OPEN ACCESS
OPUS AUGSBURG

Universität Augsburg
Universitätsbibliothek

Exploring an interface model for CKA

Bernhard Möller, Tony Hoare

Angaben zur Veröffentlichung / Publication details:

Möller, Bernhard, and Tony Hoare. 2015. "Exploring an interface model for CKA." *Lecture Notes in Computer Science* 9129: 1–29. https://doi.org/10.1007/978-3-319-19797-5_1.

Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:
Deutsches Urheberrecht
Weitere Informationen finden Sie unter: / For more information see:
https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publiz/

Exploring an Interface Model for CKA

Bernhard Möller[1] and Tony Hoare[2]

[1] Institut für Informatik, Universität Augsburg, Germany
[2] Microsoft Research, Cambridge, UK

Abstract. Concurrent Kleene Algebras (CKAs) serve to describe general concurrent systems in a unified way at an abstract algebraic level. Recently, a graph-based model for CKA has been defined in which the incoming and outgoing edges of a graph define its input/output interface. The present paper provides a simplification and a significant extension of the original model to cover notions of states, predicates and assertions in the vein of algebraic treatments using modal semirings. Moreover, it uses the extension to set up a variant of the temporal logic CTL^* for the interface model.

Keywords: Concurrency, Temporal Logic, Algebra, Formal Methods

1 Introduction

Concurrent Kleene Algebra (CKAs) is intended to describe general concurrent systems in a unified way at an abstract algebraic level. It may be used to define and explore the behaviour of a computer system embedded in its wider environment, including even human users. Moreover, it provides an algebraic presentation of common-sense (non-metric) spatial and temporal reasoning about the real world.

Its basic ingredients are *events*, i.e., occurrences of certain primitive actions, such as assignments or sending/receiving on channels. The different occurrences of an action are thought as being distinguished, for instance, by time or space coordinates. A *trace* is a set of events, and a *program* or *specification* is a set of traces. Programs and specifications are distinguished by the operators that are applicable to them. For instance, while set-theoretic intersection or complement make sense for specifications, they do not for programs. Another essential concept is that of a *dependence* relation between events which expresses causal or temporal succession. The events of a trace and their dependence therefore define a graph. The fundamental connection operators on traces and programs are sequential composition ; and concurrent composition | , governed by characteristic laws.

Various models of these laws have been proposed. In [18] an interface model for CKA was sketched. It considers *graphlets*, i.e., subgraphs of an overall trace, as "events" of their own. The *interface* of a graphlet consists of the arrows that connect it to neighbouring ones. In the present paper we simplify and significantly extend that model. Moreover, we show how the interface operators can be used to cover notions of states, predicates and assertions in the vein of algebraic treatments using modal semirings. Finally we set up a variant of the temporal logic CTL^* for the interface model.

Temporal logics, even for so-called branching time, were inspired by interleaving semantics and hence linear traces. It has been shown e.g. in [23] that this can be abstracted to more general algebraic structures such as quantales. We use this idea to set up a temporal logic for the interface model. The resulting semantics mainly covers sequential aspects. We think, though, that it can be useful for analysing quasilinear threads of CKA programs, i.e., subgraphs which do not consider incoming or outgoing arrows from the environment but only their own "inner flow" of causal or temporal succession.

Besides these main topics, the paper also sets up links to other semantic notions, in particular, to the traditional assertional interpretation of Hoare Logic and to the modal operators diamond and box. It is structured as follows.

Sect. 2 repeats the most important notions of CKA and its overall approach to semantics.

In Sect. 3 the interface model is presented. Using restriction operators, simple while programs and a counterpart of standard Hoare triples are defined. A variant of the sequential composition operator allows a connection to the more recent type of Hoare triples from [30,15]. Finally, restriction operators are used to define modal box and diamond operators for the interface model.

After that, Sect. 4 deals with the mentioned variant of CTL^*. After some basic material on temporal logics and their algebraic semantics in general, based on earlier work in [23], an adaptation for the interface model is achieved and shown at work in some small examples.

The paper finishes with a short conclusion and outlook; an appendix provides deferred detailed proofs.

2 Basic Concepts of CKA

To make the paper self-contained, we repeat some basic facts about CKA.

2.1 Traces

Assume a set of *events*. As mentioned in the Introduction, these might be occurrences of basic actions, distinguished by time or space stamps or the like. A *trace* is a set of such events. Traces will be denoted by p, q, r and primed versions of these. CKA assumes a *refinement relation* $\Rightarrow$ between traces which is at least a preorder. $p \Rightarrow q$ means that every event of p is also possible for q where q may be a specification of all desirable events of p.

The trace *skip* (also denoted by 1) describes doing nothing, for example because the task at hand has been accomplished. *Bottom* ($\bot$) is an error element standing for a trace which is physically impossible, for example because it requires an event to occur before an event which caused it. It is the least element w.r.t. $\Rightarrow$. *Top* ($\top$) stands for a trace which contains a generic programming error, for example a null dereference, a race condition or a deadlock. It is the greatest element w.r.t. $\Rightarrow$.

The two essential binary composition operators ; and | require disjointness of their argument traces to achieve modularisation in the sense of Separation Logic [24]. The *sequential composition* $p\,;q$ describes execution of both p and q, where p can finish before q starts. *Concurrent composition* $p\,|\,q$ describes execution of both p and q, where p and q can start together and can finish together. In between, they can interact with each other and with their common environment.

Both ; and | are assumed to be associative with unit 1 and zero $\bot$; in addition | has to be commutative. Moreover, ; and | have to be *covariant*[3] in both arguments. For example, $p \Rightarrow q$ implies $p; r \Rightarrow q; r$. This allows refinement in larger contexts composed using | and ; .

Sequential and concurrent composition need to satisfy the following inequational analogue of the *exchange (or interchange) law* of Category Theory, also known as subsumption or subdistribution:

$$(p \mid q) \,;\, (p' \mid q') \Rightarrow (p \,;\, p') \mid (q \,;\, q') \ . \tag{exchange}$$

Since 1 is a shared unit of ; and |, specialising q or/and p' to 1 and use of commutativity of | yields the *frame laws*

$$p \,;\, (p' \mid q') \Rightarrow (p \,;\, p') \mid q' \ , \tag{frame I}$$

$$(p \mid q) \,;\, q' \Rightarrow p \mid (q \,;\, q') \ , \tag{frame II}$$

$$p \,;\, q' \Rightarrow p \mid q' \ . \tag{frame III}$$

2.2 Programs and Specifications

A *program* or a *specification* is a non-empty set P of traces that is downward closed w.r.t. $\Rightarrow$, i.e., satisfies $p \in P \wedge p' \Rightarrow p \implies p' \in P$. As mentioned above, specifications and programs are distinguished by the operators that are admissible for them. Programs will be denoted by P, Q, R and primed versions of these. By $\bot$ we also denote the program/specification consisting just of the trace $\bot$; it corresponds to a contradictory specifying predicate, i.e., to false. The set of all traces will be denoted by U.

The function dc forms the downward closure of a set S of traces, i.e.,

$$dc(S) \ =_{df} \ \{p' \mid \exists\, p \in S : p' \Rightarrow p\} \ . \tag{1}$$

For a single element $t \in U$ we abbreviate $dc(\{t\})$ by $dc(t)$. Hence a program is a set P of traces with $P = dc(P)$. For instance, $\mathsf{skip} =_{df} dc(1) = \{\bot, 1\}$ lifts 1 to the level of programs.

The *choice* $P \,[\!]\, Q$ between programs is the union $P \cup Q$ and describes the execution of a P-trace or a Q-trace. The choice may be determined or influenced by the environment, or it may be left wholly indeterminate. The operator is

[3] also called *monotone* or *isotone*

associative, commutative and idempotent, with $\bot$ as unit and U as zero. Finally, it can be used to define the refinement relation between programs/specifications:

$$P \Rightarrow Q \quad \text{iff} \quad P \mathbin{[\!]} Q = Q \; .$$

By downward closure, $\Rightarrow$ coincides with inclusion on programs. Pointwise one has

$$P \Rightarrow P' \iff_{df} \forall p \in P : \exists p' \in P' : p \Rightarrow p' \; . \tag{2}$$

By this, a program P refines a specification P' if each of its traces refines a trace admitted by the specification. $\bot$ and U are the least and greatest elements w.r.t. $\Rightarrow$.

Occasionally we also use the intersection operator $\cap$ on specifications; it is associative, commutative and idempotent with unit U and zero $\bot$.

2.3 Lifting and Laws

We now present the important principle of lifting operators and their laws from the level of traces to that of programs.

