $f, g, h : S \to S$ be isotone functions and let $\circ$ mean function composition

and $\leq$ denote the pointwise lifting of the order $\leq$ to functions. Then

$$\text{(Right Subfusion)} \qquad \frac{f \circ g \leq g \circ h}{\mu f \leq g(\mu h)} \tag{6}$$

$$\text{(Left Subfusion)} \quad \frac{f \text{ continuous and strict} \quad f \circ g \leq h \circ f}{f(\mu g) \leq \mu h} \tag{7}$$

$$\text{(Fusion)} \quad \frac{f \text{ continuous and strict} \quad f \circ g = h \circ f}{f(\mu g) = \mu h} \tag{8}$$

For instance, right subfusion provides the induction rules in (2). Next, by the fusion rule,

$$a^+ \;=\; \mu x \,.\, a + a \cdot x \;=\; \mu x \,.\, a + x \cdot a \;. \tag{9}$$

Applications of the other fusion rules will be given below.

Proof of Lemma 4.3.

1. By neutrality twice, exchange and neutrality twice again,

$$a * b \;=\; (a * b)\,;(1 * 1) \;\leq\; (b\,;1) * (a\,;1) \;=\; b * a \;.$$

   Since a, b are arbitrary, also the reverse inequality holds and we are done.
2. Immediate from Part 1 and exchange.
3. By neutrality twice, Part 2 and neutrality twice again,

$$a\,;b \;=\; (a * 1)\,;(1 * b) \;\leq\; (a\,;1) * (1\,;b) \;=\; a * b \;.$$

4. By neutrality, Part 2 and neutrality again,

$$(a * b)\,;c \;=\; (a * b)\,;(1 * c) \leq (a\,;1) * (b\,;c) \;=\; a * (b\,;c) \;.$$

5. Symmetric to Part 4. □

Proof of Lemma 5.2.

1. Immediate from the definitions and neutrality of 1.
2. ($\Leftarrow$) follows directly from isotony of composition. For ($\Rightarrow$) we have

$$\begin{array}{ll} & \forall\, a, c : a \,\{b\}\, c \;\Rightarrow\; a \,\{b'\}\, c \\ \Leftrightarrow & \quad \{\![\text{ definitions }]\!\} \\ & \forall\, a, c : a \cdot b \leq c \;\Rightarrow\; a \cdot b' \leq c \\ \Leftrightarrow & \quad \{\![\text{ indirect inequality (4) }]\!\} \\ & \forall\, a : a \cdot b' \leq a \cdot b \\ \Rightarrow & \quad \{\![\text{ choose } a = 1 \,]\!\} \\ & b' \leq b \;. \end{array}$$

3. Immediate from Part 2 and antisymmetry of $\leq$.
4. ($\Leftarrow$) By the definitions, isotony of $\cdot$ and transitivity of $\leq$,

$$a \,\{b\}\, d \wedge d \,\{b'\}\, c \;\Leftrightarrow\; a \cdot b \leq d \wedge d \cdot b' \leq c \;\Rightarrow\; a \cdot b \cdot b' \leq c \;\Leftrightarrow\; a \,\{b \cdot b'\}\, c \;.$$

   ($\Rightarrow$) Choose $d = a \cdot b$.
5. By isotony and the assumptions, $a \cdot b \leq d \cdot b \leq e \leq c$.
6. Immediate from the definitions and the annihilation property of 0.
7. By the definitions. distributivity and standard order theory,

$$\begin{aligned} a \,\{b + b'\}\, c &\Leftrightarrow a \cdot (b + b') \leq c \;\Leftrightarrow\; a \cdot b + a \cdot b' \leq c \\ &\Leftrightarrow a \cdot b \leq c \wedge a \cdot b' \leq c \;\Leftrightarrow\; a \,\{b\}\, c \wedge a \,\{b'\}\, c \;. \end{aligned}$$

8. ($\Rightarrow$) Using the definitions, the second star induction rule in (2) and idempotence of $+$ we have

$$a \,\{b^+\}\, a \;\Leftrightarrow\; a \cdot b \cdot b^* \leq a \;\Leftarrow\; a \cdot b + a \cdot b \leq a \;\Leftrightarrow\; a \cdot b \leq a \;\Leftrightarrow\; a \,\{b\}\, a \;.$$

   The second implication follows from $b^* = 1 + b^+$ and the skip and choice rules.
   ($\Leftarrow$) follows from $b \leq b^+ \leq b^*$ and Part 2. □

Proof of Lemma 5.4.

