Concurrent Kleene Algebra and its Foundations[☆]

Tony Hoare

Microsoft Research, Cambridge, UK

Bernhard Möller

Universität Augsburg, Germany

Georg Struth

University of Sheffield, UK

Ian Wehrman

University of Texas at Austin, USA

Abstract

A Concurrent Kleene Algebra offers two composition operators, related by a weak version of an exchange law: when applied in a trace model of program semantics, one of them stands for sequential execution and the other for concurrent execution of program components. After introducing this motivating concrete application, we investigate its abstract background in terms of a primitive independence relation between the traces. On this basis, we develop a series of richer algebras; the richest validates a proof calculus for programs similar to that of a Jones style rely/guarantee calculus. On the basis of this abstract algebra, we finally reconstruct the original trace model, using the notion of atoms from lattice theory.

Keywords: concurrency, dependence, algebra, Hoare calculus, rely/guarantee calculus

1. Introduction

Kleene algebra [10] has been recognised and developed [26, 27, 11] as an algebraic framework (or structural equivalence) that unifies diverse theories for conventional sequential programming by axiomatising the fundamental concepts of choice, sequential composition and finite iteration. Its many familiar models include binary relations, with operators for union, relational composition and reflexive transitive closure, as well as formal languages, with operators for union, concatenation and Kleene star.

This paper introduces a 'double' Kleene algebra, which adds an operator for concurrent composition. In fact, we summarise a whole family of algebras under the common heading of *concurrent Kleene algebra (CKA)*. In it, sequential composition ; and concurrent composition $*$ are related by the law $(a * b) ; (c * d) \leq (a ; c) * (b ; d)$, an inequational weakening of the corresponding equational exchange law of two-category or bicategory theory (cf. [29]). For elements r that satisfy special conditions (including $r ; r = r$) this weak form can be strengthened to the equational law $r * (a ; b) = (r * a) ; (r * b)$, by which concurrent composition distributes through sequential. The purpose of the paper is to introduce the basic operators and their laws, and study them both in their concrete representation and in their abstract, axiomatic form.

[☆]Significantly extended and revised version of [21, 22].

Email addresses: thoare@microsoft.com (Tony Hoare), moeller@informatik.uni-augsburg.de (Bernhard Möller), g.struth@dcs.shef.ac.uk (Georg Struth), iwehrman@cs.utexas.edu (Ian Wehrman)

The interest of CKAs is two-fold. First, they express in their most general form the essential properties of program execution; in fact, the properties which are preserved by massively re-ordering program optimizers, by multiple threads sharing volatile variables in main memory, and even by execution on concurrent architectures with weakly ordered memory access, and computer networks connected by unreliable communication channels [18]. Second, 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 (Hoare triples [16]) and in concurrent style (rely/guarantee calculus [25]).

In our concrete model of program semantics, a program is identified with the set of traces of all the executions it may evoke. Each of the traces consists of the set of events that occur during a single execution. When two sub-programs 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. In fact, this simple definition is exactly the general definition of concurrent execution; it permits the concurrent operands to communicate in any way, for example, through channels or through access to shared memory (or even both). The definitions of our other operators are only slightly more complicated. They place restrictions on which traces may be selected for combination.

An unusual feature of our model is that we treat predicates, i.e., assertions and specifications, in exactly the same way as programs. An assertion used as a precondition is modelled as the most general, i.e., most non-deterministic, program which reaches a final state satisfying the given predicate. Our operators then have dual interpretations on predicates and programs, though of course they have the same meaning as applied to sets of traces. Union of predicates represents disjunction, but union of programs represents non-deterministic choice, made by an implementation, either at compile time or at run time. Implication between predicates is just implication (set inclusion); but implication between programs represents program refinement. Implication of a specification by a program represents program correctness.

The refusal to make distinctions between sorts is a great simplification of the algebra. Important distinctions can be introduced later, by defining subsets of elements of the algebra that satisfy particular laws, known as healthiness conditions. For example, we can define predicates to be a Boolean algebra, whereas we do not want to apply negation to programs, because that would violate their computability.

In addition to events, the other primitive of our model is a dependence relation between the events of a trace. This gives to the trace the structure of a directed graph. The transitive closure of primitive dependence represents a direct or indirect chain of dependences; it imposes time constraints on the ordering of the occurrence of events. In a sequential composition, it is obviously not allowed for an event occurring in execution of the first operand to depend on an event occurring in execution of the second operand. We take this as our definition of a liberal form of sequential composition.

The absence of dependence is as important as dependence, because it models execution of independent events that may overlap in time ('true' concurrency). To exploit independence, we define a stronger form of disjoint concurrent composition which requires all events of one operand to be independent of all events of the other, with no interaction or interference or influence of one upon the other. This is an analogue of the separating conjunction [35]. However, in this paper we do not treat ownership of variables or other resources. Thus our theory allows any or all of the variables declared in a program to be volatile, with arbitrary interference at any time. That is a reason why our algebra has to be quite weak.

Other examples of programming operators which can be simply defined by their traces include a strong form of sequentiality, in which all events of the second operand depend directly or indirectly on all events of the first operand. Another example (defined in the next section) is a form of more or less determinate selection between operands. Selection is made in one of two ways: by the program itself, for example by a Boolean condition in a conditional (or guarded) command; or it can be made by a successful interaction with the environment, for example a communication on a channel, as in the external choice operator of a process algebra. In this paper, we do not explore the properties of these additional operators.

This paper concentrates entirely on the control structure of programs. It completely ignores data flow, and the computer resources which mediate it. 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 synchronized, double-

ended or multiplexed, reliable or lossy, and perhaps subject to stuttering or even re-ordering of messages. The behaviour of variables and channels of all these kinds can also be modeled as sets of traces [18]. Control structure and data structure are brought together by the selection and description of the primitive events and actions that can be performed on the data. In this paper we do not describe or even list the primitive events by which the program evokes operations on the data.

Our model and its theorems may look elegant, but when applied to actual programs, they are far too weak to prove anything useful. In the case of concurrency, if all variables can be volatile, hardly any interesting property is true! To construct a useful calculus for concrete programs, our theory has to be extended. There are many ways to do this. New operators can be defined, and new algebraic laws can be proved about them; new components can be added to the model, for example, labels can be attached to the nodes and arrows of the graph; new data types can be defined, as described in the previous paragraph; or sort distinctions can be introduced, for example by healthiness conditions.

Fortunately, none of these developments invalidate any of our axioms, and hence their introduction does not require fresh proofs of any of our theorems. Thus, it is our hope that the theory may be developed by many theorists independently, and applied to a range of well-known programming languages and calculi. Interesting directions of development are mentioned in the conclusions of the paper.

In the last technical section, we introduce the notion of an event-based concurrent Kleene algebra which reconstructs the concrete trace model in terms of the more abstract order-theoretic notions of atoms and irreducible elements. We show that in such algebras the dependence relation can be recovered from the operators of sequential and concurrent composition. This reconfirms the basic theory and the trace model and shows that the algebraic and the dependence-based mathematical definitions are in a sense equivalent. Most of our reasoning has been checked by computer using the automated theorem proving system Prover9/Mace4 [30]. A collection of input files and proofs can be found in the accompanying report [23].

The paper is organized as follows. Section 2 summarizes the definitions of the trace model and its essential operators. In Section 3 we develop an abstract calculus of independence relations, which then is algebraised in Section 4. After that, Section 5 presents idempotent semirings and quantales as fundamental algebraic structures. In Section 6 we give axiomatisations of various concurrent structures that offer two operators for concurrent and sequential composition, related by the above-mentioned inequational exchange law. In Section 7 we give a more abstract view of the composition operators used in the concrete trace model. Section 8 enriches the setting by operators for finite and infinite iteration, which leads to concurrent Kleene and omega algebras. In Section 9 we present an algebraic view of Hoare triples, which serve as basic ingredients of the rely/guarantee calculus of later sections. As a preparation for that, Section 10 gives a definition of invariants. In Section 11 we establish the equivalence of two fundamental laws with (weak) acyclicity and transitivity of the basic dependence relation. The results are used in Section 12 to define a further class of algebras that are tailored to the needs of the rely/guarantee calculus presented in Section 13 and, in a simplified form, in Section 14. Finally, Sections 15 and 16 develop the notion of event-based concurrent algebras and reconstruct the trace model and the dependence relation in terms of that notion. Section 17 presents related work, while Section 18 contains conclusion and outlook. Appendix A summarises the laws characterising the most important algebraic structures involved. Appendix B shows a sample input file for the automated theorem prover Prover9.

2. Operators on Traces and Programs

This section presents a concrete model of Concurrent Kleene Algebra which serves as a motivation for the abstract algebraic treatment in later sections.

We assume a set EV of *events*, which are occurrences of primitive actions, together with a *dependence relation* $\to \subseteq EV \times EV$ between them: $e \to f$ indicates a flow of data or control from event e to event f. No particular properties of $\to$ are presupposed.

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

We keep the definition of traces and programs so liberal to accommodate systems with very loose coupling of events; e.g., "conventional" linear traces can be obtained by including unique 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 false $=_{df} \emptyset$ has no traces, and therefore cannot be executed at all. In the context of program development by stepwise refinement, it serves the rôle of the 'miracle' [33] .

Following [19] we study 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 concurrent composition, with no dependence in either direction;
$P [] Q$	alternation – at most one of P or Q contributes events.

Specific interpretations of the concepts in this list in various models of concurrency are discussed in Section 17. To express the restrictions in these concepts we introduce the following notion.

Definition 2.2. A trace tp is *independent* of a trace tq, written $tp \not\leftarrow tq$, if there are no dependence arrows from events of tq to events of tp:

$$tp \not\leftarrow tq \;\Leftrightarrow_{df}\; \neg\exists\, e \in tp, f \in tq : f \to e \ .$$

The intention is that the events in tq do not influence or constrain the execution of the events in tp in the sense that these do not depend in any way on any event in tq.

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

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

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

The above informal descriptions are captured by the definitions

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

It is clear that $([]) \subseteq (\parallel) \subseteq (;) \subseteq (*)$ and that $(*)$, $(\parallel)$ and $([])$ are symmetric. Moreover, skip is a neutral element for all operators $\circ \in \{*, ;, \parallel, []\}$, i.e.,

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

The operator [] can be explained as follows: A trace tr is in $P [] Q$ iff tr is in P and Q contains the empty trace (as a kind of an indication that Q will completely give way to a P-trace) or tr is in Q and P contains the empty trace.

Example 2.3. We illustrate the operators with a small example. 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 $x := x + 1, y := y + 2, z := x + 3$, respectively. 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 at the right-hand side of f. This means that for our three events we have exactly $ax \to az$. 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

$*$	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	$\emptyset$	$[ay, az]$	$\emptyset$

$\parallel$	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	$\emptyset$	$\emptyset$	$\emptyset$

Figure 1: Composition tables

$[\,]$ for skip). Figure 1 lists the composition tables for our operators on these programs. They show that the operator $*$ allows forming concurrent programs with race conditions in that, e.g., $P_x * P_y = P_y * P_x$ allows both events ax and az in either order or even concurrently, whereas ; and $\parallel$ respect dependences in that one or both of the compositions yield the empty trace. □

It is straightforward from the definitions that $*, \parallel$ and $[]$ are commutative and that $[] \subseteq \parallel \subseteq ; \subseteq *$, where for operators $\bullet, \circ \in \{*, ;, \parallel, []\}$ the formula $\bullet \subseteq \circ$ abbreviates $\forall P, Q : P \bullet Q \subseteq P \circ Q$. In the remainder of this paper we shall mostly concentrate on the more interesting operators $*$ and ;.