Definition 2.1 When $\circ : U \times U \to U$ is a, possibly partial, binary operator on U, its *pointwise lifting* to programs P, P' is defined as

$$P \circ P' =_{df} dc(\{t \circ t' \mid t \in P, t' \in P' \text{ and } t \circ t' \text{ is defined}\}) \; .$$

A sufficient condition for an inequational law $p \Rightarrow p'$ to lift from traces to programs is *linearity*, viz. that every variable occurs at most once on both sides of the law and that all variables in the left hand side P also occur in the right hand side P'. Examples are the exchange and frame laws. For equations a sufficient condition is *bilinearity*, meaning that both constituting inequations are linear. Examples are associativity, commutativity and neutrality. The main result is as follows.

Theorem 2.2 *If a linear law $p \Rightarrow p'$ holds for traces then it also holds when all variables in p, p' are replaced by variables for programs and the operators are interpreted as the liftings of the corresponding trace operators.*

We illustrate the gist of the proof for the case of the frame law $P;P' \Rightarrow P|P'$.

$$\begin{array}{ll}
 & r \in P \,;\, P' \\
\Leftrightarrow & \quad \{\![\text{ by Def. 2.1 and (1) }]\!\} \\
 & \exists\, t \in P, t' \in P' : r \Rightarrow t \,;\, t' \\
\Rightarrow & \quad \{\![\text{ by } t \,;\, t' \Rightarrow t \mid t' \text{ (frame III) and transitivity of } \Rightarrow]\!\} \\
 & \exists\, t \in P, t' \in P' : r \Rightarrow t \mid t' \\
\Rightarrow & \quad \{\![\text{ by Def. 2.1 and (1) }]\!\} \\
 & r \in P \mid P' \; .
\end{array}$$

The full proof for general preorders can be found in [17].

Corollary 2.3 (Laws of Trace Algebra for Programs) *The liftings of the operators* ; *and* | *to programs are associative and have* 1 *as a shared unit. Moreover,* | *is commutative and the exchange law and therefore also the frame laws hold.*

There are further useful consequences of our definition of programs. The set $\mathcal{P}$ of all programs forms a complete lattice w.r.t. the inclusion ordering; it has been called the *Hoare power domain* in the theory of denotational semantics (e.g. [31,22,4]). The least element is the program $\{\perp\}$, again denoted by $\perp$, while the greatest element is the program U consisting of all traces. Infimum and supremum coincide with intersection and union, since downward closed sets are also closed under these operations. Also, refinement coincides with inclusion, i.e., $P \Rightarrow Q \iff P \subseteq Q$. We will use this latter notation throughout the rest of the paper.

By completeness of the lattice we can define (unbounded) choice between a set $\mathcal{Q} \subseteq \mathcal{P}$ of programs as

$$[]\ \mathcal{Q} =_{df} \bigcup \mathcal{Q} \ .$$

The lifted versions of covariant trace operators are covariant again, but even distribute through arbitrary choices between programs. This means that the set of all programs forms a *quantale* (e.g. [25]) w.r.t. to the lifted versions of both ; and | . This will be used in Sect. 4 to set up a connection with temporal logics.

Covariance of the lifted operators, together with completeness of the lattice of programs and the Tarski-Knaster fixed point theorem guarantees that recursion equations have least and greatest solutions. More precisely, let $f : \mathcal{P} \to \mathcal{P}$ be a covariant function. Then f has a least fixed point μf and a greatest fixed point νf, given by the formulas

$$\mu f \ = \bigcap \{P \,|\, f(P) \subseteq P\} \ , \qquad \nu f \ = \bigcup \{P \,|\, P \subseteq f(P)\} \ . \tag{3}$$

With the operator ; , this can be used to define the Kleene star (see e.g. [5]), i.e., unbounded finite sequential iteration, of a program P as $P^* =_{df} \mu f_P$, where

$$f_P(X) =_{df} \mathsf{skip}\ []\ P\ []\ X\,;X \ .$$

Equivalently, $P^* = \mu g = \mu h$, where

$$g_P(X) =_{df} \mathsf{skip}\ []\ P\,;X \ , \qquad h_P(X) =_{df} \mathsf{skip}\ []\ X\,;P \ . \tag{4}$$

Since f_P, by the above remark, distributes through arbitrary choices between programs, it is even continuous and Kleene's fixed point theorem tells us that $P^* = \mu f_P$ has the iterative representation

$$P^* = \bigcup \{f_P^i(\emptyset) \,|\, i \in \mathbb{N}\} \ , \tag{5}$$

which transforms into the well known representation of star, viz.

$$P^* = \bigcup \{P^i \,|\, i \in \mathbb{N}\}$$

with $P^0 =_{df} \mathsf{skip}$ and $P^{i+1} =_{df} P\,;P^i$.

We show an example for the interplay of Kleene star and the exchange law.

Lemma 2.4 $(P \mid Q)^* \subseteq P^* \mid Q^*$.

Proof. Given the representation of least fixed points in Eq. (3) it suffices to show that $P^* \mid Q^*$ is contracted by the generating function $g_{P|Q}$ of $(P \mid Q)^*$. By the fixed point property of star, distributivity of lifted operators, neutrality of skip and omitting choices, exchange law and covariance, and definition of $g_{P|Q}$:

$$\begin{array}{ll} & P^* \mid Q^* \\ = & (\mathsf{skip} \mathbin{[\,]} P\,;P^*) \mid (\mathsf{skip} \mathbin{[\,]} Q\,;Q^*) \\ = & (\mathsf{skip} \mid \mathsf{skip}) \mathbin{[\,]} (\mathsf{skip} \mid (Q\,;Q^*)) \mathbin{[\,]} ((P\,;P^*) \mid \mathsf{skip}) \mathbin{[\,]} ((P\,;P^*) \mid (Q\,;Q^*)) \\ \supseteq & \mathsf{skip} \mathbin{[\,]} ((P\,;P^*) \mid (Q\,;Q^*)) \\ \supseteq & \mathsf{skip} \mathbin{[\,]} ((P \mid Q)\,;(P^* \mid Q^*)) \\ = & g_{P|Q}(P^* \mid Q^*)\ . \end{array}$$

□

Infinite iteration P^ω can be defined as the greatest fixed point νg_P where

$$g_P(X) =_{df} P\,;X\ .$$

However, there is no representation of P^ω similar to (5) above, because semicolon does not distribute through intersection and hence is not co-continuous; we only have the inequation

$$P^\omega \subseteq \bigcap\{P^i\,;U \mid i \in \mathbb{N}\}\ .$$

To achieve equality, in general the iteration and intersection would need to be transfinite.

Sometimes it is convenient to work with an iteration operator that leaves it open whether the iteration is finite or infinite; this can be achieved by Back and von Wright's operator [1]

$$P^{\widehat{\omega}} =_{df} P^\omega \mathbin{[\,]} P^*\ ,$$

which is the greatest fixed point of the function g_P above.

Along the same lines, unbounded finite and infinite concurrent iteration of a program can be defined.

3 An Interface Model for CKA

3.1 Motivation and Basic Notions

This section presents a simplification and substantial extension of a CKA model given in [15]. It supports the process of developing a system architecture and design at any desired level of granularity. It abstracts from the plethora of internal events of a trace, and models only the external interface of each component. The interface is described as a set of arrows, each of which connects a pair of events, one of them inside the component, and one of them outside.

The set of all arrows defines a desired minimum level of granularity of the global dependence relation which is induced as its reflexive-transitive closure. Events and arrows together define a graph. A part of that graph will be called a graphlet below.

3.2 Graphlets

Assume, as in Sect. 2.1, a set EV of *events* and additionally a set AR of *arrows*. Moreover, assume total functions $s, t : \mathrm{AR} \to \mathrm{EV}$ that yield the *source* and *target* of each arrow. For event sets $E, E' \subseteq \mathrm{EV}$ we define, by a slight abuse of notation, the set $E{\times}E'$ of arrows by

$$a \in E{\times}E' \iff_{df} s(a) \in E \wedge t(a) \in E' \ .$$

We choose this notation, since × will play the role of the conventional Cartesian product of sets of events as used in relationally based models such as the trace model of [17]. The operator × distributes through *disjoint union* + and hence is covariant in both arguments. We let × bind tighter than + and ∩. This gives the following properties (with $\overline{E} =_{df} \mathrm{EV} - E$) which are immediate by Boolean algebra:

$$\begin{aligned} \overline{E{\times}E'} &= \mathrm{EV}{\times}\overline{E'} + \overline{E}{\times}E' = E{\times}\overline{E'} + \overline{E}{\times}\mathrm{EV} \\ &= E{\times}\overline{E'} + \overline{E}{\times}E' + \overline{E}{\times}\overline{E'}\ , \\ (E \cap E'){\times}(E'' \cap E''') &= (E{\times}E'') \cap (E'{\times}E''') = (E{\times}E''') \cap (E'{\times}E'')\ , \\ \emptyset{\times}E &= \emptyset = E{\times}\emptyset\ . \end{aligned} \tag{6}$$

Definition 3.1 A *graphlet* is a pair $G = (E, A)$ with $E \subseteq \mathrm{EV}, A \subseteq \mathrm{AR}$ satisfying the following healthiness condition: there are no “loose” arrows, i.e.,

$$A \cap \overline{E}{\times}\overline{E} = \emptyset\ , \tag{7}$$

In words, every arrow in A has at least one event in E.