1. $a \,\{b\}\, c \wedge a' \,\{b'\}\, c'$
   $\Leftarrow$ {[definition]}
   $a \,;\, b \leq c \wedge a' \,;\, b' \leq c'$
   $\Rightarrow$ {[isotony of $*$]}
   $(a \,;\, b) * (a' \,;\, b') \leq c * c'$
   $\Rightarrow$ {[exchange (Lemma 4.3.2)]}
   $(a * a') \,;\, (b * b') \leq c * c'$
   $\Leftrightarrow$ {[definition]}
   $(a * a') \,\{b * b'\}\, (c * c')$
2. $a \,\{b\}\, c$
   $\Leftrightarrow$ {[definition]}
   $a \,;\, b \leq c$
   $\Rightarrow$ {[isotony of $*$]}
   $d * (a \,;\, b) \leq d * c$
   $\Rightarrow$ {[by Lemma 4.3.4]}
   $(d * a) \,;\, b \leq d * c$
   $\Leftrightarrow$ {[definition]}
   $(d * a) \,\{b\}\, (d * c)$.

□

Proof of Theorem 6.4.

1. By neutrality of 1 and isotony of $\circ$ we have $a = 1 \circ a \leq r \circ a$. The second inequation is shown symmetrically.
2. This is immediate from $a \,;\, b \leq a * b$ (Lemma 4.3.3) and transitivity of $\leq$.
3. By Part 1 we have $r \leq r \circ r$; the converse equation holds by definition and Part 2, respectively.
4. $$\begin{array}{ll} & r \,;\, (a * b) \\ \leq & \quad \{\!\![\text{ by Lemma 4.3.5 }]\!\!\} \\ & (r \,;\, a) * b \\ \leq & \quad \{\!\![\text{ by Part 1 and isotony }]\!\!\} \\ & (r \,;\, a) * (r \,;\, b) \;. \end{array}$$
   The second law is proved symmetrically.
5. $$\begin{array}{ll} & r \,;\, a \,;\, r \\ \leq & \quad \{\!\![\text{ by Lemma 4.3.3 }]\!\!\} \\ & r * a * r \\ = & \quad \{\!\![\text{ commutativity of } *]\!\!\} \\ & r * r * a \\ \leq & \quad \{\!\![\text{ by Part 3 }]\!\!\} \\ & r * a \;. \end{array}$$
6. ($\Rightarrow$) By the definition of invariants and Part 2 we have $1 + r * r \leq r$. Hence star induction (2) shows $r^* \leq r$. The converse inequation $r \leq r^*$ holds by (KA).
   ($\Leftarrow$) follows from (KA).
7. By (KA), $a \leq a^*$. Moreover, a^* is an invariant by Part 6 and (KA) again. Finally, if r is an invariant with $a \leq r$ then $a^* \leq r^* = r$ by isotony of * and Part 6. $\square$

Proof of Theorem 6.5.

1. By Theorem 6.4.6 the invariants are exactly the fixpoints of the * operation. Since this operation is isotone, Tarski's theorem shows the completeness claim. Leastness of 1 in $I(S)$ is an axiom. Since $\top$ is the greatest element, we have $1 \leq \top$ and $\top \cdot \top \leq \top$ and hence $\top \in I(S)$.
2. First, $r \leq r' \;\Rightarrow\; r * r' \leq r' * r' = r'$ by isotony and Theorem 6.4.3.
   Second, by Theorem 6.4.1 $r \leq r * r'$ and hence $r * r' = r'$ implies $r \leq r'$.
3. First, $1 \leq r$ and $1 \leq r'$ imply $1 \leq r \sqcap r'$. Second, by isotony of $*$ and Theorem 6.4.3, $(r \sqcap r') * (r \sqcap r') \leq r * r = r$. Likewise, $(r \sqcap r') * (r \sqcap r') \leq r'$. Hence $(r \sqcap r') * (r \sqcap r') \leq r \sqcap r'$. This shows that $r \sqcap r'$ is in $I(S)$ and therefore also the infimum of r and r' in $I(S)$.
4. First, $1 = 1 * 1 \leq r * r'$ and $(r * r') * (r * r') = r * r * r' * r' \leq r * r'$ show that $r * r' \in I(S)$ as well. The supremum property is a well known fact about the natural order and hence follows from Part 2. The second assertion is straightforward from that and Part 3.