We can now also explain informally, why the exchange law

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

mentioned in the introduction is valid. In both programs, dependence arrows from Q to P and from S to R need to be excluded. However, in the program on the left-hand side, dependence arrows from Q to R and from S to P need to be excluded, too. Therefore the left program may have fewer traces than the right one. This is illustrated by the diagram

In the next section we will develop a simple calculus that allows a formal verification of this and related laws on a general basis.

Another essential operator is union, which again is $\subseteq$-isotone and distributes through arbitrary unions. However, in contrast to $[]$, it is *not* false-strict. When P and Q both contain the empty trace then $P [] Q$ and $P \cup Q$ coincide.

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 Section 8.

3. Aggregation and Independence

To derive interesting laws about our operators in a general and concise way, we take a more abstract view of systems, such as programs, their parts and their interactions. The main concepts we study are *aggregation*—how systems are built from their parts—and *(in)dependence*—how systems and their parts interact.

Definition 3.1. An *aggregation algebra* is a structure $(A, +)$ formed by a set A and a possibly partial binary operator $+ : A \times A \to A$.

When $p+q$ is defined, we interpret it as the system that is formed or aggregated from the parts p and q. For instance, A may be the set of traces and $+$ disjoint trace union. For the time being, the algebra $(A,+)$ need not satisfy any laws. Later we will assume aggregation algebras that are (commutative) semigroups or monoids.

Definition 3.2. An *independence relation* on an aggregation algebra $(A,+)$ is a binary relation R on A that is bilinear in the following sense: whenever the aggregates involved are defined,

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

A system p is *independent of* a system q if $R(p,q)$ holds.

In the sequel we will leave the qualification on definedness implicit. We use predicate rather than infix notation for independence relations to save parentheses around aggregated arguments. The linearity conditions say that a combined system is independent of another one if and only if both its parts are.

Example 3.3.

1. Consider the aggregation algebra $(\mathcal{P}(EV),\cup)$ of traces. On that algebra, our first example of an independence relation is $\not\leftarrow$ as given in Def. 2.2.

2. Consider a set A and the aggregation algebra $(\mathcal{P}(A),\cup)$, where $\cup$ is set union. Then, for all $X,Y \subseteq A$, the relation defined by $R(X,Y)$ if and only if X and Y are disjoint is an independence relation. This holds since $(X \cup Y) \cap Z = \emptyset$ iff $X \cap Z = \emptyset$ and $Y \cap Z = \emptyset$.

3. Consider the set $(G,\cup)$ of digraphs under (disjoint) union. Then, for all digraphs $g_1, g_2 \in G$, the relation defined by $R(g_1,g_2)$ if and only if there is no arrow with source in g_1 and target in g_2 is an independence relation. The same facts hold for digraphs with respect to arrows from g_2 to g_1 and for undirected graphs with respect to adjacency.

4. Consider subspaces of some vector space with $+$ being the span. Then orthogonality is an independence relation.

5. Let t_1 and t_2 be subtrees of a tree t. Let them be related by R if their roots are not on a single path from the root of t to its leaves. Let $+$ correspond to forming the least subtree of t that has both t_1 and t_2 as subtrees. Then R is *not* an independence relation in the above sense, because a tree t_3 which is related by R to both t_1 and t_2 can be "captured" as a subtree of $t_1 + t_2$.

Examples 1–4 show that some natural notions of independence are covered by the above definition, whereas Example 5 shows that some other natural notions, such as disjointness of subtrees in a tree, are not. □

Lemma 3.4. *Let $(A,+)$ be an aggregation algebra and let R be an independence relation.*

1. $R((p+q)+r,s) \;\Leftrightarrow\; R(p+(q+r),s)$.
2. $R(p,(q+r)+s) \;\Leftrightarrow\; R(p,q+(r+s))$.
3. $R(p+q,r) \;\Leftrightarrow\; R(q+p,r)$.
4. $R(p,q+r) \;\Leftrightarrow\; R(p,r+q)$.
5. $R(p+p,q) \;\Leftrightarrow\; R(p,q)$.
6. $R(p,q+q) \;\Leftrightarrow\; R(p,q)$.
7. $R(p+q,r) \wedge R(p,q) \;\Leftrightarrow\; R(q,r) \wedge R(p,q+r)$.

Proof. By bilinearity and the fact that conjunction is associative, commutative and idempotent. □

We now consider two independence relations R and S.

Lemma 3.5. *Let $(A,+)$ be an aggregation algebra. Let R and S be independence relations that satisfy $R \subseteq S$.*

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

Proof. We only prove Part 1; Part 2 is similar.

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

□

Next we prove a property that will imply the crucial inequational exchange law mentioned in the introduction. We write $S^\smile$ for the relational converse of S.

Proposition 3.6. *Let $(A,+)$ be an aggregation algebra. Let R and S be two independence relations. Let $R \subseteq S$ and $S = S^\smile$ (S is symmetric). Then*

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

Proof.

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

□

The proofs in this section are only intended to give a flavour of the approach. In fact, they have all been automated, hence formally verified, with Prover9.

4. Algebraisation of the Calculus

This section further pursues the idea of interpreting independence arrows as algebraic operators. Formally, the algebraisation is achieved by lifting the aggregation algebra to power sets.

Definition 4.1. For an aggregation algebra $(A,+)$ and an independence relation R, we define an operator $\circledR$ of *R-composition* (or complex product w.r.t. R) of type $\mathcal{P}(A) \times \mathcal{P}(A) \to \mathcal{P}(A)$ for all $a, b \subseteq A$ by

$$a \mathbin{\circledR} b =_{df} \{p+q \mid p \in a \wedge q \in b \wedge R(p,q)\}.$$

Example 4.2.

1. In Section 2 we have $\circledcirc = \circ$.

2. Let $A = \Sigma^*$ be the set of strings over the alphabet Σ. For all $a, b \in \Sigma^*$ let $a+b$ be string concatenation and let R be the universal relation which relates all elements of A. Let $B, C \subseteq \Sigma^*$ be sets of strings. Then $A \mathbin{Ⓡ} B$ is the usual complex product of regular language theory.

□

In order to obtain more interesting results, we assume the aggregation algebra to be a *semigroup* or *monoid*, meaning that $+$ is associative and, in the latter case, additionally has a unit 0 that plays the rôle of the empty system. In some cases, we also consider independence relations that are not only bilinear, but also *bistrict*, i.e., they satisfy

$$R(p, 0) \qquad \text{and} \qquad R(0, p).$$

These rather natural assumptions say that the empty system depends on nothing and nothing depends on it.

Call an equational law involving a function *linear* if every variable in it occurs exactly once on each side of the equation. Such laws are inherited by the pointwise extension (see e.g. [28]). Typical examples of such laws are associativity and commutativity. This entails the following result.

Proposition 4.3.

1. *Let $(A, +)$ be a semigroup and R be bilinear. Then $(\mathcal{P}(A), \mathbin{Ⓡ})$ is a semigroup.*
2. *Let $(A, +, 0)$ be a monoid, and R be bilinear and bistrict. Then $(\mathcal{P}(A), \mathbin{Ⓡ}, \{0\})$ is a monoid.*

5. Semirings and Quantales

In powerset algebras, next to the pointwise extensions of basic aggregation algebras, we have all the set theoretic operations available. As already mentioned in Section 2, the most interesting one for us is set union, since it allows modelling nondeterminacy. This is reflected in the following definition.

Definition 5.1.

1. An *idempotent semiring* is a structure $(A, +, \cdot, 0, 1)$ with the following properties.
   - $(A, +, 0)$ is a commutative monoid with idempotent addition, i.e., $a + a = a$ for all $a \in A$.
   - $(A, \cdot, 1)$ is a monoid.
   - Multiplication distributes over addition, i.e., for all $a, b, c \in A$,
   $$a \cdot (b + c) = a \cdot b + a \cdot c \qquad \text{and} \qquad (a + b) \cdot c = a \cdot c + b \cdot c.$$
   - 0 is a left and right annihilator for multiplication, i.e., for all $a \in A$,
   $$a \cdot 0 = 0 = 0 \cdot a \ .$$
2. Every idempotent semiring is partially ordered by
   $$a \leq b \Leftrightarrow_{df} \ a + b = b.$$
   Then $+$ and $\cdot$ are isotone w.r.t. $\leq$ and 0 is the least element. Moreover, $a + b$ is the supremum of $a, b \in A$.
3. An idempotent semiring is called a *quantale* [34, 38] or *standard Kleene algebra* [10] if $\leq$ induces a complete lattice and multiplication distributes over arbitrary suprema. The infimum and the supremum of a subset $B \subseteq A$ are denoted by $\sqcap B$ and $\sqcup B$, respectively. Their binary variants are $a \sqcap b$ and $a \sqcup b$ (the latter coinciding with $a + b$).

Quantales have been used in many contexts beyond program semantics (cf. the c-semirings of [6] or the general reference [38]). They have the advantage that the general fixpoint calculus is available. A number of our proofs in Section 10 need the principle of fixpoint fusion which is a second-order property; in the first-order setting of conventional Kleene and omega algebra (see Section 8) only special cases of it, like the induction and coinduction rules, can be added as axioms. Moreover, in every quantale left and right residuals w.r.t. multiplication can be defined by the Galois connections $x \leq a/b \Leftrightarrow_{df} x \cdot b \leq a$ and $x \leq b\backslash a \Leftrightarrow_{df} b \cdot x \leq a$.

Let again $PR(EV)$ denote the set of all programs over the event set EV (cf. Definition 2.1). The following fact is immediate from the observations in Section 2.

Lemma 5.2. *$(PR(EV), \cup, *, \mathsf{false}, \mathsf{skip})$ and $(PR(EV), \cup, ;, \mathsf{false}, \mathsf{skip})$ are quantales. In each of them $\top = PR(EV)$ is the most general program over EV.*

Proposition 4.3 can now be extended.

Proposition 5.3. *Let $(A, +, 0)$ be a monoid and R be bilinear and bistrict. Then $(\mathcal{P}(A), \cup, \circledR, \emptyset, \{0\})$ is a quantale.*

This follows again from standard results about pointwise extension mentioned in Section 4 (cf. [28]).

6. Concurrent Algebras

The results of the previous section can be generalised to more than one independence relation. Here, we consider only the case of two such relations, R and S, which are defined over one single aggregation algebra.

Definition 6.1.

1. A *bisemigroup* is a structure $(A, *, ;)$ such that $(A, *)$ and $(A, ;)$ are semigroups. A *bimonoid* is a structure $(A, *, ;, 1)$ such that $(A, *, 1)$ and $(A, ;, 1)$ are monoids.
2. An *idempotent bisemiring* is a structure $(A, +, *, ;, 0, 1)$ such that $(A, +, *, 0, 1)$ and $(A, +, ;, 0, 1)$ are idempotent semirings.

We could define bimonoids etc. more generally with two different units $1_;$ and 1_*, but we restrict our attention to cases where these operators share one single unit.

The following statement is immediate from the results of the previous section.

Proposition 6.2. *Let $(A, +, 0)$ be a monoid and let R and S be bilinear. Then $(\mathcal{P}(A), \cup, \circledR, \circledS, \emptyset, \{0\})$ is an idempotent bisemiring.*

In the above statement the independences R and S are unrelated. We now consider the situation where one of them is contained in the other, as in Section 2. This allows us to lift the statements of Lemma 3.5 and the exchange law (Proposition 3.6) to the powerset level.

Lemma 6.3.