The healthiness condition is mandatory for validating the associativity and exchange laws for the trace algebra operators on graphlets, as will become manifest in their proof.

Since arrows have identity (and do not just record whether or not a connection exists between two events, as is done in relational models), we can associate values, labels, colours etc. with them without having to include extra functions for that into the model.

Example 3.2 We specify a graph H that models a thread corresponding to a single natural-number variable `x`, already initialised to 0, with increment as the only operation. Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{N}_+ =_{df} \mathbb{N} - \{0\}$. We use $\mathrm{EV} =_{df} \{inc_i \mid i \in \mathbb{N}\}$ and $A =_{df} \mathbb{N}_+$ with the source and target maps

$$s(i) =_{df} inc_{i-1}\ , \qquad t(i) =_{df} inc_i\ .$$

Pictorially,

$$H: \quad inc_0 \xrightarrow{1} inc_1 \xrightarrow{2} \cdots$$

Some graphlets of H then are the following:

$$\begin{tikzcd}
 & inc_0 \arrow[r, "1"] & {} & \\
{} \arrow[r, "2"] & inc_2 \arrow[r, "3"] & {} & \\
inc_0 \arrow[r, "1"] & inc_1 \arrow[r, "2"] & inc_2 \arrow[r, "3"] & {}
\end{tikzcd}$$

□

Definition 3.3 The sets of *input, output* and *internal* arrows of graphlet G are

$$\begin{aligned} in(G) &=_{df}\ A \cap \overline{E}\times E\ , \\ out(G) &=_{df}\ A \cap E\times \overline{E}\ , \\ int(G) &=_{df}\ A \cap E\times E\ . \end{aligned}$$

These sets are pairwise disjoint, and by the healthiness condition (7) in Def. 3.1 we have $A = in(G) \cup out(G) \cup int(G)$. The sets $in(G)$ and $out(G)$ together constitute the *interface* of G to its environment. The set of all graphlets is denoted by G.

We want to compose graphlets G and G' by connecting output arrows of G to input arrows of G'. If these arrows carry values of some kind, we view the composition as transferring these values from the source events of these arrows in G to their target events in G'. To achieve separation, we require that the event sets of G and G' are disjoint. While concurrent composition $G\,|\,G'$ does not place any further restriction, in sequential composition $G\,;G'$ we forbid "backward" arrows from G' to G.

Definition 3.4 Let G, G' be graphlets with disjoint event sets. We set

$$\begin{aligned} G\mid G' &=_{df}\ (E+E', A\cup A')\ , \\ G\,;G' &=_{df}\ \begin{cases} G\,|\,G' & \text{if } \mathrm{CS}(G,G')\ , \\ \text{undefined otherwise}\ , \end{cases} \end{aligned}$$

where

$$\begin{aligned} \mathrm{CS}(G,G') &\iff_{df}\ A\cap A'\cap E'\times E = \emptyset \\ &\iff\ \ A\cap A' \subseteq E\times E' \end{aligned}$$

formalises the above requirement of no backward arrows from G' to G.

The "connection" of output and input arrows takes place as follows: if a is an arrow in $out(G)\cap in(G')$ then it will also be in $A\cup A'$ and both its end points will be in $E+E'$, so that it is an internal arrow of $G\,|\,G'$. Clearly, $G\,|\,G'$ is a graphlet again. Since disjoint union and union are associative, also $|$ is associative and has the *empty graphlet* $\square =_{df} (\emptyset,\emptyset)$ as its unit.

Assume that in a term $(G\mid G')\,;(H\mid H')$ the condition CS is satisfied for the operands of ;. Then it also holds for the sequential compositions $G\,;H$ and $G'\,;G''$ that occur on the right hand side of the corresponding exchange law, so that then all three sequential compositions are defined.

Lemma 3.5

1. $in(G \mid G') \;=\; (in(G) \cap \overline{E'} \times E) \,\cup\, (in(G') \cap \overline{E} \times E')$.
2. $out(G \mid G') \;=\; (out(G) \cap E \times \overline{E'}) \,\cup\, (out(G') \cap E' \times \overline{E})$.

The somewhat tedious proof is deferred to the Appendix.

As the refinement relation between graphlet terms p, p' we use what is known as the *flat order* in denotational semantics:

$$\begin{array}{lll} p \Rightarrow p' & \Longleftrightarrow_{df} & p \text{ is undefined or} \\ & & p, p' \text{ are both defined and have equal values.} \end{array}$$

Theorem 3.6 *The operators* ; *and* | *are associative and obey the exchange and frame laws. Moreover,* | *is commutative and* □ *is a shared unit of* ; *and* | *.*

See again the Appendix for details of the proof. We note, however, that absence of "loose" arrows from graphlets is essential for it.

Definition 3.7 We now use graphlets as traces. Hence in the interface model a program is a set of graphlets. The program that does nothing is $\mathsf{skip} =_{df} \{\square\}$, while U is the program consisting of all graphlets.

Since we use the flat refinement order between graphlets, the set of programs is isomorphic to the power set of the program U that consists of all graphlets. By the general lifting result quoted in Th. 2.2, the above Thm. 3.6 extends to graphlet programs as well.

Example 3.8 Consider again the graph H from Ex. 3.2 and the program P consisting of all singleton graphlets of H:

$$P \;=_{df}\; \{\; inc_0 \xrightarrow{1} \;\} \;\cup\; \{\; \xrightarrow{i} inc_i \xrightarrow{i+1} \;\mid i \in \mathbb{N}_+\} \;.$$

It represents the single-step transition relation associated with the statement `x := x + 1`. Then $P^2 = P\,;P = P \mid P$ contains

$$\xrightarrow{i} inc_i \xrightarrow{i+1} inc_{i+1} \xrightarrow{i+2}$$

for all $i \in \mathbb{N}$, but also non-contiguous graphlets such as

$$\xrightarrow{i} inc_i \xrightarrow{i+1} \qquad\qquad \xrightarrow{j} inc_{i+1} \xrightarrow{j+1}$$

for $j + 1 \neq i \neq j \neq i + 1$. We will later define a variant of ; that excludes non-contiguous graphlets. □

3.3 Restriction and while Programs

As mentioned in the motivation in Sect. 3.1, a set of arrows can be viewed as a description of a set of variable/value associations. We can use such a set to characterise "admissible" or "desired" associations. Usually one will require such a set to be *coherent* in some sense, like not having contradictory associations. We give no precise definition of coherence, but leave it a parameter to our treatment. A minimum healthiness requirement is that the empty arrow set $\emptyset$ and each of the interface sets $in(G)$ and $out(G)$ of every graphlet G should be coherent. The set $int(G)$ of internal arrows of G will usually not be coherent, since the variable/value associations may change during the internal flow of control. For brevity we refer to coherent arrow sets as *states* in the sequel.

First we introduce restriction operators and, based on these, while programs.

Definition 3.9 The *input restriction* of graphlet G to set $C \subseteq \mathrm{AR}$ is

$$C \rfloor G =_{df} \begin{cases} G & \text{if } in(G) = C \ , \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Output restriction $G \lfloor C$ is defined symmetrically.

Definition 3.10 A *predicate* is a set of arrow sets that satisfy the coherence requirement. Restriction is lifted pointwise to predicates and programs.

Hence the input restriction $S \rfloor Q$ of program Q to a predicate S retains only those graphlets $G \in Q$ for which $in(G) \in S$. An analogous remark holds for output restriction.

Predicates will mimic the role of tests, i.e., of elements below the multiplicative unit, in semirings. In the standard model of CKA and also the interface model there are, however, only the tests $\bot$ and skip. So internalising restriction as pre- or post-multiplication by tests is not possible there in a reasonable way, which is why we resort to the above separate restriction operators. In the recent model of [19] non-trivial test elements exist, but the model is restricted in a number of ways.

Restriction obeys useful laws, quite analogous to the case of semirings with tests; for the proofs see the corresponding ones for the *mono modules* introduced in [7] (Lm. 7.1 and 7.2 there).