5. Immediate from Part 4.
6. By standard Kleene algebra the operation * is a closure operation. Hence, as shown e.g. in [2] its set of fixpoints $I(S)$ is closed under arbitrary infima. □

Proof of Lemma 6.6.

1. We calculate

$$\begin{array}{ll} & (r * a)^* \leq r * a^* \\ \Leftarrow & \quad \{\![\text{ star induction (2) }]\!\} \\ & 1 + (r * a) * (r * a^*) \leq r * a^* \\ \Leftrightarrow & \quad \{\![\text{ join }]\!\} \\ & 1 \leq r * a^* \ \wedge \ (r * a) * (r * a^*) \leq r * a^* \ . \end{array}$$

   The first conjunct holds by $1 \leq r$ and $1 \leq a^*$. For the second one we have, by $\circ$-idempotence of r, the definition of star, isotony and associativity and commutativity of $*$,

$$r * a^* = (r * r) * a^* \geq (r \,;\, r) * (a * a^*) = (r * a) \,;\, (r * a^*) \ .$$

2. By Part 1, isotony of $*$ and idempotence of r we have

$$r * (r * a)^* \leq r * r * a^* = r * a^* \ .$$

   For the reverse inequation we first conclude $a \leq r * a$ from Theorem 6.4.1 and then use isotony of * and $*$. □

To prove Theorem 7.1, we first show an auxiliary lemma about the dependence relation. We recall from Section 2 the function $dep(tp) = \{q \mid \exists\, p \in tp : q \rightarrow^+ p\}$ on traces tp.

Definition B.1 Consider traces tp, tr with $tp \cap tr = \emptyset$. We define

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

and call the pair (tr', tr'') the *dependence split* of tr w.r.t tp. Then $tr' \cup tr'' = tr$.

Lemma B.2 *Consider arbitrary traces tp and tq.*

1. *The function dep is $\subseteq$-isotone and hence subdistributive over intersection, i.e., $dep(tp \cap tq) \subseteq dep(tp) \cap dep(tq)$.*
2. *$dep(dep(tp)) \subseteq dep(tp)$.*

Let now tp, tr be traces with $tp \cap tr = \emptyset$ and let (tr', tr'') be the dependence split of tr w.r.t tp.

3. *$dep(tr') \subseteq dep(tr) \cap dep(tp)$.*
4. *$tr'' \cap dep(tp) = \emptyset$.*

5. *For arbitrary trace tq we have $tq \cap dep(tp) = \emptyset \Rightarrow tq \cap dep(tr') = \emptyset$.*
6. *$tp \cap dep(tp) = \emptyset \Rightarrow tp \cap dep(tr') = \emptyset$.*
7. *$tr'' \cap dep(tr') = \emptyset$.*
8. *Assume that $\rightarrow$ is acyclic and $tp = \{e\}$ for some event $e \in EV$. Then $\{e\} \cap dep(tr') = \emptyset$ and hence $\{tr\} * [e] = \{tr'\} ; [e] ; \{tr''\}$.*

Proof. 1. As a general property, $\subseteq$-isotony is equivalent to subdistributivity over intersection.
2. Immediate from transitivity of $\rightarrow^+$.
3. By Parts 1 and 2,

$$dep(tr') = dep(tr \cap dep(tp)) \subseteq dep(tr) \cap dep(dep(tp)) \subseteq dep(tr) \cap dep(tp) \ .$$

4. Immediate from the definition of tr'' and Boolean algebra.
5. By Part 3 and the assumption about tq,

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

6. Immediate from Parts 3 and 5.
7. Immediate from Parts 4 and 5.
8. This follows from the equivalence

$$\{e\} \cap dep(\{e\}) = \emptyset \Leftrightarrow e \notin dep(\{e\}) \Leftrightarrow \neg(e \rightarrow^+ e) \ .$$

□

Proof of Theorem 7.1.
We first note that power invariants $R = \mathcal{P}(E)$ satisfy a stronger form of downward closure than the one stated in Theorem 6.5.5, namely $tr \in R \wedge tr' \subseteq tr \Rightarrow tr' \in R$. In particular, the components of any dependence split of tr are in R again.