1. *Let $(A, +)$ be an aggregation algebra and let R and S be independence relations on A. Then $R \subseteq S$ implies $a \circledR b \subseteq a \circledS b$.*
2. *Let $(A, +)$ be a commutative aggregation algebra and let R be a symmetric independence relation on A. Then $a \circledR b = b \circledR a$.*

Proof. The proof of the first statement is entirely trivial. We display the proof of the second statement to show the rôle of commutativity.

$$\begin{aligned} p \in a \mathbin{\textcircled{R}} b &\Leftrightarrow \exists\, q, r : (p = q + r \wedge q \in a \wedge r \in b \wedge R(q, r)) \\ &\Leftrightarrow \exists\, q, r : (p = r + q \wedge q \in a \wedge r \in b \wedge R(r, q)) \\ &\Leftrightarrow p \in b \mathbin{\textcircled{R}} a. \end{aligned}$$

□

Proposition 6.4. *Let $(A, +)$ be a semigroup and let R and S be bilinear independence relations such that $R \subseteq S$. Then*

1. $(a \mathbin{\textcircled{S}} b) \mathbin{\textcircled{R}} c \subseteq a \mathbin{\textcircled{S}} (b \mathbin{\textcircled{R}} c)$,
2. $a \mathbin{\textcircled{R}} (b \mathbin{\textcircled{S}} c) \subseteq (a \mathbin{\textcircled{R}} b) \mathbin{\textcircled{S}} c$.

Proof. We only prove the first inequality.

$$\begin{aligned} p \in &(a \mathbin{\textcircled{S}} b) \mathbin{\textcircled{R}} c \\ &\Leftrightarrow \exists\, q, r, s : (p = (q + r) + s \wedge q \in a \wedge r \in b \wedge s \in c \wedge R(q + r, s) \wedge S(q, r)) \\ &\Rightarrow \exists\, q, r, s : (p = q + (r + s) \wedge q \in a \wedge r \in b \wedge s \in c \wedge S(q, r + s) \wedge R(r, s)) \\ &\Leftrightarrow p \in a \mathbin{\textcircled{S}} (b \mathbin{\textcircled{R}} c). \end{aligned}$$

The second step uses associativity of $+$, Lemma 3.5.1 and bilinearity. □

Proposition 6.5. *Let $(A, +)$ be a commutative semigroup. Let R and S be bilinear independence relations such that $R \subseteq S$ and S is symmetric. Then the following exchange law holds:*

$$(a \mathbin{\textcircled{S}} b) \mathbin{\textcircled{R}} (c \mathbin{\textcircled{S}} d) \subseteq (a \mathbin{\textcircled{R}} c) \mathbin{\textcircled{S}} (b \mathbin{\textcircled{R}} d).$$

Proof.

$$\begin{aligned} p \in (a \mathbin{\textcircled{S}} b) \mathbin{\textcircled{R}} (c \mathbin{\textcircled{S}} d) \Leftrightarrow &\exists\, q, r, s, t : (p = (q + r) + (s + t) \wedge q \in a \wedge r \in b \wedge s \in c \wedge t \in d \\ &\wedge R(q + r, s + t) \wedge S(q, r) \wedge S(s, t) \\ \Rightarrow &\exists\, q, r, s, t : (p = (q + s) + (r + t) \wedge q \in a \wedge r \in b \wedge s \in c \wedge t \in d \\ &\wedge R(q, s) \wedge R(r, t) \wedge S(q + s, r + t) \\ \Leftrightarrow &b \in (a \mathbin{\textcircled{R}} c) \mathbin{\textcircled{S}} (b \mathbin{\textcircled{R}} d). \end{aligned}$$

The second step uses associativity and commutativity of the aggregation algebra and Proposition 3.6. □

These results motivate the following definitions, abstracting $\textcircled{R}$ to ; and $\textcircled{S}$ to $*$.

Definition 6.6.

1. An *ordered semigroup* is a structure $(A, \cdot, \leq)$ such that $(A, \cdot)$ is a semigroup, A is partially ordered by $\leq$ and $\cdot$ is isotone in both arguments. An *ordered monoid* is a structure $(A, \cdot, 1, \leq)$ such that $(A, \cdot, \leq)$ is an ordered semigroup and $(A, \cdot, 1)$ is a monoid.
2. An *ordered bisemigroup* is a structure $(A, *, ; \leq)$ such that $(A, *, \leq)$ and $(A, ; \leq)$ are ordered semigroups. An *ordered bimonoid* is defined analogously.

3. A *concurrent semigroup* is an ordered bisemigroup $(A, *, ;, \leq)$ that satisfies

$$\begin{aligned} a\,;b &\leq a * b, && (3)\\ a * b &= b * a, && (4)\\ (a * b)\,;c &\leq a * (b\,;c), && (5)\\ a\,;(b * c) &\leq (a\,;b) * c, && (6)\\ (a * b)\,;(c * d) &\leq (a\,;c) * (b\,;d). && (7) \end{aligned}$$

4. A *concurrent monoid* is an ordered bimonoid $(A, *, ;, 1, \leq)$ that satisfies

$$\begin{aligned} a * b &= b * a,\\ (a * b)\,;(c * d) &\leq (a\,;c) * (b\,;d). \end{aligned}$$

5. A *concurrent semiring* is an idempotent bisemiring $(A, +, *, ;, 0, 1)$ such that $(A, *, ;, 1, \leq)$ is a concurrent monoid, where $\leq$ is the natural semiring order.

6. A concurrent semiring $(A, +, *, ;, 0, 1)$ is called a *concurrent quantale* if $(A, +, *, 0, 1)$ and $(A, +, ;, 0, 1)$ are quantales.

Lemma 6.7. *The above axioms for concurrent semigroups and concurrent semirings are irredundant.*

Proof. We have used Mace4 to find models in which all but one of the axioms are true and the remaining axiom is false, for each combination. □

The unit 1 allows us to replace the two concurrent monoid axioms by the single one

$$(a * b)\,;(c * d) \leq (b\,;c) * (a\,;d)\ , \tag{8}$$

which has its free variables in a different order than (7). Moreover, every concurrent monoid is a concurrent semigroup, as can be shown by automated theorem proving:

Lemma 6.8. *The concurrent monoid axioms entail the identities*

$$\begin{aligned} a\,;b &\leq a * b,\\ (a * b)\,;c &\leq a * (b\,;c),\\ a\,;(b * c) &\leq (a\,;b) * c. \end{aligned}$$

Moreover, Mace4 yields a two-element counterexample showing that these laws do not imply the exchange axiom (7).

The development so far can be summarised in the following theorem.

Theorem 6.9. *Let $(A, +)$ be a commutative semigroup and let R and S be bilinear independence relations with $S \subseteq R$ and S symmetric.*

1. *$(\mathcal{P}(A), \circledR, \circledS, \subseteq)$ is a concurrent semigroup.*
2. *$(\mathcal{P}(A), \cup, \circledR, \circledS, \emptyset, \{0\})$ is an concurrent semiring.*

This theorem shows that the entire structure of concurrent algebras can be obtained from the very general assumption of a (commutative) monoidal aggregation algebra and two (strict and) bilinear independence relations.

7. Generalised Sequential and Concurrent Composition

We now check that independence relations for generalised variants of sequential and concurrent composition operators from Section 2 satisfy the bistrictness and bilinearity conditions.

For these particular operators, we assume a distributive lattice $(A, +, \sqcap, 0)$ with least element 0 as the underlying aggregation algebra. This is compatible with all assumptions in previous statements. We also use a strict and additive function $F : A \to A$, which means that it satisfies

$$F(0) = 0 \qquad \text{and} \qquad F(p+q) = F(p) + F(q).$$

Such a function arises, for instance, as the preimage function over a relational structure, defined as

$$F(p) = \{a \mid \exists\, b \in p : R(a,b)\} \qquad \text{or} \qquad F(p) = \{a \mid \exists\, b \in p : R^+(a,b)\},$$

where R^+ denotes the transitive closure of a relation R. In this concrete setting, p is a set.

In our original definitions R would be $\to$ and F the following function *dep* that yields the set of events on which *tp*-events depend.

Definition 7.1. For a trace *tp*, we define the set

$$dep(tp) =_{df} \{f \mid \exists\, e \in tp : f \to e\}\ .$$

We then consider the following operators:

- *fine-grain concurrent composition* $a * b$ with $(*)(p,q) \Leftrightarrow p \sqcap q = 0$;
- *weak sequential composition* $a; b$ with $(;)(p,q) \Leftrightarrow (*)(p,q) \wedge F(p) \sqcap q = 0$;
- *disjoint concurrent composition* $a||b$ with $(||)(p,q) \Leftrightarrow (;)(p,q) \wedge p \sqcap F(q) = 0$;
- *alternation* $a [] b$ with $([])(p,q) \Leftrightarrow p = 0 \vee q = 0$.

In contrast to Section 2 we use predicate notation here for the relations to emphasise the connection to Section 6.

Lemma 7.2.

1. $([]) \subseteq (||) \subseteq (;) \subseteq (*)$.
2. $([]) = ([])^\smile$, $(||) = (||)^\smile$, $(;) \neq (;)^\smile$, $(*) = (*)^\smile$.
3. $([])$, $(||)$, $(;)$ *and* $(*)$ *are bilinear.*
4. $([])$, $(||)$, $(;)$ *and* $(*)$ *are bistrict.*

Proof. The proofs of (1), (2) and (4) are trivial, so we only consider case (3).

- Fine-grain concurrent composition.

$$(*)(p+q,r) \Leftrightarrow (p+q) \sqcap r = 0 \Leftrightarrow p \sqcap r = 0 \wedge q \sqcap r = 0 \Leftrightarrow (*)(p,r) \wedge (*)(q,r).$$

  The second linearity condition is similar.

- Weak sequential composition.

$$\begin{aligned}(;)(p+q,r) &\Leftrightarrow (*)(p+q,r) \wedge F(p+q) \sqcap r = 0\\ &\Leftrightarrow (*)(p,r) \wedge (*)(q,r) \wedge (F(p) + F(q)) \sqcap r = 0\\ &\Leftrightarrow (*)(p,r) \wedge (*)(q,r) \wedge F(p) \sqcap r = 0 \wedge F(q) \sqcap r = 0\\ &\Leftrightarrow (;)(p,r) \wedge (;)(q,r).\end{aligned}$$

  The second linearity condition is again similar.

- Disjoint concurrent composition. The proof is similar to the previous one.
- Alternation.

$$(\!|\,|\!)(p+q,r) \Leftrightarrow p+q=0 \vee r=0 \Leftrightarrow (p=0 \wedge q=0) \vee r=0 \Leftrightarrow (\!|\,|\!)(p,r) \wedge (\!|\,|\!)(q,r).$$

□

These lemmas together with the previous considerations show in particular that the combination of fine-grain concurrency and weak sequential composition leads to concurrent monoids.

8. Iteration: Kleene and Omega Algebras

We now repeat the well known axiomatisations of finite and infinite iteration a^* and a^ω of an element a and work out what they mean for our idempotent semirings of programs.

Definition 8.1.

1. A *Kleene algebra* [26] is a structure $(A,+,\cdot,^*,0,1)$ such that $(A,+,\cdot,0,1)$ is an idempotent semiring and the star operation * satisfies the unfold and induction laws

$$1+a\cdot a^* \leq a^*, \qquad 1+a^*\cdot a \leq a^*, \tag{9}$$
$$c+a\cdot b\leq b \Rightarrow a^*\cdot c\leq b, \qquad c+b\cdot a\leq b \Rightarrow c\cdot a^*\leq b. \tag{10}$$

   The Kleene star should not be confused with the separation operator $*$ above.

2. 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.