Lemma 3.11 *For predicate S and programs P, Q,*

$$\begin{aligned} S \rfloor (P \cap Q) \ &= \ (S \rfloor P) \cap Q \ = \ P \cap (S \rfloor Q) \ = \ (S \rfloor P) \ \cap \ (S \rfloor Q) \ , \\ (P \cap Q) \lfloor S \ &= \ (P \lfloor S) \cap Q \ = \ P \cap (Q \lfloor S) \ = \ (P \lfloor S) \ \cap \ (Q \lfloor S) \ . \end{aligned}$$

In particular, for $Q = U$ with U being the program consisting of all graphlets, $S \rfloor P = P \cap S \rfloor U$ and $P \lfloor S = P \cap U \lfloor S$.

Now we can define an if then else and a while do construct.

Definition 3.12 Given a predicate S and programs P, Q we set

$$\begin{aligned} &\textsf{if } S \textsf{ then } P \textsf{ else } Q \;\; =_{df} \; (S \downharpoonleft P) \mathbin{[]} (\overline{S} \downharpoonleft Q) \ , \\ &\textsf{while } S \textsf{ do } P =_{df} \; (S \downharpoonleft P)^* \downharpoonright \overline{S} \ , \end{aligned}$$

where $\overline{S}$ is the complement of S in the set of predicates.

Since this definition is analogous to the one for semirings with tests, all the usual algebraic rules for while programs carry over to the present setting.

If one wants to include the possibility of loops with infinite iteration one can use the program ($\textsf{while } S \textsf{ do } P$) $\mathbin{[]}$ $(S \downharpoonleft P)^\omega$.

3.4 Hoare Triples

We show now how predicates can be used to define Hoare triples for graphlet programs.

Definition 3.13 The *standard Hoare triple* $\{S\}\, P\, \{S'\}$ with predicates S, S' and program P is defined by

$$\{S\}\, P\, \{S'\} \iff_{df} \; S \downharpoonleft P \subseteq P \downharpoonright S' \ .$$

This means that, starting with a variable/value association in S, execution of P is guaranteed to "produce" a variable/value association in S'. See [2] for a closely related early relational definition. We give two equivalent formulations of the standard Hoare triple.

Lemma 3.14 *Let U, as in Def. 3.7, be the program consisting of all graphlets.*

$$\{S\}\, P\, \{S'\} \iff S \downharpoonleft P \subseteq U \downharpoonright S' \iff S \downharpoonleft P = (S \downharpoonleft P) \downharpoonright S' \ .$$

The formula $S \downharpoonleft P \subseteq U \downharpoonright S'$ expresses more directly that the "outputs" of $S \downharpoonleft P$ satisfy the predicate S' and is calculationally advantageous, since the variable P is not repeated. The formula $S \downharpoonleft P = (S \downharpoonleft P) \downharpoonright S'$ says that post-restriction of $S \downharpoonleft P$ to S' is no proper restriction, since its "outputs" already satisfy S'.

Proof. For the first equivalence let $Q = S \downharpoonleft P$. By Boolean algebra, Lm. 3.11, definition of intersection, and since $Q \subseteq P$ is true by the definition of restriction:

$$\begin{aligned} Q \subseteq P \downharpoonright S' &\iff Q \subseteq U \cap (P \downharpoonright S') \iff Q \subseteq (U \downharpoonright S') \cap P \\ &\iff Q \subseteq U \downharpoonright S' \wedge Q \subseteq P \iff Q \subseteq U \downharpoonright S' \ . \end{aligned}$$

For the second one we have by Lm. 3.11, and by the just shown first equivalence:

$$(S \downharpoonleft P) \downharpoonright S' = (S \downharpoonleft P) \cap (U \downharpoonright S') = S \downharpoonleft P \ .$$

□

Def. 3.13 entails the well known Hoare calculus with all its inference rules, in particular the $\textsf{if then else}$ and $\textsf{while}$ rules.

Let us connect this to another, more recent view of Hoare triples [15].

Definition 3.15 For programs P, P' and Q one sets

$$P \;\boldsymbol{\{}Q\boldsymbol{\}}\; P' \;\Longleftrightarrow_{df}\; P\,;Q \subseteq P' \;.$$

This expresses that, after any graphlet in "pre-history" P, execution of Q is guaranteed to yield an overall graphlet in P'. For the case where programs are relations between states this definition appears already in [29]. These new triples enjoy many pleasant properties; see again [15] for details. We show two samples which will be taken up again in the next section.

Lemma 3.16 *Let program P be an invariant of program Q, i.e., assume $P \;\boldsymbol{\{}Q\boldsymbol{\}}\; P$.*

1. *P is also an invariant of program Q^*, i.e., $P \;\boldsymbol{\{}Q^*\boldsymbol{\}}\; P$.*
2. *$(P\,;Q)^\omega \subseteq P^\omega$ and $P \;\boldsymbol{\{}(Q\,;P)^\omega\boldsymbol{\}}\; P^\omega$.*

Proof.

1. By standard Kleene algebra, in generalisation of Eq. (4) one has $P\,;Q^* = \mu k_{PQ}$, where $k_{PQ}(X) =_{df} P \mathbin{[\,]} X;Q$. Thus the claim is shown if we can prove that also P is a fixed point of k_{PQ}. Indeed, by the assumption $P\,;Q \subseteq P$,
$$k_{PQ}(P) \;=\; P \mathbin{[\,]} P\,;Q = P \;.$$
2. The first claim is immediate from the assumption and covariance of the $^\omega$ operator. The second claim follows from the first one by the so-called rolling rule for the omega operator: $(P\,;Q)^\omega = P\,;(Q\,;P)^\omega$.

□

A relationship between standard Hoare triples and a variant of the new ones will be set up in Lm. 3.19 of the next section.

3.5 Quasilinear Sequential Composition

We define a strengthened form of sequential composition.

Definition 3.17 For graphlets G, G' the *quasilinear sequential composition* is defined by

$$G \mathbin{\boldsymbol{;}} G' =_{df} \begin{cases} G' & \text{if } G = \Box \;, \\ G & \text{if } G' = \Box \;, \\ G\,;G' & \text{if } G, G' \neq \Box \text{ and } \mathit{out}(G) = \mathit{in}(G') \;, \\ \text{undefined} & \text{otherwise} \;. \end{cases}$$

We call it quasilinear, since it does not allow "in-branching" or "out-branching" at the border between the composed graphlets and therefore leads to quasi linear thread-like graphlets when iterated finitely or infinitely. Note, however, that in the overall graph there may still be arrows to or from the environment which have been disregarded in selecting the arrows belonging to the graphlets in question.

The definition could be simplified if graphlets were allowed to have "loose" arrows; however, as remarked after Th. 3.6, this would destroy associativity of $|$ and $;$.

Lemma 3.18 *Quasilinear sequential composition is associative and satisfies the following laws (that do not hold for ;), assuming definedness of the terms involved.*

$$\begin{array}{rcl}
\mathit{in}(G \,\boldsymbol{;}\, G') = \mathit{in}(G) = \mathit{in}(G \downharpoonleft \mathit{in}(G')) & \Leftarrow & G \neq \Box\ , \\
\mathit{out}(G \,\boldsymbol{;}\, G') = \mathit{out}(G') = \mathit{out}(\mathit{out}(G) \rfloor G') & \Leftarrow & G' \neq \Box\ , \\
(G \downharpoonleft C) \,\boldsymbol{;}\, G' = G \,\boldsymbol{;}\, (C \rfloor G|) & \Leftarrow & G, G' \neq \Box\ , \\
C \rfloor (G \,\boldsymbol{;}\, G') = (C \rfloor G) \,\boldsymbol{;}\, G' & \Leftarrow & G \neq \Box\ , \\
(G \,\boldsymbol{;}\, G') \downharpoonleft C = G \,\boldsymbol{;}\, (G' \downharpoonleft C) & \Leftarrow & G' \neq \Box\ .
\end{array}$$

The proof is again somewhat tedious and hence deferred to the Appendix.

We lift **;** to programs as usual. The lifted operator has skip as its unit.

Let again U denote the universal program consisting of all graphlets. Then we have the following connection between standard Hoare triples and a variant of the new ones.

Lemma 3.19 *Assume that* $\Box \in P \iff \mathit{in}(\Box) = \emptyset \in S \iff \emptyset \in S'$. *Then*

$$\{S\}\, P\, \{S'\} \iff U \downharpoonleft S\ \boldsymbol{\{}P\boldsymbol{\}}\ U \downharpoonleft S'\ .$$

The right hand side means that an arbitrary "pre-history" with a result that satisfies S, followed by P, makes an overall history with a result satisfying S'.

Proof. According to the definitions, $U \downharpoonleft S\ \boldsymbol{\{}P\boldsymbol{\}}\ U \downharpoonleft S'$ spells out to $(U \downharpoonleft S) \,\boldsymbol{;}\, P \subseteq U \downharpoonleft S'$. By the assumption, the laws of Lm. 3.18 lift to the programs and predicates involved.