1. If $e \in E$ then $R * [e] = \emptyset$ and the claim holds trivially. Hence we calculate, assuming $e \notin E$:

$$\begin{array}{ll} & R * [e] \\ = & \quad \{\![\text{ definition of } *]\!\} \\ & \bigcup_{tr \in R} \{tr * \{e\}\} \\ \subseteq & \quad \{\![\text{ by Lemma B.2.8 and downward closure of } R]\!\} \\ & \bigcup_{tr' \in R} \bigcup_{tr'' \in R} \{tr' ; \{e\} ; tr''\} \\ = & \quad \{\![\text{ definition of } ;]\!\} \\ & R ; [e] ; R \ . \end{array}$$

2. We show the property for singleton programs $P = \{tp\}$, $Q = \{tq\}$ with traces tp, tq; then a similar calculation as for Part 1 extends it to arbitrary programs P, Q.

The property holds trivially if $R*(P;Q) = \emptyset$. Therefore assume $R*(P;Q) \neq \emptyset$ and consider an arbitrary trace $tr \in R$ with $\{tr\} * (P\,;Q) \neq \emptyset$. This implies that tp, tq, tr are pairwise disjoint and $P\,;Q \neq \emptyset$, hence $dep(tp) \cap tq = \emptyset$. Moreover, $ts =_{df} tr * (tp\,;tq) = tr \cup tp \cup tq$.
Let now (tr', tr'') be the dependence split of tr w.r.t. tp. We show that then $ts = (tr' * tp)\,;(tr'' * tq)$ and hence $ts \in (R * P)\,;(R * Q)$.
(a) By Lemma B.2.7 $dep(tr') \cap tr'' = \emptyset$.
(b) By Lemma B.2.5 $dep(tr') \cap tq = \emptyset$.
(c) By Lemma B.2.4 $dep(tp) \cap tr'' = \emptyset$.
Now, by definition of tr', tr'', associativity and commutativity of union and (a),(b),(c) as well as $dep(tp) \cap tq = \emptyset$ we have

$$ts = tr \cup tp \cup tq = tr' \cup tr'' \cup tp \cup tq = tr' \cup tp \cup tr'' \cup tq = (tr' * tp)\,;(tr'' * tq)\ .$$

□

Proof of Lemma 7.5.

1. For the first equation we use the fusion law (8) with the functions $f(x) =_{df} r * x$, $g(x) =_{df} 1 + a \circ x$ and $h(x) =_{df} r + (r * a) \circ x$. First, by the quantale assumptions, f is strict and continuous. Second,

$$\begin{array}{ll} & f(g(x)) \\ = & \quad \{\![\text{ definitions }]\!\} \\ & r * (1 + a \circ x) \\ = & \quad \{\![\text{ distributivity of } * \text{ over } +]\!\} \\ & r * 1 + r * (a \circ x) \\ = & \quad \{\![\text{ neutrality of 1 and } *\text{-distributivity }]\!\} \\ & r + (r * a) \circ (r * x) \\ = & \quad \{\![\text{ definitions }]\!\} \\ & h(f(x))\ . \end{array}$$

   For the equation $r * a^* = r \circ (r * a)^*$ we choose symmetrically $g'(x) =_{df} 1 + x \circ a$ and $h'(x) =_{df} r + x \circ (r * a)$.
2. Analogously, with $g(x) =_{df} a + a \circ x$ and $h(x) =_{df} r * a + (r * a) \circ x$. □

Proof of Theorem 8.3.

1. This holds by the definition of invariants.
2. For $\circ = *$ the claim is immediate from the definition of guar and isotony of $*$; for the other operators $\circ$ it follows from $\circ \subseteq *$.
3. Using the assumption, invariance of g and star induction we calculate

$$a \leq g \;\Rightarrow\; a \circ g \leq g \circ g = g \;\Rightarrow\; 1 + a \circ g \leq g \;\Rightarrow\; a^* \leq g\ .$$

4. Immediate from the definitions. □

Proof of Theorem 8.4.
The guarantee part is covered by Theorem 8.3.2. For the remainder we note that the assumptions b' guar g' guar r and b guar g guar r' by transitivity of guar imply b' guar $r \wedge b$ guar r' and calculate