3. An *omega algebra* [9] is a structure $(A,+,\cdot,^*,^\omega,0,1)$ such that $(A,+,\cdot,^*,0,1)$ is a Kleene algebra and the omega operator $^\omega$ satisfies the unfold and coinduction laws

$$a^\omega \leq a\cdot a^\omega, \qquad b\leq c+a\cdot b \Rightarrow b\leq a^\omega + a^*\cdot c.$$

The axioms of Kleene and omega algebras entail many useful laws. As examples we mention

$$1\leq a^*\ , \quad a\leq a^*\ , \quad a^*\cdot a^* = (a^*)^* = a^*\ , \quad (a+b)^* = a^*\cdot(b\cdot a^*)^*\ , \tag{KA}$$
$$1^\omega = \top\ , \qquad (a\cdot b)^\omega = a\cdot(b\cdot a)^\omega\ , \qquad (a+b)^\omega = a^\omega + a^*\cdot b\cdot(a+b)^\omega\ . \tag{OA}$$

It is well known that in a quantale A the finite iteration a^* exists for all $a\in A$ and is given by $a^* = \mu x\,.\,1+a\cdot x$, where μ denotes the least fixpoint operator. Since in a quantale the defining function for star is continuous, Kleene's fixpoint theorem shows that $a^* = \bigsqcup_{i\in\mathbb{N}} a^i$. If the complete lattice $(A,\leq)$ in a quantale A is completely distributive, i.e., if $+$ distributes over arbitrary infima, then also the infinite iteration a^ω exists for all $a\in A$ and is given by $a^\omega = \nu x\,.\,a\cdot x$, where ν denote the greatest fixpoint operator.

We now define concurrent versions of these types of algebras.

Definition 8.2.

1. A *bi-Kleene algebra* is a structure $(A,+,*,;,\circledast,\odot,0,1)$ such that $(A,+,*,\circledast,0,1)$ and $(A,+,;,\odot,0,1)$ are Kleene algebras.

2. A *concurrent Kleene algebra (CKA)* is a bi-Kleene algebra $(A,+,*,;,\circledast,\odot,0,1)$ over a concurrent monoid $(A,*,;,1,\leq)$.

3. *Bi-omega algebras* and *concurrent omega algebras* are defined analogously.

The above discussion entails the following result.

Theorem 8.3. *Let $(A,+,0)$ be a commutative monoid and let R and S be bilinear and bistrict independence relations with $S \subseteq R$ and S symmetric. Then the structure $(\mathcal{P}(A), \cup, \circledR, \circledS, {}^{\circledR}, {}^{\circledS}, \emptyset, \{0\})$ is a concurrent Kleene algebra with $a^{\circledT} =_{df} \mu x \,.\, \{0\} \cup a \circledT x$ for $T \in \{R,S\}$. An analogous property holds for omega iteration, for which the greatest-fixpoint operator ν is used.*

Corollary 8.4. *Let A be a bounded distributive lattice and let $*$ and $;$ be defined as in Section 7. Then the structure $(\mathcal{P}(A), \cup, *, ;, {}^{\circledast}, {}^{\circledcirc}, \emptyset, \{0\})$ is a concurrent Kleene algebra with $a^{\circledast} =_{df} \mu x \,.\, 1 + a * x$ and $a^{\circledcirc} =_{df} \mu x \,.\, 1 + a \,;\, x$*

We now explain the behaviour of iteration in our program quantales. For a program P, the program $P^{\circledcirc}$, denoted by P^{∞} in [19], consists of all sequential compositions of finitely many traces in P. The program $P^{\circledast}$ consists of all disjoint unions of finitely many traces in P; it may be considered as describing all finite concurrent spawnings of traces in P.

The disjointness requirement that is built into the definition of $*$ and $;$ does not mean that an iteration cannot repeat a primitive action a: the iterated program just needs to supply sufficiently many (e.g., countably many) events that stand for occurrences of a; it can then use a fresh one from these in each round of iteration.

Example 8.5. With the notation of Example 2.3, let $P =_{df} P_x \cup P_y \cup P_z$. We first look at the powers of P 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 $P^{\circledast}$ consists of all traces with at most three events from $\{ax, ay, az\}$ (the empty trace is in $P^{\circledast}$, too, since by definition skip is contained in every program of the form $Q^{\circledast}$). Hence $P^{\circledast}$ coincides with the set of all possible traces over the three events; this connection will be taken up again in Section 10.

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 $e \not\leftarrow f$ or $f \not\leftarrow e$, i.e., provided the trace $[e, f]$ is consistent with the dependence relation. Only if there were a cyclic dependence $e \leftarrow f \leftarrow e$ we would have $Q \,;\, Q = \emptyset$, but still $Q * Q = [e, f]$. □

Since $PR(EV)$ is a power set lattice, it is completely distributive. Hence it forms a concurrent omega algebra. The infinite iteration P^{ω} w.r.t. the composition operator $*$ is similar to the unbounded concurrent spawning $!P$ of traces in P in the π-calculus (cf. [39]).

9. Hoare Calculus

Essential tools for reasoning about programs are the Hoare calculus and its variants for the concurrent setting. We now show how to treat the Hoare calculus algebraically in our setting. In [19], Hoare triples relating programs are defined by $P \,\{\!\{Q\}\!\}\, R \Leftrightarrow_{df} P \,;\, Q \subseteq R$. Hence such a triple expresses that the program Q is guaranteed to extend every trace in the "pre-history" P to a trace in R.

Again, it is beneficial to abstract from the concrete case of programs.

Definition 9.1. Given an ordered monoid $(A, \cdot, 1, \leq)$ we define, for elements $a, b, c \in A$, the *Hoare triple* $a \,\{\!\{b\}\!\}\, c$ by

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

We show that this very general definition entails all the familiar properties of Hoare triples associated with *partial* correctness.

Lemma 9.2. *Assume an ordered monoid* $(A, \cdot, 1, \leq)$.

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

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 Kleene algebra, we also have

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

Proof.

1. Immediate from the definitions and neutrality of 1.
2. ($\Leftarrow$) follows directly from isotony of composition. For ($\Rightarrow$) set $a = 1$ and $c = b$, and expand the definition.
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 the definition of the supremum,

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

8. The implication ($\Leftarrow$) of the first equivalence follows from Part 2 and $b \leq b^+$. For ($\Rightarrow$) we have, using the definitions, the second star induction rule in (10) and idempotence of $+$,

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

   The second equivalence follows from $b^* = 1 + b^+$ and the skip and choice rules. □

Lemma 9.2 can be expressed more concisely in relational notation. For $b \in A$ the relation $\{b\} \subseteq A \times A$ between precondition elements a and postcondition elements c is defined 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. $\{0\} = \top\!\!\top$ where $\top\!\!\top$ 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.

Properties 4 and 2 allow us to determine the weakest premise ensuring that two composable Hoare triples establish a third one:

Lemma 9.3. *Assume again an ordered monoid $(A, \cdot, 1, \leq)$. Then*

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

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

Lemma 9.4. *Let $A = (A, +, 0, *, ;)$ be a concurrent semigroup and $a, a', b, b', c, c', d \in A$ 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)*

Proof.

1. $a \,\{b\}\, c \;\wedge\; a' \,\{b'\}\, c'$
   $\Leftrightarrow$ {[definition]}
   $a \,;\, b \leq c \;\wedge\; a' \,;\, b' \leq c'$
   $\Rightarrow$ {[isotony of $*$]}
   $(a \,;\, b) * (a' \,;\, b') \leq c * c'$
   $\Rightarrow$ {[exchange (7)]}
   $(a * a') \,;\, (b * b') \leq c * c'$
   $\Leftrightarrow$ {[definition]}
   $(a * a') \,\{b * b'\}\, (c * c')$

2. $a \,\{b\}\, c$
   $\Leftrightarrow$ {[definition]}
   $a \,;\, b \leq c$
   $\Rightarrow$ {[isotony of $*$]}

$$
\begin{array}{ll}
& d * (a\,;b) \leq d * c \\
\Rightarrow & \quad \{\![\text{ by Lemma 6.8.5 }]\!\} \\
& (d * a)\,;b \leq d * c \\
\Leftrightarrow & \quad \{\![\text{ definition }]\!\} \\
& (d * a)\,\mathbf{\{}b\mathbf{\}}\,(d * c)\ .
\end{array}
$$

□

10. Invariants

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

Definition 10.1. 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 a program often 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 10.2. Consider again the event set EV from Example 2.3. 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 working with the concurrent 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 13, invariants will be used to express properties of the environment on which a program wants to rely (whence the identifier R).

Power invariants satisfy many useful laws. To state them, we want to define a function that maps a program to the smallest power invariant containing it.

Let $\lceil P \rceil =_{df} \bigcup P$ denote the set of all events occurring in traces of a program P; when convenient, $\lceil P \rceil$ can also be considered as a trace. It is straightforward to check that the function $\lceil \_ \rceil$ distributes through arbitrary unions. Hence it has an upper adjoint F, defined by the Galois connection

$$\lceil P \rceil \subseteq X \Leftrightarrow P \subseteq F(X)\ .$$

This entails $F(X) = \mathcal{P}(X)$ and $\lceil \mathcal{P}(X) \rceil = X$. Moreover, as adjoints of a Galois connection, $\mathcal{P}(\_)$ and $\lceil \_ \rceil$ are $\subseteq$-isotone. Setting $X = \lceil P \rceil$ yields $P \subseteq \mathcal{P}(\lceil P \rceil)$. Thus for $X, Y \subseteq EV$ we have $\mathcal{P}(X) \subseteq \mathcal{P}(Y) \Leftrightarrow X \subseteq Y$.

Motivated by the above remarks, we now define $\mathsf{INV}(P) =_{df} \mathcal{P}(\lceil P \rceil)$. 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, too.

We now prepare for our abstract notion of invariant. An *invariant* is a program R with $R = \mathsf{INV}(R)$. In particular, every invariant in our concrete quantale of programs is a power invariant. In general concurrent semirings we will replace INV by a suitable abstract operator the properties of which will be discussed below. By definition, invariants are fixpoints of an isotone function and hence, by Tarski's theorem, form a complete lattice under the inclusion order.

The operator ∇ from [19] and INV are interrelated. To show this, we set $\mathsf{SINGLES}(P) =_{df} \{\{e\} \mid \{e\} \in P\}$. 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 10.3. *Let P and Q be programs.*

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. $\mathsf{skip} \subseteq \mathsf{INV}(P)$.
5. $\mathsf{INV}(P * Q) \subseteq \mathsf{INV}(P \cup Q)$.
6. $\mathsf{INV}(P) * \mathsf{INV}(P) \subseteq \mathsf{INV}(P)$.

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, $\lceil \mathcal{P}(X) \rceil = X$, we have
$$\mathsf{INV}(\mathsf{INV}(P)) = \mathcal{P}(\lceil \mathcal{P}(\lceil P \rceil) \rceil) = \mathcal{P}(\lceil P \rceil) = \mathsf{INV}(P) \ .$$
3. By Part 1 we have $P \subseteq \mathsf{INV}(P)$. By the Galois connection INV is isotone and by Part 2 it is idempotent.
4. Immediate from the definition of INV.
5. By the definition of $*$ we have $\lceil P * Q \rceil \subseteq \lceil P \cup Q \rceil$ and the property follows by isotony of $\mathcal{P}$.
6. From the definitions it is straightforward to check that $\lceil P * Q \rceil \subseteq \lceil P \rceil \cup \lceil Q \rceil$. Hence
$$\begin{array}{lll} \mathsf{INV}(P) * \mathsf{INV}(P) & \subseteq & \mathsf{INV}(\mathsf{INV}(P) * \mathsf{INV}(P)) = \mathcal{P}(\lceil \mathsf{INV}(P) * \mathsf{INV}(P) \rceil) \subseteq \mathcal{P}(\lceil \mathsf{INV}(P) \rceil \cup \lceil \mathsf{INV}(P) \rceil) \\ & = & \mathcal{P}(\lceil \mathsf{INV}(P) \rceil) = \mathsf{INV}(\mathsf{INV}(P)) = \mathsf{INV}(P) \end{array}$$
by Parts 1 and 2. □

Since INV is a closure operator we have the following (cf. [5]).