($\Rightarrow$) By Lm. 3.14, isotony of lifted operators, Lm. 3.18, and $U \,\boldsymbol{;}\, U \subseteq U$ with isotony of restriction:

$$\begin{array}{l}
\{S\}\, P\, \{S'\} \iff S \rfloor P \subseteq U \downharpoonleft S' \Longrightarrow U \,\boldsymbol{;}\, (S \rfloor P) \subseteq U \,\boldsymbol{;}\, (U \downharpoonleft S') \\
\qquad\qquad \iff (U \downharpoonleft S) \,\boldsymbol{;}\, P \subseteq (U \,\boldsymbol{;}\, U) \downharpoonleft S' \Longrightarrow (U \downharpoonleft S) \,\boldsymbol{;}\, P \subseteq U \downharpoonleft S'\ .
\end{array}$$

($\Leftarrow$) By neutrality of skip, isotony of lifted operators, Lm. 3.18, and the assumption:

$$S \rfloor P = \mathsf{skip} \,\boldsymbol{;}\, (S \rfloor P) \subseteq U \,\boldsymbol{;}\, (S \rfloor P) = (U \downharpoonleft S) \,\boldsymbol{;}\, P \subseteq U \downharpoonleft S'\ .$$

□

A dual construction using input restriction gives a connection between standard Hoare triples and the analogous variant of *Milner triples*, see [16]. These take the form $P \rightarrow_Q R$ and are defined by

$$P \rightarrow_Q R \iff_{df} P \supseteq Q; R\ .$$

Although the Milner triple modulo renaming has the same definition as the new Hoare triple, its informal interpretation is more that of operational semantics: it says that one way of executing P is to first execute Q (or to do a Q-transition), leading to the residual program R.

Example 3.20 Using $\boldsymbol{;}$ in place of ; in forming powers and star of a program eliminates non-contiguous graphlets like the one shown in Ex. 3.8. Considering again the program P from Ex. 3.2 we can write a loop that increments the variable `x` until it becomes 10:

$$\mathsf{while}\ [0,9]\ \mathsf{do}\ P\ ,$$

where $[0,9]$ is the interval from 0 to 9.

It consists of the graphlets

$$\xrightarrow{i} inc_i \xrightarrow{i+1} \cdots \xrightarrow{9} inc_9 \xrightarrow{10}$$

for $i \in [1,9]$ and

$$inc_0 \xrightarrow{1} inc_1 \xrightarrow{2} \cdots \xrightarrow{9} inc_9 \xrightarrow{10}$$

□

We conclude this section with an invariance property, related to that of Lm. 3.16.2.

Lemma 3.21 *If $\square \notin P$ and S is an invariant of P, i.e., if $\{S\}\,P\,\{S\}$, then it is also preserved throughout the infinite iteration of P, i.e.,*

$$S \rfloor P^\omega \;=\; (S \rfloor P)^\omega\ ,$$

where $^\omega$ is taken w.r.t. $\boldsymbol{;}$.

Proof. ($\supseteq$) We do not even need the assumption: by the fixed point definition of $(S \rfloor P)^\omega$, the definition of restriction, Lm. 3.18, the fixed point definition again, $S \rfloor P \subseteq S$ and isotony of $^\omega$,

$$\begin{aligned}(S \rfloor P)^\omega = (S \rfloor P) \boldsymbol{;} (S \rfloor P)^\omega = (S \rfloor (S \rfloor P)) \boldsymbol{;} (S \rfloor P)^\omega = S \rfloor ((S \rfloor P) \boldsymbol{;} (S \rfloor P)^\omega)\\ = S \rfloor (S \rfloor P)^\omega \subseteq S \rfloor P^\omega\ .\end{aligned}$$

($\subseteq$) By the fixed point definition, Lm. 3.18, the assumption with Lm. 3.14, and Lm. 3.18:

$$S \rfloor P^\omega = S \rfloor (P \boldsymbol{;} P^\omega) = (S \rfloor P) \boldsymbol{;} P^\omega = ((S \rfloor P) \lfloor S) \boldsymbol{;} P^\omega = (S \rfloor P) \boldsymbol{;} (S \rfloor P^\omega)\ ,$$

which means that $S \rfloor P^\omega$ is a fixed point of $\lambda x \,.\, (S \rfloor P) \boldsymbol{;} x$ and hence below the greatest fixed point $(S \rfloor P)^\omega$ of that function. □

Example 3.22 Consider again the program P from Ex. 3.8. If the powers of P as well as P^* and P^ω are taken w.r.t $\boldsymbol{;}$ rather than ; then the non-contiguous subsequences of the overall graph are ruled out from them. Moreover, letting again $\mathbb{N}_+$ be the predicate consisting of all positive natural numbers, we have $\{\mathbb{N}_+\}\,P\,\{\mathbb{N}_+\}$ and hence

$$\mathbb{N}_+ \rfloor P^\omega \;=\; (\mathbb{N}_+ \rfloor P)^\omega\ .$$

This example will be taken up later in connection with the temporal logic CTL^*.

□

3.6 Disjoint Concurrent Composition

We now briefly deal with *interaction-free* or *disjoint concurrent composition* $|||$, a special variant of concurrent composition $|$ that does not admit any arrows between its operands and hence does not prescribe any particular interleavings of their events.

Definition 3.23

$$G \,|||\, G' =_{df} \begin{cases} G \mid G' & \text{if } \mathrm{CD}(G, G')\ , \\ \text{undefined} & \text{otherwise}\ , \end{cases}$$

where

$$\mathrm{CD}(G, G') \iff_{df} A \cap E' \times E = \emptyset = A' \cap E \times E'$$

excludes any arrow between G and G'.

Lemma 3.24 *Recall that + means disjoint union.*

$$\begin{aligned} \mathit{in}(G \,|||\, G') &= \mathit{in}(G) + \mathit{in}(G')\ , \\ \mathit{out}(G \,|||\, G') &= \mathit{out}(G) + \mathit{out}(G')\ . \end{aligned}$$

The proof can be found in the Appendix.

Lemma 3.25 *The operators* **;** *and* $|||$ *satisfy the reverse exchange law*

$$(G \,\mathbf{;}\, G') \,|||\, (G'' \,\mathbf{;}\, G''') \Rightarrow (G \,|||\, G'') \,\mathbf{;}\, (G' \,|||\, G''')\ ,$$

in which the order of refinement is the reverse of that in (exchange).

Proof. Straightforward from Lm. 3.24 and the definition of **;**. □

The operator $|||$ is closely related to the disjoint concurrent composition operator $*$ in [6], for which also a reverse exchange law holds.

We conclude this section by an inference rule for Hoare triples involving $|||$: for graphlets G, G' and states C, C', D, D' we have

$$\frac{\{C\}\, G\, \{D\} \qquad \{C'\}\, G'\, \{D'\}}{\{C + D\}\, G \,|||\, G'\, \{D + D'\}}\ ,$$

where we have identified graphlets and singleton programs. The proof is again straightforward from Lm. 3.24 and the definitions.

3.7 Modal Operators

We continue with another interesting connection, namely to the theory of modal semirings (e.g. [9]), which will be useful for the variant of CTL presented in Sect. 4.4.

Input restriction obeys the following laws, where false is the empty predicate and true is the predicate containing all states:

$$\begin{aligned}
&\mathsf{false}\rfloor P = \bot \ , \\
&\mathsf{true}\rfloor P \ = \ P \\
&(S \cup S')\rfloor P \ = \ (S\rfloor P) \mathbin{[]} (S'\rfloor P) \ , \\
&(S \cap S')\rfloor P \ = \ S\rfloor (S'\rfloor P) \ , \\
&P \ = \ S\rfloor P \iff \mathit{in}(P) \subseteq S \ .
\end{aligned}$$

The first four of these say that restriction acts as a kind of module operator between states and graphlets. Symmetric laws hold for output restriction.

The last law means that *in* satisfies the characteristic property of an abstract domain operator as known from modal semirings [9], namely that $\mathit{in}(G)$ is the least preserver of G under input restriction. The law is equivalent to the following pair of laws:

$$\mathit{in}(G)\rfloor G \ = \ G \ , \qquad \mathit{in}(S\rfloor G) \ \subseteq \ S \ .$$

The domain operator can also be viewed as an "enabledness" predicate (cf. [28]).

Lifting the *in* function to programs P and predicates S we can now define modal operators.