$$
\begin{array}{ll}
& (a \sqcap a') \,;\, ((r \sqcap r') * (b * b')) \leq c \sqcap c' \\
\Leftrightarrow & \quad \{\![\text{ characterisation of intersection }]\!\} \\
& (a \sqcap a') \,;\, ((r \sqcap r') * (b * b')) \leq c \;\wedge\; (a \sqcap a') \,;\, ((r \sqcap r') * (b * b')) \leq c' \\
\Leftarrow & \quad \{\![\text{ intersection, isotony }]\!\} \\
& a \,;\, (r * (b * b')) \leq c \wedge a' \,;\, (r' * (b * b')) \leq c' \\
\Leftarrow & \quad \{\![\; b' \text{ guar } r \wedge b \text{ guar } r' \text{ and isotony }]\!\} \\
& a \,;\, (r * (b * r)) \leq c \wedge a' \,;\, (r' * (r' * b')) \leq c' \\
\Leftarrow & \quad \{\![\text{ associativity and commutativity of } * \,]\!\} \\
& a \,;\, ((r * r) * b) \leq c \wedge a' \,;\, ((r' * r') * b') \leq c' \\
\Leftrightarrow & \quad \{\![\text{ idempotence of } * \text{ on invariants (Theorem 6.4.3) }]\!\} \\
& a \,;\, (r * b) \leq c \wedge a' \,;\, (r' * b) \leq c' \\
\Leftarrow & \quad \{\![\text{ definition of quadruples and assumption }]\!\} \\
& \mathsf{TRUE} \;.
\end{array}
$$

Proof of Theorem 8.5.
The guarantee part is again covered by Theorem 8.3.2. Specialising b, d, b', c, e in Lemma 5.3 to $(r * b), c, (r' * b'), c', ((r \sqcap r') * (b \,;\, b'))$, respectively, we obtain that the weakest condition implying the remainder of the claim is

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

Since Theorem 6.5.6 and the assumption on I imply $r \sqcap r' \in I$ again, this follows by $*$-distributivity and isotony of $*$ and $;$.

Proof of Theorem 8.6.

1. The guarantee part 1 guar g holds by the definition of invariants. For the remainder of the claim we have by the definition and neutrality of 1,
   $a \,;\, (r * 1) \leq s \;\Leftrightarrow\; a \,;\, r \leq s \;\Leftrightarrow\; a \,\{\!|\, r \,|\!\}\, s$.
2. By the definitions, distributivity and lattice algebra we have
   $a \; r \,\{\!|\, b + b' \,|\!\}\, g \; s \;\Leftrightarrow\; a \,;\, (r * (b + b')) \leq s \wedge b + b' \leq g \;\Leftrightarrow$
   $a \,;\, (r * b) + a \,;\, (r * b') \leq s \wedge b \leq g \wedge b' \leq g \;\Leftrightarrow\; a \; r \,\{\!|\, b \,|\!\}\, g \; s \wedge a \; r \,\{\!|\, b' \,|\!\}\, g \; s$.
3. This is immediate from Theorem 7.1.1. □

Proof of Theorem 8.7.
The first law is immediate from Lemma 7.5.2, Lemma 5.2.8 and Theorem 8.3.3. The second one follows from the first one by $b^* = 1 + b^+$ and the choice and skip rules.

As a sample input file for Prover9 we show the one for proving some of the laws about Hoare triples from Section 5. One sees that the axioms and the proof goals can be stated almost in the same syntax as we have used in our definitions. Since Prover9 allows only one positive goal, most of the goals are commented out. A collection of further input files and proofs can be found under `http://www.dcs.shef.ac.uk/~georg/ka/`.

```
formulas(assumptions).

% partial order
  x <= x.
  x <= y & y <= z -> x <= z.
  x <= y & y <= x -> x = y.

% ordered monoid
  x;(y;z) = (x;y);z.
  1;x = x.
  x;1 = x.
  x <= y -> x;z <= y;z.
  y <= z -> x;y <= x;z.

% Hoare triple
  hoa(x,y,z) <-> x;y <= z.

% idempotent semiring
%  x+(y+z) = (x+y)+z.
%  0+x = x.
%  x+0 = x.
%  x+x = x.
%  x+y = y+x.
%  x;0 = 0.
%  0;x = 0.
%  x;(y+z) = x;y + x;z.
%  (x+y);z = x;z + y; z.
%  x <= y <-> x+y = y.
end_of_list.

formulas(goals).
%hoa(x,1,z) <-> x <= z.
(all x all z (hoa(x,y1,z) -> hoa(x,y2,z))) <-> y2 <= y1.
%(all x all z (hoa(x,y1,z) <-> hoa(x,y2,z))) <-> y1 = y2.
%hoa(x,y1;y2,z) <-> (exists u  (hoa(x,y1,u) & %hoa(u,y2,z))).
% for idempotent semiring:
%hoa(x,0,z).
hoa(x,y1+y2,z) <-> hoa(x,y1,z) & hoa(x,y2,z).
end_of_list.
```