Corollary 10.4. *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, respectively.*

We now abstract again from the concrete case of programs. It turns out that the properties in Theorem 10.3.4 and 10.3.6 largely suffice for characterising invariants.

Definition 10.5. An *invariant* in an ordered monoid A is an element $r \in A$ satisfying $1 \leq r$ and $r * r \leq r$. In a concurrent semiring these two axioms can equivalently be combined into $1 + r * r \leq r$. The set of all invariants of A is denoted by $I(A)$.

We now give a number of algebraic properties of invariants that are useful in proving the soundness of the rely/guarantee-calculus in Section 13.

Theorem 10.6. *Assume a concurrent monoid A, an $r \in I(A)$ and arbitrary $a, b \in A$.*

1. $a \leq r * a$ *and* $a \leq a * r$.
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. *If A is a CKA then* $r \in I(A) \Leftrightarrow r = r^{\circledast}$.

7. *If A is a CKA then the least invariant comprising a is* $a^{\circledast}$.

Proof.

1. By neutrality of 1 and isotony of $*$ we have $a = 1*a \leq r*a$. The proof of the second inequation is symmetric.

2. This is immediate from $a\,;b \leq a*b$ (3) and transitivity of $\leq$.

3. By Part 1 we have $r \leq r*r$; the converse equation holds by definition and Part 2, respectively.

4. $$\begin{array}{ll} & r\,;(a*b) \\ = & \quad \{\![\text{ by Part 3 }]\!\} \\ & (r*r)\,;(a*b) \\ \leq & \quad \{\![\text{ by (7) }]\!\} \\ & (r\,;a)*(r\,;b)\ . \end{array}$$
   The proof of the second law is symmetric.

5. $$\begin{array}{ll} & r\,;a\,;r \\ \leq & \quad \{\![\text{ by (3) }]\!\} \\ & r*a*r \\ = & \quad \{\![\text{ commutativity of } * \ ]\!\} \\ & r*r*a \\ \leq & \quad \{\![\text{ definition of invariants }]\!\} \\ & r*a\ . \end{array}$$

6. ($\Rightarrow$) By the definition of invariants we have $1 + r*r \leq r$. Hence star induction (10) shows $r^{\circledast} \leq r$. The converse inequation $r \leq r^{\circledast}$ holds by (KA).
   ($\Leftarrow$) follows from (9).

7. By (KA), $a \leq a^{\circledast}$. Moreover, $a^{\circledast}$ is an invariant by Part 6 and (KA) again. Finally, if r is an invariant with $a \leq r$ then $a^{\circledast} \leq r^{\circledast} = r$ by isotony of $^{\circledast}$ and Part 6. □

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

Theorem 10.7. *Assume a CKA A.*

1. *If A is a complete lattice, then so is $(I(A), \leq)$. Its least and greatest elements are* 1 *and* $\top$*, respectively.*

2. *For $r, r' \in I(A)$ 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 operator $*$ on $I(A)$.*

3. *If $r, r' \in I(A)$ have an infimum $r \sqcap r'$ in A then this coincides with the infimum of r and r' in $I(A)$.*

4. *$r*r'$ is the supremum of r and r' in $I(A)$. In particular, $r \leq r'' \wedge r' \leq r'' \Leftrightarrow r*r' \leq r''$.*

5. *Invariants are downward closed: $r*r' \leq r'' \Rightarrow r \leq r''$.*

6. *If A is a complete lattice then $I(A)$ is even closed under arbitrary infima, i.e., for a subset $U \subseteq I(A)$, the infimum $\sqcap U$ taken in A coincides with the infimum of U in $I(A)$.*

Proof.

1. By Theorem 10.6.6 the invariants are exactly the fixpoints of the $^{\circledast}$ operation. Since this operation is isotone, Tarski's theorem shows the completeness claim. Leastness of 1 in $I(A)$ is an axiom. Since $\top$ is the greatest element, we have $1 \leq \top$ and $\top \cdot \top \leq \top$ and hence $\top \in I(A)$.

2. First, $r \leq r' \Rightarrow r * r' \leq r' * r' = r'$ by isotony and Theorem 10.6.3. The reverse inequation $r' \leq r * r'$ holds by Theorem 10.6.1.
   Second, by Theorem 10.6.1, $r \leq r * r'$ 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 10.6.3, $(r \sqcap r') * (r \sqcap r') \leq r * r = r$. 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 $I(A)$ and therefore also the infimum of r and r' in $I(A)$.

4. First, $1 = 1 * 1 \leq r * r'$ and $(r * r') * (r * r') = r * r * r' * r' \leq r * r'$ show that $r * r' \in I(A)$ 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 standard lattice theory.

5. Immediate from Theorem 10.6.1 and transitivity of $\leq$.

6. By standard Kleene algebra, the operation $^{\circledast}$ is a closure operation. Hence, as shown e.g. in [5] its set of fixpoints $I(A)$ is closed under arbitrary infima. □

Next we state two laws about iteration.

Lemma 10.8. *Assume a CKA A and let $r \in I(A)$ be an invariant and $a \in A$ be arbitrary.*

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

Proof.

1. We calculate

$$\begin{array}{ll} & (r * a)^{\circledast} \leq r * a^{\circledast} \\ \Leftarrow & \quad \{\![\text{ star induction (10) }]\!\} \\ & 1 + (r * a) * (r * a^{\circledast}) \leq r * a^{\circledast} \\ \Leftrightarrow & \quad \{\![\text{ join }]\!\} \\ & 1 \leq r * a^{\circledast} \wedge (r * a) * (r * a^{\circledast}) \leq r * a^{\circledast} \,. \end{array}$$

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

$$r * a^{\circledast} = (r * r) * a^{\circledast} \geq (r * r) * (a * a^{\circledast}) = (r * a) * (r * a^{\circledast}) \,.$$

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

$$r * (r * a)^{\circledast} \leq r * r * a^{\circledast} = r * a^{\circledast} \,.$$

   For the reverse inequation we first conclude $a \leq r * a$ from Theorem 10.6.1 and then use isotony of $^{\circledast}$ and $*$. □

The above view of invariants is too special for some circumstances. Therefore we define a more liberal notion of invariants based on the fact that INV is a closure and take Parts 4 and 5 of Theorem 10.3 as the characteristics of abstract invariants, since these properties suffice to prove the results about the rely/guarantee calculus in Section 13 we are after.

Definition 10.9. A *concurrent semiring with invariants* is a structure $(A, +, 0, *, ;, 1, \iota)$ such that $(A, +, 0, *, ;, 1)$ is a concurrent semiring and $\iota : A \to A$ is a closure operator that satisfies, for all $a, b \in A$,

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

A *closure invariant* is an element $a \in A$ with $\iota\, a = a$.

Lemma 10.10. *By the definition* $\iota\, a =_{df} a^{\circledast}$ *every CKA becomes a concurrent semiring with invariants.*

Proof. By standard Kleene algebra, $^{\circledast}$ is a closure operator with $1 \leq a^{\circledast}$. The remaining axiom is shown by star induction (10), $a, b \leq a + b$ and isotony as follows:

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

Again it is clear that the closure invariants form a complete lattice with properties analogous to those of Corollary 10.4. Moreover, one has the usual Galois connection for closures (cf. [12]):

$$a \leq \iota\, b \;\Leftrightarrow\; \iota\, a \leq \iota\, b \;. \tag{11}$$

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

Theorem 10.11. *Let A be a concurrent semiring with invariants and $\circ$ be an isotone binary operator on A that has 1 as neutral element and satisfies $\forall\, a, b : \iota\,(a \circ b) \;\subseteq\; \iota\,(a + b)$. Then, for closure 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 invariance 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 $\circ$. □

Example 10.12. Consider a concurrent semiring A with invariants. Setting $\circ = ;$ we obtain by (3) and isotony of ι that $\iota\,(a\,;b) \leq \iota\,(a * b)$. Since, in turn, $\iota\,(a * b) \leq \iota\,(a + b)$ by Def. 10.9, Theorem 10.11 shows $r\,;r = r$ for all closure invariants r. □

11. Characterising Dependence

Invariants are of central importance for the rely/guarantee calculus in Sections 13 and 14. Their most fundamental property is the star distribution rule, the inequational form of which has been shown in Theorem 10.6.4. We will now characterise the dependence relations for which this rule and another related one are valid.

Theorem 11.1. *Let $R = \mathcal{P}(E)$ be a power invariant in $PR(EV)$ and assume that $\to$ is transitive.*

1. *If $\to$ is acyclic and $e \in EV$ then*
$$R * [e] \;\subseteq\; R\,;[e]\,;R \;,$$
where $[e]$ is again the single-event program $\{\{e\}\}$ (cf. Section 2).
2. *If $\to$ is transitive then for all $P, Q \in PR(EV)$ we have*
$$R * (P\,;Q) \;\subseteq\; (R * P)\,;(R * Q) \;.$$

This means that the two properties of Theorem 11.1 hold if $\to$ is a strict-order.

To prove Theorem 11.1, we first show an auxiliary lemma about the dependence relation. To formulate it, we need an additional notion.

Definition 11.2. Remember the function *dep* from Def. 7.1 that, for a trace *tp* yields the set of events on which *tp*-events depend. Given traces tp, tr with $tp \cap tr = \emptyset$, we define

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

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

Lemma 11.3. *Assume again transitivity of* $\rightarrow$ *and consider arbitrary traces tp and tq.*

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

Let now tp and 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
$$\{e\} \cap dep(\{e\}) = \emptyset \Leftrightarrow e \notin dep(\{e\}) \Leftrightarrow \neg(e \rightarrow e) \Leftrightarrow \mathsf{TRUE} ,$$
since $\rightarrow$ is transitive and acyclic. □

Proof of Theorem 11.1.
We first note that power invariants $R = \mathcal{P}(E)$ satisfy a stronger form of downward closure than the one stated in Theorem 10.7.5, 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 $e \notin E$:

$$\begin{array}{ll} & R * [e] \\ = & \quad \{\![\text{ definition of } *]\!\} \\ & \bigcup_{tr \in R} \{tr * \{e\}\} \\ \subseteq & \quad \{\![\text{ by Lemma 11.3.8 and downward closure of } R]\!\} \\ & \bigcup_{tr' \in R} \bigcup_{tr'' \in R} \{tr' \,;\, \{e\} \,;\, tr''\} \\ = & \quad \{\![\text{ definition of } ;]\!\} \\ & R \,;\, [e] \,;\, R \;. \end{array}$$

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 11.3.7 $dep(tr') \cap tr'' = \emptyset$.
   (b) By Lemma 11.3.5 $dep(tr') \cap tq = \emptyset$.
   (c) By Lemma 11.3.4 $dep(tp) \cap tr'' = \emptyset$.
   By definition of tr', tr'', associativity and commutativity of union and (a),(b),(c) as well as $dep(tp) \cap tq = \emptyset$ we have

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

□

Next we want to see that in a sense also the reverse implications of Theorem 11.1 hold. To formulate this we need a further notion.

Definition 11.4. 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 acyclicity 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 11.5. *Let $[e]$ be again the single-event program $\{\{e\}\}$.*
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 e, f, g with $f \rightarrow g$ and $g \rightarrow e$ but $f \not\rightarrow e$. This implies $f \neq g$ and $g \neq e$. Assume now $e \neq f$ and set $P =_{df} [e]$, $Q =_{df} [f]$ and $R =_{df} [\,] \cup [g]$. Then $P \,;\, Q = [e, f]$ and $R * (P \,;\, Q) = [e, f] \cup [g, e, f]$. Moreover, $R * P = [e] \cup [g, e]$ and $R * Q = [f] \cup [g, f]$, hence $(R * P) \,;\, (R * Q) = [e, f]$ contradicting the assumed property. Therefore we must have $e \leftarrow f$. □

We abstract this as follows.