Definition 3.26 The forward diamond operator $|P\rangle$ and box operator $|P]$ are given by

$$\begin{aligned}
|P\rangle S &=_{df} \ \mathit{in}(P\lfloor S) \ , \\
|P]S &=_{df} \ \neg|P\rangle\neg S \ .
\end{aligned}$$

The backward operators $\langle P|$ and $[P|$ are defined symmetrically using the lifted *out* function.

The forward diamond $|P\rangle S$ calculates all immediate predecessor states of S-states under program P, i.e., all states from which an S state can be reached by one P-step. The forward box operator $|P]S$ calculates all states from which every P-transition is guaranteed to satisfy S; it corresponds to Dijkstra's wlp operator, all of whose laws can be derived algebraically from these definitions. For the relational case analogous definitions appear already in [3,26]. Diamond distributes through union and is strict and covariant in both arguments. Box anti-distributes through union and hence is contravariant in its first argument, while distributes through intersection and hence is covariant in its second argument.

4 CKA and Temporal Logics

The temporal logic CTL* and its sublogics CTL and LTL are prominent tools in the analysis of concurrent and reactive systems. First algebraic treatments of these logics were obtained by von Karger and Berghammer [20,21]. A partial treatment in the framework of fork algebras was also given by Frías and Lopez Pombo [13]. For LTL compact closed expressions could be obtained by Desharnais, Möller and Struth in [8]. This was extended in [23] to a semantics for full

CTL^* and its sublogics in the framework of quantales. Since, as mentioned in Sect. 2.2, CKA has a quantale semantics, that approach can be applied to the interface model as well.

In this, sets of states and hence the semantics of state formulas can be represented as predicates, while general programs represent the semantics of path formulas.

4.1 An Algebraic Semantics of CTL^*

To make the paper self-contained, we repeat some basic facts about CTL^* and its algebraic semantics.

The language $\varPsi$ of CTL^* *formulas* (see e.g. [12]) over a set $\varPhi$ of atomic propositions is defined by the grammar

$$\varPsi \ ::= \ \bot \mid \varPhi \mid \varPsi \to \varPsi \mid \mathsf{X}\varPsi \mid \varPsi\,\mathsf{U}\,\varPsi \mid \mathsf{E}\varPsi,$$

where $\bot$ denotes falsity, $\to$ is logical implication, X and U are the next-time and until operators and E is the existential quantifier on paths.

The use of the next-time operator X means that implicitly a small-step semantics for the specified or analysed transition system is used. Informally, the formula $\mathsf{X}\varphi$ is true for a trace p if φ is true for the remainder of p after one such step. We will briefly discuss below which algebraic properties the semantic element, a program, corresponding to X should have.

The formula $\varphi\,\mathsf{U}\,\psi$ is true for a trace p if

- after zero or more X steps within p the formula ψ holds for the remaining trace and
- for all intermediate trace pieces for which ψ does not yet hold the formula φ is true.

The logical connectives $\neg, \wedge, \vee, \mathsf{A}$ are defined, as usual, by $\neg\varphi =_{df} \varphi \to \bot$, $\top =_{df} \neg\bot$, $\varphi \wedge \psi =_{df} \neg(\varphi \to \neg\psi)$, $\varphi \vee \psi =_{df} \neg\varphi \to \psi$ and $\mathsf{A}\varphi =_{df} \neg\,\mathsf{E}\neg\varphi$. Moreover, the "finally" operator F and the "globally" operator G are defined by

$$\mathsf{F}\psi =_{df} \top\mathsf{U}\psi \qquad\text{and}\qquad \mathsf{G}\psi =_{df} \neg\mathsf{F}\neg\psi\ .$$

Informally, $\mathsf{F}\psi$ holds if after a finite number of steps the remainder of the trace satisfies ψ, while $\mathsf{G}\psi$ holds if after every finite number of steps ψ still holds.

The sublanguages $\varSigma$ of *state formulas* that denote sets of states and $\varPi$ of *path formulas* that denote sets of traces are given by

$$\begin{aligned}\varSigma &::= \bot \mid \varPhi \mid \varSigma \to \varSigma \mid \mathsf{E}\varPi,\\ \varPi &::= \varSigma \mid \varPi \to \varPi \mid \mathsf{X}\,\varPi \mid \varPi\,\mathsf{U}\,\varPi.\end{aligned}$$

To motivate our algebraic semantics, we briefly recapitulate the standard CTL^* semantics formulas. Its basic objects are traces σ from T^+ or T^ω, the sets of finite non-empty or infinite words over some set T of states. The i-th element of σ (indices starting with 0) is denoted σ_i, and σ^i is the trace that results from σ by removing its first i elements.

Each atomic proposition $\pi \in \Phi$ is associated with the set $T_\pi \subseteq T$ of states for which π is true. The relation $\sigma \models \varphi$ of *satisfaction* of a formula φ by a trace is defined inductively (see e.g. [12]) by

$$\begin{array}{ll}
\sigma \not\models \bot, & \\
\sigma \models \pi & \text{iff } \ \sigma_0 \in T_\pi, \\
\sigma \models \varphi \rightarrow \psi & \text{iff } \ \sigma \models \varphi \text{ implies } \sigma \models \psi, \\
\sigma \models \mathsf{X}\,\varphi & \text{iff } \ \sigma^1 \models \varphi, \\
\sigma \models \varphi \,\mathsf{U}\, \psi & \text{iff } \ \exists\, j \geq 0 \,.\, \sigma^j \models \psi \text{ and } \forall\, k < j \,.\, \sigma^k \models \varphi, \\
\sigma \models \mathsf{E}\varphi & \text{iff } \ \exists\, \tau \,.\, \tau_0 = \sigma_0 \text{ and } \tau \models \varphi.
\end{array}$$

In particular, $\sigma \models \neg\varphi$ iff $\sigma \not\models \varphi$.

From this semantics one can extract a set-based one by assigning to each formula φ the set $[\![\varphi]\!] =_{df} \{\sigma \mid \sigma \models \varphi\}$ of paths that satisfy it. This is the basis of the algebraic semantics in terms of quantales.

We now repeat the algebraic interpretation of CTL^* over a Boolean left quantale B from [23]. To save some notation we set Φ equal to the set of *tests*, i.e., elements below the multiplicative unit 1, of that quantale. These abstractly represent sets of states of the modelled transition system. Moreover, we fix an element X that represents the transition system underlying the logic. The precise requirements for X are discussed in Sect. 4.3. Then the concrete semantics above generalises to a function $[\![\ ]\!] : \Psi \rightarrow B$, where $[\![\varphi]\!]$ abstractly represents the set of paths satisfying formula φ and $+$ and $\cdot$ are now the general quantale operators of choice and sequential composition. We note that in quantales with non-trivial test sets, left multiplication $t \cdot a$ for test t and general element a corresponds to input restriction: it leaves only those transitions of a that start with a state in t. With this, we can transform the above concrete semantics into the abstract algebraic one.

Definition 4.1 The *general quantale semantics* of CTL^* formula φ is defined inductively over the structure of φ, where $\bot$ and $\top$ are the least and greatest elements, resp., and $\overline{}$ is the complement operator:

$$\begin{aligned}
[\![\bot]\!] &= \bot \ , \\
[\![t]\!] &= t \cdot \top \ , \\
[\![\varphi \rightarrow \psi]\!] &= \overline{[\![\varphi]\!]} + [\![\psi]\!] \ , \\
[\![\mathsf{X}\,\varphi]\!] &= \mathsf{X} \cdot [\![\varphi]\!] \ , \\
[\![\varphi \,\mathsf{U}\, \psi]\!] &= \bigsqcup_{j \geq 0} (\mathsf{X}^j \cdot [\![\psi]\!] \sqcap \prod_{k<j} \mathsf{X}^k \cdot [\![\varphi]\!]) \ , \\
[\![\mathsf{E}\varphi]\!] &= \ulcorner [\![\varphi]\!] \cdot \top \ .
\end{aligned}$$

Here the domain operator $\ulcorner$ is the algebraic abstraction of the operator that yields the set of starting states τ_0 of a set of CTL^* traces τ.

4.2 CTL^* for the Interface Model

We now concretise this semantics for the CKA interface model. In fact, we can do so in two ways, namely by interpreting quantale multiplication $\cdot$ by sequential composition $;$ or quasilinear composition $\pmb{;}$. Choice $+$ is in both cases interpreted as $\mathbin{[\!]}$. Since we have used the flat refinement order on graphlets, the set of programs, i.e., of downward closed sets of graphlets, is isomorphic to the power set of the set U of all graphlets, so that the complement operator is well defined.

To model the next-step operator X of CTL^* we assume a fixed graphlet program, denoted again by X, that reflects the small-step semantics of the system to be analysed. An example for such a program would be P from Ex. 3.8. The other operators of CTL^* deal with the iteration (both finite and infinite) of the single-step semantics, i.e., with the large-step semantics. To make CTL^* work in the sense that the standard properties are obtained, the small-step program X needs to satisfy a number of assumptions that are reflected by certain CTL^* axioms. This is discussed in detail in Sect. 4.3.

To give a CTL^* semantics based on the interface model, we want to assign to every CTL^* formula φ a program, i.e., a set of graphlets, $[\![\varphi]\!]$ that describes admissible iterations of the single-step program X. In adapting the general quantale semantics of Def. 4.1, we replace tests by predicates, left multiplication by tests by restriction, $\top$ by the universal program U and the operator $^\ulcorner$ by *in*.

Definition 4.2 The semantics of CTL^* in the interface model is given as follows, where $\cdot \in \{;, \pmb{;}\}$ and the powers of X are taken w.r.t. $\cdot$.

$$\begin{aligned}
[\![\bot]\!] &= \bot, \\
[\![S]\!] &= S \downharpoonright U, \\
[\![\varphi \to \psi]\!] &= \overline{[\![\varphi]\!]} \mathbin{[\!]} [\![\psi]\!], \\
[\![\mathsf{X}\,\varphi]\!] &= \mathsf{X} \cdot [\![\varphi]\!], \\
[\![\varphi \,\mathsf{U}\, \psi]\!] &= \bigcup_{j \geq 0} (\mathsf{X}^j \cdot [\![\psi]\!] \cap \bigcap_{k<j} \mathsf{X}^k \cdot [\![\varphi]\!]), \\
[\![\mathsf{E}\varphi]\!] &= \mathit{in}([\![\varphi]\!]) \downharpoonright U.
\end{aligned}$$

Using these definitions, it is straightforward to check that

$$[\![\varphi \vee \psi]\!] = [\![\varphi]\!] \mathbin{[\!]} [\![\psi]\!], \qquad [\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!], \qquad [\![\neg\varphi]\!] = \overline{[\![\varphi]\!]}. \tag{8}$$

We have the following important property.

Theorem 4.3 *Assume that multiplication with X from the left distributes through arbitrary joins and binary meets, i.e., $\mathsf{X} \cdot (P \cap Q) = \mathsf{X} \cdot P \cap \mathsf{X} \cdot Q$ for all programs P, Q.*

1. *$[\![\varphi \,\mathsf{U}\, \psi]\!]$ is the least fixed point μf of the function $f(Y) =_{df} [\![\psi]\!] \mathbin{[\!]} ([\![\varphi]\!] \cap \mathsf{X} \cdot Y)$.*
2. *$[\![\mathsf{F}\psi]\!] = \mathsf{X}^* \cdot [\![\psi]\!]$. In particular, $[\![\mathsf{F}\top]\!] = U$.*

The proof is a direct translation of the one for general quantales given in [23].

Example 4.4 Our program P from Ex. 3.8 satisfies the assumption of that theorem. □

4.3 Requirements for Small-Step Programs

We now want to find suitable requirements on X. A fundamental requirement in standard CTL^* is validity of the axiom

$$\neg\mathsf{X}\varphi \;\leftrightarrow\; \mathsf{X}\neg\varphi\ . \tag{9}$$

To satisfy it in the algebraic setting, we need to have for all formulas φ and their semantic values $Q =_{df} \llbracket\varphi\rrbracket$,

$$\overline{\mathsf{X}\cdot Q} \;=\; \llbracket\neg\mathsf{X}\varphi\rrbracket \;=\; \llbracket\mathsf{X}\neg\varphi\rrbracket \;=\; \mathsf{X}\cdot\overline{Q}. \tag{10}$$

This semantic property can equivalently be characterised as follows (see again [23]).

Lemma 4.5 *Assume the same properties of* X *as in Th. 4.3.*

1. $\forall\, Q : \mathsf{X}\cdot\overline{Q} \subseteq \overline{\mathsf{X}\cdot Q} \iff \forall\, Q, R : \mathsf{X}\cdot(Q\cap R) = \mathsf{X}\cdot Q \cap \mathsf{X}\cdot R.$
2. $\forall\, Q : \overline{\mathsf{X}\cdot Q} \subseteq \mathsf{X}\cdot\overline{Q} \iff \mathsf{X}\cdot U = U \iff \mathsf{X}^\omega = U.$

Let us explain the meaning of these formulas. In relation algebra, the special case $\mathsf{X}\cdot\overline{\mathsf{skip}} \subseteq \overline{\mathsf{X}}$ of the property in Lm. 4.5.1 characterises X as a partial function and is equivalent to the full property (10) (see [27]). But in general quantales the special and the full case are not equivalent [10]. Moreover, again from [10], we know that in quantales such as that of formal languages under concatenation, left composition with an element X distributes over meet iff X is prefix-free, i.e. if no member of X is a prefix of another member. This holds in particular if all words in X have equal length, which is the case if X models a transition relation and hence consists only of words of length 2. The program P from Ex. 3.8 has analogous character.

The equivalent condition $\forall\, b : \mathsf{X}\cdot Q \cap \mathsf{X}\cdot\overline{Q} = 0$ was used in the computation calculus of R.M. Dijkstra [11].

But what about the property in Lm. 4.5.2? Only rarely will the set of all graphlets be "generated" by an element X in the sense that $\mathsf{X}^\omega = U$. The solution is to choose a left-distributive and right-strict element X and restrict the set of semantic values to the subset $\mathrm{SEM}(\mathsf{X}) =_{df} \{Q : Q \subseteq \mathsf{X}^\omega\}$, taking complements relative to X^ω. This set is clearly closed under ▯ and $\cap$ and under prefixing by X, since by isotony

$$\mathsf{X}\cdot Q \subseteq \mathsf{X}\cdot\mathsf{X}^\omega = \mathsf{X}^\omega\ .$$

Finally, it also contains all elements $S\rfloor\mathsf{X}^\omega$ for predicates S, since $S\rfloor Q \subseteq Q$ for all Q. Hence the above semantics is well-defined in $\mathrm{SEM}(\mathsf{X})$ if we replace U by X^ω. This entails

$$\llbracket\mathsf{G}\psi\rrbracket \;=\; \bigcap_{i\in\mathbb{N}} \mathsf{X}^i\cdot\llbracket\psi\rrbracket\ ,$$

in pleasant symmetry to the property $\llbracket\mathsf{F}\psi\rrbracket \;=\; \mathsf{X}^*\cdot\llbracket\psi\rrbracket = \bigcup\limits_{i\in\mathbb{N}} \mathsf{X}^i\cdot\llbracket\psi\rrbracket$.

We conclude this section by showing that one can transform the semantics of $\mathsf{G}\psi$ into closed form.

Theorem 4.6 *Assume Eq.* (10)*, i.e.,* $\forall\, Q : \overline{\mathsf{X} \cdot Q} \;=\; \mathsf{X} \cdot \overline{Q}$.

1. *Left multiplication by* X *distributes through arbitrary intersections.*
2. *If* $\cdot$ *is interpreted as* $\mathbin{;}$, *for a state formula* ψ *with* $\llbracket \psi \rrbracket \;=\; S \downharpoonleft U$ *for some predicate* S *we have*
$$\llbracket \mathsf{G}\psi \rrbracket \;=\; (S \downharpoonleft \mathsf{X})^{\omega} \ .$$

The lengthy proof is presented in the Appendix.

Example 4.7 For the program P from Ex. 3.8 we obtain, using Ex. 3.22,

$$1 \downharpoonleft P^{\omega} \;\subseteq\; \mathbb{N}_{+} \downharpoonleft P^{\omega} \;\subseteq\; (\mathbb{N}_{+} \downharpoonleft P)^{\omega} \;=\; \llbracket \mathsf{G}\mathbb{N}_{+} \rrbracket \ .$$

This means that if the variable `x` starts with value 1 then its contents will always remain positive under infinite iteration of the increment program P. □

A more elaborate example is presented in Sect. 4.6.

4.4 From CTL^* to CTL

For a number of applications the sublogic CTL of CTL^* suffices. Syntactically, it consists of those CTL^* state formulas that only use path formulas of the restricted form $\Pi \ ::= \ \mathsf{X}\,\Sigma \mid \Sigma\,\mathsf{U}\,\Sigma$.

It can be shown (in analogy to [23]) that the semantics of every CTL formula has the form $S \downharpoonleft U$ for some predicate S; moreover, S can be retrieved as $S = \mathit{in}(S \downharpoonleft U)$. This is reflected by the simplified semantics

$$\llbracket \varphi \rrbracket_s \;=_{df}\; \mathit{in}(\llbracket \varphi \rrbracket) \ ,$$

which enables us to calculate solely with predicates in the case of CTL.