Definition 11.6. A concurrent semiring A with invariants is $*$-*distributive* if all closure invariants r and all $a, b \in A$ satisfy

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

We still have to prove Part 1 of Theorem 11.5. Rather than doing this directly, we investigate a slightly more general property which is equivalent to an interesting property of traces that are more general than single-event ones.

Definition 11.7. A trace tp is *convex* if for all events $e, f \in tp$ and arbitrary event g we have

$$e \rightarrow^+ g \rightarrow^+ f \ \Rightarrow \ g \in tp \ .$$

A convex trace can be considered as "closed" under dependence.
Remember again the function dep from Def. 7.1. Then we have

Lemma 11.8. 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 $g \notin tp$ we have
$$tp \cap dep(\{g\}) = \emptyset \ \ \vee \ \ \{g\} \cap dep(tp) = \emptyset \ .$$
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 $g \notin tp$. The premise $e \rightarrow^+ g$ implies $e \in tp \cap dep(\{g\})$ while $g \rightarrow^+ f$ implies $g \in \{g\} \cap dep(tp)$. In particular, both sets are non-empty, contradicting Part 1. □

We now establish a first connection between convexity and weak acyclicity.

Lemma 11.9. *The relation $\rightarrow$ is weakly acyclic iff all singleton traces $\{e\}$ are convex.*

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

We now want to show that also the reverse of Lemma 11.8 holds.

Lemma 11.10. 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 11.3 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 $e \in tp \cap dep(tr')$, say $e \rightarrow^+ g$ for some $g \in tr'$. By definition of tr' there is an $f \in tp$ with $g \rightarrow^+ f$. Since tp is assumed to be convex, this implies $g \in tp$, a contradiction to $g \in tr'$ and $tp \cap tr' = \emptyset$. □

Next, we consider general programs.

Definition 11.11. A program is *convex* if all its traces are.

Lemma 11.12. *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 11.10.
($\Leftarrow$) 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 $tr' \not\leq tp' \wedge tp' \not\leq tr'' \wedge tr' \not\leq tr''$ such that $tp \cup tr = tr' \cup tp' \cup tr''$. But, by disjointness, this implies $tp' = tp$ and we are done. □

These results motivate the following abstraction.

Definition 11.13. An element a of a concurrent semiring 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 10.11) 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.

12. Rely/Guarantee Algebras

As before, we abstract the results of the previous section into general algebraic terms. The terminology stems from the applications in the following section.

Definition 12.1. A *rely/guarantee semiring* is a pair (A, I) such that A is a concurrent semiring with invariants and $I \subseteq \iota(A)$ is a sublattice of closure invariants. In particular, for all $r, r' \in I$ their meet $r \sqcap r' \in I$ is assumed to exist. Moreover, we assume $1 \in I$ and $r * r' \in I$ whenever $r, r' \in I$. Finally, all $r \in I$ and $a, b \in A$ have to satisfy $r * (a\,;b) \leq (r * a)\,;(r * b)$.

A *rely/guarantee CKA (quantale)* is a rely/guarantee semiring that is a CKA (quantale).

The restriction that I be a sublattice of $I(A)$ is motivated by the rely/guarantee-calculus in Section 13 below. Using Mace4 it can be shown that the axiomatisation is irredundant.

Together with the exchange law (7), $*$-idempotence of r and commutativity of $*$ the definition implies

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

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

Using Theorem 11.1 we can prove

Lemma 12.2. *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 semiring.*

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 can now explain why it was necessary to introduce the subset I of closure invariants in a rely/guarantee semiring. 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 12.3. Assume an event set EV with three different events $e, f, g \in EV$ and a transitive dependence $\rightarrow$ with $e \rightarrow g \rightarrow f$. 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 $R * [r] = [r] \cup [e, f, g]$, but $R\,;[g]\,;R = [g]$. □

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

Lemma 12.4. *Assume a rely/guarantee quantale (A, I), an invariant $r \in I$, an arbitrary $a \in A$, and $\circ \in \{;, *\}$.*

1. $r * a^{\circledcirc} = (r * a)^{\circledcirc} \circ r = r \circ (r * a)^{\circledcirc}$.
2. $(r * a)^+ = r * a^+$, *where* $a^+ =_{df} a \circ a^{\circledcirc}$.

For the proof we use the following fusion rule for least fixpoints that is valid in quantales (cf. [2]). Let $f, g, h : A \to A$ be isotone functions. Then

$$\frac{\begin{array}{c} f \text{ continuous and strict} \\ \forall\, x : f(g(x)) = h(f(x)) \end{array}}{f(\mu g) = \mu h} \tag{12}$$

Proof of Lemma 12.4

1. For the first equation we use the fusion law (12) 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 $h(x) =_{df} r * a + (r * a) \circ x$. □

13. Jones's Rely/Guarantee-Calculus

In [25] Jones has presented a calculus that considers properties of the environment on which a program wants to rely and the ones it does, in turn, guarantee for the environment. We now provide an abstract algebraic treatment of this calculus.

The original motivation for discussing invariants was that they should allow guaranteeing that a program only uses events from a given admissible set. To this end we base our treatment on a concurrent monoid with invariants and define a *guarantee relation*, slightly more liberally than [19], by

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

meaning that a guarantees the closure invariant of b. Since ι as a closure is extensive, isotone and idempotent, the right hand side is equivalent to $a \leq \iota\, b$. If b is an invariant, i.e., $b = \iota\, b$, we obtain by (11)

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

Example 13.1. With the notation $P_u =_{df} [au]$ for $u \in \{x, y, z\}$ of Example 2.3 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 13.2. *Assume a rely/guarantee semiring* (A, I).

1. 1 guar g.
2. *If* g, g' *are closure invariants and* $\circ$ *is an isotone binary operator satisfying* $\forall\, a, b : \iota\,(a \circ b) \leq \iota\,(a + b)$ *then*
$$b \text{ guar } g \;\wedge\; b' \text{ guar } g' \;\Rightarrow\; (b \circ b') \text{ guar } (g + g') \;.$$
3. *If* A *is a rely/guarantee CKA then for* $\circ \in \{*, ;\}$ *we have* a guar $g \Leftrightarrow a^{\circledcirc}$ guar g.
4. *For the concrete case of programs,* $[e]$ guar $G \Leftrightarrow e \in \lceil G \rceil$.

Proof.

1. Immediate from the axioms and the above remark on guar.
2. $$\begin{array}{ll} & b \text{ guar } g \wedge b' \text{ guar } g' \\ \Leftrightarrow & \quad \{\![\text{ above remark on guar }]\!\} \\ & \iota\, b \leq g \wedge \iota\, b' \leq g' \\ \Rightarrow & \quad \{\![\text{ isotony of } + \,]\!\} \\ & \iota\, b + \iota\, b' \leq g + g' \\ \Rightarrow & \quad \{\![\text{ subdistributivity of } \iota \,]\!\} \\ & \iota\,(b + b') \leq g + g' \\ \Rightarrow & \quad \{\![\text{ assumption about } \circ \,]\!\} \\ & \iota\,(b \circ b') \leq g + g' \\ \Leftrightarrow & \quad \{\![\text{ extensivity of } \iota \,]\!\} \\ & \iota\,(b \circ b') \leq \iota\,(g + g') \\ \Leftrightarrow & \quad \{\![\text{ definition }]\!\} \\ & (b \circ b') \;\; \text{guar} \;\; (g + g') \;. \end{array}$$
3. Using the assumption, invariance of g and star induction, we calculate
$$a \leq g \;\Rightarrow\; a \circ g \leq g \circ g = g \;\Rightarrow\; 1 + a \circ g \leq g \;\Rightarrow\; a^{\circledcirc} \leq g \;.$$
The reverse implication follows by $a \leq a^{\circledcirc}$.
4. By the definitions and the Galois connection for $\lceil \_ \rceil$,
$$[e] \text{ guar } G \Leftrightarrow \mathsf{INV}([e]) \subseteq \mathsf{INV}(G) \Leftrightarrow \{e\} \subseteq \mathsf{INV}(G) \Leftrightarrow e \in \lceil G \rceil \;.$$ □

Using the guarantee relation, Jones quintuples can be defined as in [19]:

Definition 13.3.
$$a \;\; r \;\{b\}\; s \;\; g \;\; \Leftrightarrow_{df} \;\; a \;\{r * b\}\; s \;\; \wedge \;\; b \text{ guar } g \;,$$
where r and g are invariants, and Hoare triples are again interpreted in terms of sequential composition $;$.

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

Theorem 13.4. *Consider a rely/guarantee semiring* (A, I). *For invariants* $r, r', g, g' \in I$ *and elements* a, $a', b, b', c, c' \in A$ *such that the meets* $a \sqcap a'$ *and* $c \sqcap c'$ *exist,*
$$\begin{array}{l} a \;\; r \;\{b\}\; c \;\; g \;\; \wedge \;\; a' \;\; r' \;\{b'\}\; c' \;\; g' \;\; \wedge \;\; g' \text{ guar } r \;\; \wedge \; g \text{ guar } r' \;\; \Rightarrow \\ \qquad (a \sqcap a') \;\; (r \sqcap r') \;\{b * b'\}\; (c \sqcap c') \;\; (g * g') \;. \end{array}$$

Proof. The guarantee part is covered by Theorem 13.2.2. For the remainder we note that the assumptions b' guar g' guar r and b guar g guar r' imply, by transitivity of guar, that b' guar $r \wedge b$ guar r', and calculate

$$
\begin{array}{ll}
& (a \sqcap a') \,;\, ((r \sqcap r') * (b * b')) \leq c \sqcap c' \\
\Leftrightarrow & \quad \{\![\text{ characterisation of intersection }]\!\} \\
& (a \sqcap a') \,;\, ((r \sqcap r') * (b * b')) \leq c \;\wedge\; (a \sqcap a') \,;\, ((r \sqcap r') * (b * b')) \leq c' \\
\Leftarrow & \quad \{\![\text{ intersection, isotony }]\!\} \\
& a \,;\, (r * (b * b')) \leq c \wedge a' \,;\, (r' * (b * b')) \leq c' \\
\Leftarrow & \quad \{\![\; b' \text{ guar } r \wedge b \text{ guar } r' \text{ and isotony }]\!\} \\
& a \,;\, (r * (b * r)) \leq c \wedge a' \,;\, (r' * (r' * b')) \leq c' \\
\Leftarrow & \quad \{\![\text{ associativity and commutativity of } * \,]\!\} \\
& a \,;\, ((r * r) * b) \leq c \wedge a' \,;\, ((r' * r') * b') \leq c' \\
\Leftrightarrow & \quad \{\![\text{ idempotence of } * \text{ on invariants (Theorem 10.6.3) }]\!\} \\
& a \,;\, (r * b) \leq c \wedge a' \,;\, (r' * b) \leq c' \\
\Leftarrow & \quad \{\![\text{ definition of quadruples and assumption }]\!\} \\
& \mathsf{TRUE} \,.
\end{array}
$$

□

Note that $r \sqcap r'$ and $g * g'$ are again invariants by Definition 12.1. For sequential composition we have

Theorem 13.5. *Assume a rely/guarantee semiring (A, I). Then for invariants $r, r', g, g' \in I$ and arbitrary a, b, b', c, c',*

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

Proof. The guarantee part is again covered by Theorem 13.2.2. Specialising b, d, b', c, e in Lemma 9.3 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 Definition 12.1 implies $r \sqcap r' \in I$ again, this follows by $*$-distributivity and isotony of $*$ and $;$. □

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

Theorem 13.6. *Assume a rely/guarantee semiring (A, I). Then for invariants $r, g \in I$ and arbitrary $s \in A$,*

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

Proof.

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
   $a \; r \; \{b + b'\} \; s \; g \Leftrightarrow a \,;\, (r * (b + b')) \leq s \wedge b + b' \leq g \Leftrightarrow$
   $a \,;\, (r * b) + a \,;\, (r * b') \leq s \wedge b \leq g \wedge b' \leq g \Leftrightarrow a \; r \; \{b\} \; s \; g \wedge a \; r \; \{b'\} \; s \; g$.
3. This is immediate from Definition 11.13 and the remark following it. □

Finally we give rely/guarantee rules for iteration.

Theorem 13.7. *Assume a rely/guarantee CKA (A, I) and let $^{\circledcirc}$ be finite iteration w.r.t. $\circ \in \{*, ;\}$. Then for invariants $r, g \in I$ and arbitrary elements $a, b \in A$,*

$$\begin{array}{rcl} a\ r\ \{b\}\ a\ g & \Rightarrow & a\ r\ \{b^+\}\ a\ g\ , \\ a\ \{r\}\ a \wedge a\ r\ \{b\}\ a\ g & \Rightarrow & a\ r\ \{b^{\circledcirc}\}\ a\ g\ . \end{array}$$

Proof. Recall from Definition 8.1 that in any Kleene algebra $b^+ = b \circ b^*$ and hence $b^* = 1 + b^+$. Thus the first law is immediate from Lemma 12.4.2, Lemma 9.2.8 and Theorem 13.2.3. The second one follows from the first one by $b^* = 1 + b^+$ and the choice and skip rules. □

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

Example 13.8. We consider again the programs $P_u = [au]$ and invariants $G_u = P_u \cup \mathsf{skip}$ $(u \in \{x, y\})$ from Example 13.1. Moreover, we assume an event av with $v \neq x, y$, $ax \not\to av$ and $ay \not\to av$ and set $P_v =_{df} [av]$. We will show that

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

holds. In particular, the concurrent 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 9.2.4 we obtain

$$P_v\ \{R_x\ ;\ P_x\ ;\ R_x\}\ S_x\ .$$

By Theorem 13.6.3 we obtain $P_v\ R_y\ \{P_x\}\ S_x\ G_x$ and, similarly, $P_v\ R_x\ \{P_y\}\ S_y\ G_y$. Now the claim follows from the clauses (a),(b),(c),(d) and Theorem 13.4. □

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.

14. A Simplified Rely/Guarantee-Calculus

For certain purposes, the following type of quadruples with an invariant r works just as well as the Jones quintuples:

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

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 containing b.

Note that the quadruples can be retrieved as special cases of quintuples:

$$a\ r\ \{b\}\ s \Leftrightarrow a\ r\ \{b\}\ s\ b\ . \tag{13}$$

We give the simplified versions of the original rely/guarantee-properties; the proofs result in a straightforward way from the ones above by embedding (13). Throughout this section we assume a rely/guarantee semiring (A, I).

For concurrent composition we obtain

Theorem 14.1. *For invariants r, r' and elements $a, a', b, b', c, c' \in A$ such that the meets $a \sqcap a'$ and $c \sqcap c'$ exist,*

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

For sequential composition one has

Theorem 14.2. *For invariants r, r',*

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

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

Theorem 14.3.

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 [21]; therefore we give a quick proof for the abstract form here:

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

15. Event-Based Algebras

The definition of a concurrent semiring does not mention the dependence relation any more. In this section we show that in particular concurrent algebras it can be recovered from the ; and $*$ operators. To this end, we now give algebraic characterisations of traces and events.

Throughout this section we assume a concurrent semiring A with $1 \neq 0$. A *subatom* is an element a such that $b \leq a \ \Rightarrow\ b = 0 \vee b = a$. A subatom different from 0 is called an *atom* (e.g. [5]).

Definition 15.1. An element $t \in A$ is called a *trace* if it is a subatom and join-prime, i.e., if

$$\forall\, a \in A : a \leq t \ \Rightarrow\ a = 0 \vee a = t\ , \\ \forall\, T \subseteq A : T \neq \emptyset \wedge t \leq \sqcup\, T \ \Rightarrow\ \exists\, a \in T : t \leq a\ .$$

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

$$TR(b)\ =_{df}\ \{a \in TR(A) \,|\, a \leq b\}\ .$$

By this definition, 0 is a trace, which saves a number of case distinctions. It is immediate that every trace a is $+$-irreducible, i.e.,

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

Moreover, if a is a trace and $b \leq a$ then b is a trace, too. In particular, if $a * b$ is a trace then by (3) also $a\,;b$ is a trace.

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

Definition 15.2. In a concurrent bimomoid A we define a relation $\sqsubseteq$ by

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

To investigate its properties we need

Definition 15.3. A subset $E \subseteq A$ 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 15.4.

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

Assume now that $E \subseteq A$ 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 from the definition of $a \sqsubseteq b$, since 0 is an annihilator for $*$. Otherwise suppose $b = a * c$ 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 the relation $\sqsubseteq$ 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 $\bigsqcup\!\!\!\!\!\!* \ D$.

Lemma 15.5. *If $0 \in D \subseteq E$ then $0 = \bigsqcup\!\!\!\!\!\!* \ D$.*

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

Definition 15.6. 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

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

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 15.7. A concurrent semiring A 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 $a \in A$ we have $a = \bigsqcup TR(a)$.

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

$$EV(t) =_{df} \{e \in EV(A) \,|\, e \sqsubseteq t\} \;.$$

(d) The set $TR(A)$ of traces is a complete lattice w.r.t. $\sqsubseteq$ and every trace is the supremum of its events, i.e., for all $t \in TR(A)$ we have $t = \bigsqcup\!\!\!\!\!* \; EV(t)$.

(e) For all events e we have $e * e = 0$ and hence $e \,;\, e = 0$.

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

Hence our concrete model of programs forms an event-based concurrent semiring. Event-based concurrent semirings are quite similar to the feature algebras developed in [24] for the description of product families.

The definition of an event-based concurrent semiring A immediately yields

Lemma 15.8.

1. $EV(0) = EV(A)$.
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 $a * b = \bigsqcup\!\!\!\!\!* \; \{a, b\}$.*

16. Abstract Dependence

In this section we define an abstract counterpart to the dependence relation used in our concrete trace model.

Definition 16.1. 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 10.11:

Lemma 16.2.

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:

$$\{tp\} \not\leftarrow \{tq\} \quad \Leftrightarrow \quad \forall\, e \in tp, f \in tq : \neg(e \leftarrow f)\ .$$

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

Definition 16.3. In an event-based concurrent semiring we define the dependence relation between events e, f by

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

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

Lemma 16.4. *Consider traces tp, tq of an event-based concurrent semiring that respects dependence.*

1. *If $e \to f$ for some $e \in EV(tp)$ and $f \in EV(tq)$ then $tp\,;tq = 0$.*
2. *If $tp * tq \neq 0$ then*
$$tp \not\leftarrow tq \quad \Leftrightarrow \quad \forall\, e \in EV(tp), f \in EV(tq) : e \not\leftarrow f\ .$$

Proof.

1. By additivity of ; we have $tp\,;tq = \bigsqcup\!\!\!\!\ast\ \{u\,;v \,|\, u \in EV(tp), v \in EV(tq)\}$ and the claim follows from Lemma 15.5.
2. ($\Leftarrow$) Immediate from event-basedness and additivity of $*$ and ;.
($\Rightarrow$) By Part 1 we have $e\,;f \neq 0$ for all $e \in EV(tp)$ and $f \in EV(tq)$. Since $TR(A)$ is assumed to be well behaved, also $e * f$ is a trace, and from $e\,;f \leq e * f$ it follows that $e\,;f = e * f$. □

With these prerequisites it is now possible to completely replay the proof of Theorem 11.1 in the abstract setting of event-based concurrent semirings; we omit the details.

17. Related Work

Although our basic model and its algebraic abstraction reflect a non-interleaving view of concurrency, we try to set up a connection with familiar process algebras such as ACP [4], CCS [31], CSP [17], mCRL2 [15] and the π-calculus [39]. It is not easy to relate their operators to those of CKA. The closest analogies seem to be the following ones.

CKA operator	corresponding operator
$+$	non-deterministic choice in CSP
$*$	concurrent composition | in ACP, π-calculus and CCS, and !! in CSP
$\|$	interleaving ||| 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. In ACP, for instance, we only have the law $(a + b).c = a.c + b.c$ but not $a.(b + c) = a.b + a.c$.

As the observation after Theorem 11.1 shows, our basic model falls into the class of partial-order models for true concurrency. Of the numerous works in that area we discuss some approaches that have explicit

operators for composition related to our $*$ and $;$. Whereas we assume that our dependence relation is fixed a priori, in the pomset approach [14, 13, 37] it is constructed by the composition operators. The operators there are sequential and concurrent composition; there are no choice and iteration, though. Moreover, no laws are given for the operators. In Winskel's event structures [40] there are choice (sum) and concurrent composition, but no sequential composition and iteration. Again, there are no interrelating laws. Another difference to our approach is that the "traces" are required to observe certain closure conditions.

Among the axiomatic approaches to partial order semantics we mention the following ones. Boudol and Castellani [7] present the notion of trioids, which are algebras offering the operators of choice, sequential and concurrent composition. However, there are no interrelating laws and no iteration. Chothia and Kleijn07 [8] use a double semiring with choice, sequential and concurrent composition, but again no interrelating laws and no iteration. The application is to model quality of service, not program semantics.

The approach closest in spirit to ours is that of Prisacariu's synchronous Kleene algebras (SKA) [36]. The main differences are the following. SKAs are restricted to a finite alphabet of actions and hence have a complete and even decidable equational theory. There is only a restricted form of concurrent composition, and the exchange law is equational rather than inequational. Iteration is present but not used in an essential way. Nevertheless, Prisacariu's paper is the only of the mentioned ones that explicitly deals with Hoare logic. It does so using the approach of Kleene algebras with tests [27]. This is not feasible in our basic model, since tests are required to be below the element 1, and 0 and 1 are the only such elements. Note, however, that Mace4 [30] quickly shows that this is not a consequence of the CKA axioms but holds only for the particular model.

18. 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. Some other interesting models of CKA than the trace model have been developed; they will be presented in other papers.

Further work is also needed to see how far the trace model and its algebra can be applied to other familiar process algebras and programming paradigms. We suspect that the easiest candidates will be the π-calculus and Dijkstra's imperative language of weakest preconditions; and this could show a convenient way of combining the two calculi. The connection with separation logic and separation algebra could be fruitful. Other challenges will be a treatment of external choice, atomicity refinement, transactions, and exceptions. We hope that these extensions can be made incrementally, and then combined automatically, without invalidating earlier developments or conflicting with each other.

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

[1] R. Back, J. von Wright: Refinement calculus — a systematic introduction. Springer 1998
[2] R. Backhouse et al: Fixed point calculus. Information Processing Letters 53, 131–136 (1995)
[3] J. Benabou: Introduction to bicategories. In: Reports of the Midwest Category Seminar. Lecture Notes in Math. 47. Springer 1967, 1Â-77
[4] J.A. Bergstra, I. Bethke, A. Ponse: Process algebra with iteration and nesting. The Computer Journal 37(4), 243–258 (1994)
[5] Birkhoff, G. Lattice Theory, 3rd ed. Amer. Math. Soc. 1967
[6] S. Bistarelli, U. Montanari, F. Rossi: Semiring-based constraint satisfaction and optimization. J. ACM, 44(2):201-236 (1997)
[7] G. Boudol, I. Castellani: On the semantics of concurrency: partial orders and transition systems. In: Ehrig, H., Levi, G., Montanari, U. (eds.) CAAP 1987 and TAPSOFT 1987. LNCS 249. Springer 1987, 123–137
[8] T. Chothia, J. Kleijn: Q-Automata: modelling the resource usage of concurrent components. Electr. Notes Theor. Comput. Sci. 175(2): 153–167 (2007)

[9] E. Cohen: Separation and reduction. In: R. Backhouse, J. Oliveira (eds.): Mathematics of Program Construction (MPC'00). LNCS 1837. Springer 2000, 45–59
[10] J. Conway: Regular algebra and finite machines. Chapman&Hall 1971
[11] J. Desharnais, B. Möller, G. Struth: Kleene algebra with domain. Trans. Computational Logic 7, 798–833 (2006)
[12] 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)
[13] J. Gischer: Partial orders and the axiomatic theory of shuffle. PhD thesis, Stanford University (1984)
[14] Grabowski, J.: On partial languages. Fundamenta Informaticae 4(1), 427–498 (1981)
[15] J. Groote, A. Mathijssen, M. van Weerdenburg, Y. Usenko. From μCRL to mCRL2: motivation and outline. In: Proc. Workshop Essays on Algebraic Process Calculi (APC 25). ENTCS 162, 191–196 (2006)
[16] C.A.R. Hoare: An axiomatic basis for computer programming. Commun. ACM. 12, 576–585 (1969)
[17] C.A.R. Hoare: Communicating sequential processes. Prentice Hall 1985
[18] C.A.R. Hoare: Unifying models of data flow. Lecture Notes, International Summer School Marktoberdorf, August 2010
[19] C.A.R. Hoare, I. Wehrman, P. O'Hearn: Graphical models of separation logic. In M. Broy et al. (eds.): Engineering Methods and Tools for Software Safety and Security. IOS Press 2009, 177–202
[20] C.A.R. Hoare: Unifying models of execution history. Manuscript, June 2009
[21] C.A.R. Hoare, B. Möller, G. Struth, I. Wehrman: Concurrent Kleene algebra. In M. Bravetti, G. Zavattaro (eds.): Concurreny Theory (CONCUR 2009), LNCS 5710, Springer 2009, 399–414
[22] C.A.R. Hoare, B. Möller, G. Struth, I. Wehrman: Foundations of concurrent Kleene algebra. In R. Berghammer, A. M. Jaoua, B. Möller (eds.): Relations and Kleene Algebra in Computer Science, LNCS 5827, Springer 2009, 166–186
[23] C.A.R. Hoare, B. Möller, G. Struth, I. Wehrman: Concurrent Kleene algebra and its foundations. University of Sheffield, Department of Computer Science, Research Report CS-10-04, August 2010. Accessible via `http://www.dcs.shef.ac.uk/~georg/ka/`
[24] 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
[25] C. Jones: Development methods for computer programs including a notion of interference. PhD Thesis, University of Oxford. Programming Research Group, Technical Monograph 25, 1981
[26] D. Kozen: A completeness theorem for Kleene algebras and the algebra of regular events. Information and Computation 110, 366–390 (1994)
[27] D. Kozen: Kleene algebra with tests. Trans. Programming Languages and Systems 19, 427–443 (1997)
[28] P. Lescanne: Modèles non déterministes de types abstraits. R.A.I.R.O. Informatique théorique 16, 225-244 (1982)
[29] S. Mac Lane: Categories for the working mathematician (2nd ed.). Springer 1998
[30] W. McCune: Prover9 and Mace4. `http://www.prover9.org/` (accessed March 1, 2009)
[31] R. Milner: A Calculus of communicating systems. LNCS 92. Springer 1980
[32] J. Misra: Axioms for memory access in asynchronous hardware systems. ACM Trans. Program. Lang. Syst. 8, 142–153 (1986)
[33] C. Morgan: Programming from specifications. Prentice Hall 1990
[34] C. Mulvey: &. Rendiconti del Circolo Matematico di Palermo 12, 99–104 (1986)
[35] P. O'Hearn: Resources, concurrency, and local reasoning. Theor. Comput. Sci. 375, 271–307 (2007)
[36] Cristian Prisacariu: Synchronous Kleene algebra. J. Log. Algebr. Program. 79(7): 608-635 (2010)
[37] Pratt, V.R.: Modelling concurrency with partial orders. Journal of Parallel Programming 15(1) (1986)
[38] K. Rosenthal: Quantales and their applications. Pitman Research Notes in Math. No. 234 Longman Scientific and Technical 1990
[39] D. Sangiorgi, D. Walker: The π-calculus — A theory of mobile processes. Cambridge University Press 2001
[40] G. Winskel: Event structures. In: W. Brauer, W. Reisig, G. Rozenberg (eds.) Advances in Petri Nets 1986. LNCS 255. Springer 1987, 325–392

Appendix A. Axiom Systems

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

1. An *idempotent semiring* is a structure $(A, +, \cdot, 0, 1)$ such that $(A, +, 0)$ is a commutative monoid with idempotent addition, i.e., $a + a = a$ for all $a \in A$, $(A, \cdot, 1)$ is a monoid, multiplication distributes over addition, i.e., for all $a, b, c \in A$,

$$a \cdot (b + c) = a \cdot b + a \cdot c \qquad \text{and} \qquad (a + b) \cdot c = a \cdot c + b \cdot c,$$

and 0 is a left and right annihilator for multiplication, i.e., for all $a \in A$,

$$a \cdot 0 = 0 = 0 \cdot a \ .$$

2. Every idempotent semiring is partially ordered by

$$a \leq b \Leftrightarrow_{df} a + b = b.$$

Then $+$ and $\cdot$ are isotone w.r.t. $\leq$ and 0 is the least element. Moreover, $a + b$ is the supremum of $a, b \in A$.

3. A idempotent semiring is called a *quantale* [34] or *standard Kleene algebra* [10] if $\leq$ induces a complete lattice and multiplication distributes over arbitrary suprema. The infimum and the supremum of a subset $B \subseteq A$ are denoted by $\sqcap B$ and $\sqcup B$, respectively. Their binary variants are $a \sqcap b$ and $a \sqcup b$ (the latter coinciding with $a + b$).

4. An *ordered monoid* is a structure $(A, \cdot, 1, \leq)$ such that $(A, \cdot, 1)$ is a monoid, A is partially ordered by $\leq$ and $\cdot$ is isotone in both arguments.

5. A *concurrent monoid* is a structure $(A, *, ;, 1, \leq)$ such that $(A, *, 1, \leq)$ and $(A, ; 1, \leq)$ are ordered monoids and the following axioms hold:

$$\begin{aligned} a * b &= b * a, \\ (a * b)\,;(c * d) &\leq (a\,;c) * (b\,;d). \end{aligned}$$

6. A *concurrent semiring* is a structure $(A, +, *, ;, 0, 1)$ such that $(A, +, *, 0, 1)$ and $(A, +, ;, 0, 1)$ are idempotent semirings and $(A, *, ;, \leq)$ is a concurrent semigroup, where $\leq$ is the natural semiring order.

7. A concurrent semiring $(A, +, *, ;, 0, 1)$ is called a *concurrent quantale* if $(A, +, *, 0, 1)$ and $(A, +, ;, 0, 1)$ are quantales.

8. A *Kleene algebra* [26] is a structure $(A, +, \cdot, ^*, 0, 1)$ such that $(A, +, \cdot, 0, 1)$ is an idempotent semiring and the star operator * satisfies the unfold and induction laws

$$1 + a \cdot a^* \leq a^*, \qquad 1 + a^* \cdot a \leq a^*, \tag{A.1}$$

$$c + a \cdot b \leq b \Rightarrow a^* \cdot c \leq b, \qquad c + b \cdot a \leq b \Rightarrow c \cdot a^* \leq b. \tag{A.2}$$

9. A *concurrent Kleene algebra (CKA)* is a structure $(A, +, *, ;, \circledast, \odot, 0, 1)$ such that $(A, +, *, ;, 0, 1)$ is a *concurrent semiring* and $(A, +, *, \circledast, 0, 1)$ and $(A, +, ;, \odot, 0, 1)$ are Kleene algebras.

10. An *invariant* in a concurrent semiring A is an element r satisfying $1 \leq r$ and $r * r \leq r$, equivalently, $1 + r * r \leq r$. The set of all invariants of A is denoted by $I(A)$.

11. A *concurrent semiring with invariants* is a structure $(A, +, 0, *, ;, 1, \iota)$ such that $(A, +, 0, *, ;, 1)$ is a concurrent semiring and $\iota : A \to A$ is a closure operator that satisfies, for all $a, b \in A$,

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

A *closure invariant* is an element $a \in A$ with $\iota\, a = a$.

12. A concurrent semiring A with invariants is $*$-*distributive* if all closure invariants r and all $a, b \in A$ satisfy

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

13. A *rely/guarantee semiring* is a pair (A, I) such that A is a concurrent semiring with invariants and $I \subseteq \iota\,(A)$ is a sublattice of closure invariants with $1 \in I$ and $r * r' \in I$. In particular, for all $r, r' \in I$ their meet $r \sqcap r' \in I$ is assumed to exist. Moreover, all $r \in I$ and $a, b \in A$ have to satisfy $r*(a;b) \leq (r*a);(r*b)$. A *rely/guarantee CKA (quantale)* is a rely/guarantee semiring that is a CKA (quantale).

Appendix B. Sample Input File for Automated Theorem Proving

As a sample input file for Prover9 we show the one for proving some of the laws about Hoare triples from Section 9. 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 no more than one goal in form of a Horn formula, 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/`.

```
op(500, infix, "+").
op(450, infix, "*").
op(450, infix, ";").
formulas(assumptions). % concurrent semiring
     x+(y+z)=(x+y)+z.
     x+y=y+x.
     x+0=x.
     x+x=x.
     x<=y <-> x+y=y.
     x*(y*z)=(x*y)*z.
     x*y=y*x.
     x*1=x.
     x*(y+z)=x*y+x*z.
     x*0=0.
     x;(y;z)=(x;y);z.
     x;1=x.
     1;x=x.
     x;(y+z)=x;y+x;z.
     (x+y);z=x;z+y;z.
     0;x=0.
     x;0=0.
     (w*x);(y*z)<= (w;y)*(x;z).
% concurrent Kleene algebra
     1+x*s1(x)=s1(x).
     1+s1(x)*x=s1(x).
     z+x*y<=y -> s1(x)*z<=y.
     z+y*x<=y -> z*s1(x)<=y.
     1+x;s2(x)=s2(x).
     1+s2(x);x=s2(x).
     z+x;y<=y -> s2(x);z<=y.
     z+y;x<=y -> z;s2(x)<=y.
end_of_list.

formulas(goals).
%     s1(x)=x -> y<=x;y & y<= y;x & y<=x*y & y<=x*y.
%     s1(x)=x -> x;x<=x.
%     s1(x)=x -> x*x=x & x;x=x.
%     s1(x)=x -> x*x=x;x.
%     s1(x)=x -> x;(y*z)<=(x;y)*(x;z).
%     s1(x)=x -> x;(y;x)<=x*y.
%     s1(x)=x -> x;(y;x)=x*y.             % 4-element counterexample
%     s1(x)=x -> s1(x)=s2(x).
%     s2(x)=x -> s1(x)=s2(x).             % 4-element counterexample
%     x<=s1(y) -> s1(x)<=s1(y).           % 10.6.7
```

```
%     x<=s1(y) -> s2(x)<=s1(y).           % 10.6.7
end_of_list.
```