First, for the Boolean connectives we obtain

$$\llbracket \varphi \vee \psi \rrbracket_s \;=\; \llbracket \varphi \rrbracket_s \cup \llbracket \psi \rrbracket_s, \qquad \llbracket \varphi \wedge \psi \rrbracket_s \;=\; \llbracket \varphi \rrbracket_s \cap \llbracket \psi \rrbracket_s, \qquad \llbracket \neg\varphi \rrbracket_s \;=\; \overline{\llbracket \varphi \rrbracket_s}.$$

To make this simplified semantics work well, we need strong properties of restriction and the *in* operator. Therefore, to use Lm. 3.18, we now choose the interpretation $\cdot = \mathbin{;}$. Then the inductive behaviour of $\llbracket \_ \rrbracket_s$ for all CTL formulas is as follows; it involves now the modal operators from Sect. 3.7.

Theorem 4.8

1. $\llbracket \bot \rrbracket_s = \emptyset$,
2. $\llbracket S \rrbracket_s = S$,
3. $\llbracket \varphi \to \psi \rrbracket_s = \overline{\llbracket \varphi \rrbracket_s} \cup \llbracket \psi \rrbracket_s$,
4. $\llbracket \mathsf{EX}\varphi \rrbracket_s = |\mathsf{X}\rangle \llbracket \varphi \rrbracket_s$,
5. $\llbracket \mathsf{AX}\varphi \rrbracket_s = |\mathsf{X}] \llbracket \varphi \rrbracket_s$,
6. $\llbracket \mathsf{E}(\varphi \mathsf{U} \psi) \rrbracket_s = |(\llbracket \varphi \rrbracket_s \mathbin{;} \mathsf{X})^*\rangle \llbracket \psi \rrbracket_s$,
7. $\llbracket \mathsf{A}(\varphi \mathsf{U} \psi) \rrbracket_s = \llbracket \mathsf{F}\psi \rrbracket_s \,\cap\, |(\overline{\llbracket \psi \rrbracket_s} \mathbin{;} \mathsf{X})^*](\llbracket \varphi \rrbracket_s + \llbracket \psi \rrbracket_s)$.

Parts 4 and 5 mean that the existential and universal quantifiers of CTL are semantically reflected as the existential and universal modal operators diamond and box. Part 6 means that the starting states of the traces in $[\![\mathsf{E}(\varphi \mathsf{U} \psi)]\!]_s$ are precisely those from which after finitely many X steps through φ states a ψ state can be reached. Part 7 characterises $[\![\mathsf{A}(\varphi \mathsf{U} \psi)]\!]_s$ as the set of those states from which eventually a ψ state must be reached and for which iteration through non-ψ states must lead to a φ or a ψ state. The proofs are again direct translations of the corresponding ones in [23].

This theorem entails the following pleasant characterisations.

Corollary 4.9

$$\begin{aligned} [\![\mathsf{EF}\psi]\!]_s &= |\mathsf{X}^*\rangle [\![\psi]\!]_s \ , & [\![\mathsf{EG}\psi]\!]_s &= \mathit{in}(([\![\psi]\!]_s \,;\, \mathsf{X})^\omega) \ , \\ [\![\mathsf{AG}\psi]\!]_s &= |\mathsf{X}^*] [\![\psi]\!]_s \ , & [\![\mathsf{AF}\psi]\!]_s &= \overline{\mathit{in}((\overline{[\![\psi]\!]_s} \,;\, \mathsf{X})^\omega)} \ . \end{aligned}$$

4.5 CTL* and LTL

The logic LTL is the fragment of CTL^* in which only A may occur, once and outermost only, as path quantifier. More precisely, the LTL path formulas are given by

$$\Pi ::= \Phi \mid \bot \mid \Pi \to \Pi \mid \mathsf{X}\,\Pi \mid \Pi \,\mathsf{U}\, \Pi.$$

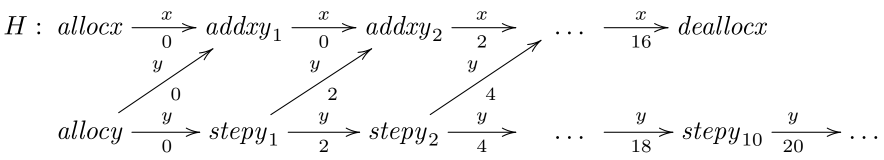
The LTL semantics is embedded into the CTL^* one by assigning to $\varphi \in \Pi$ the semantic value $[\![\mathsf{A}\varphi]\!]$.

In so far, LTL for the interface model is covered by the semantics in Sect. 4.2. If instead of the graphlet quantale the quantale of finite and infinite sequences is used, by a slight twist of the general semantics one can obtain a simplified version using only *finite* iteration and the modal operators [23]. However, that twist is possible only, since the generating element X in that algebra is left cancellative w.r.t. sequential composition. Since this in general will not hold in the interface model, we just stay with the LTL semantics given above.

4.6 A Somewhat Larger Example

We consider a program that works on two variables `x` and `y` that are allocated and initialised to 0 by the actions *allocx* and *allocy*. Then the program uses two actions *stepy* and *addxy* given by `y := y+2` and `x := x+y`, respectively. These actions are performed repeatedly. For `x` only 8 such actions take place, after which `x` is deallocated; the action on `y` is repeated forever.

We use arrows of the form $\xrightarrow[v]{z}$, where z is a variable and v a value. To avoid excessive indexing we assume that arrows are, in addition to their labelling, distinguished by their position within the diagram. A set of arrows is considered coherent, i.e., a state, if for every variable z it contains at most one arrow $\xrightarrow[v]{z}$. The overall graph H of the program looks as follows.



This example exhibits the following phenomena.

- The graph contains special subgraphs the arrows of which have the shape of the letter N rotated clockwise by 90°. Each of these is induced by events $addxy_i, addxy_{i+1}$, $stepy_i, stepy_{i+1}$ $(1 \leq i \leq 7)$ or by $allocx, addxy_1, allocy$, $stepy_1$. According to a result by Gischer [14], a graph containing such N-shaped subgraphs, hence in particular the graph H, cannot be expressed as a composition of singleton graphlets using only the ; and | operators.
- The graph can, however, be decomposed into a sequential composition of the slices $\{allocx, allocy\}$, $\{addxy_i, stepy_i\}$ $(1 \leq i \leq 9)$, $\{deallocx, stepy_{10}\}$ and $\{stepy_j\}$ $(j \geq 11)$. These slices can be taken as the elements of a small-step program X that corresponds to the CTL^* next-time operator. Then the overall graph is the “longest” element of X^ω, selected from X^ω by the restriction $\{\emptyset\} \rfloor \mathsf{X}^\omega$, i.e., by taking the only trace in X^ω that has an empty input interface.
- We can express in CTL^* and show that y remains available forever, whereas x does not. A possible cause for this may be that deallocation of y has just been forgotten, since it is not discovered that y never is actually used any more. Such an analysis can be used to prevent possible memory leaks. To formalise that we say that a state A satisfies the predicate $has(z)$ for a variable z iff for some value v there is an arrow $\xrightarrow[v]{z}$ in A. Then we can express the above properties as
$$\mathsf{F}(\overline{has(z)})\ , \qquad \mathsf{XG}(has(y))\ ,$$
which indeed hold for our program.
- We can also describe and show the joint behaviour of x and y, notably their coupling by an invariant: our program satisfies $\mathsf{G}(y > 0 \implies y = x + 2)$, with the obvious semantics for the predicate following the G operator.
- If we choose a different small-step program consisting only of the elementary $stepy$ events we can specify and analyse the y subthread by itself in an analogous fashion: for instance, we have $\mathsf{G}(even(y))$.

5 Conclusion and Outlook

We have presented an interface model for CKA and have shown how its operators can be used to formalise variants of the temporal logics CTL^* and CTL that are suitable for specifying and reasoning about temporal subthreads of CKA programs. This was done using the fact that CKA induces a quantale w.r.t. to each of its composition operators. We have, however, only exploited the quantales dealing with sequential compositions ; and **;**. The quantale induced by the concurrent composition operator | could be used in a similar manner to set up analogous "spatial" logics, again based on finite or infinite iteration. Also, links between $\mathsf{CTL}, \mathsf{CTL}^*, \mathsf{LTL}$, the algebraic temporal logics of von Karger [20] and the Duration Calculus of Zhou [32] need to be established. The details of that as well as a combination of temporal and spatial logic constructs to deal properly with truly concurrent aspects will be the subject of further research.

Acknowledgments We are grateful to Jules Desharnais, Peter Höfner, Martin E. Müller, Patrick Roocks, Stephan van Staden and the anonymous referees for thorough proofreading and valuable comments.

References

1. Back, R., von Wright, J.: Refinement Calculus - A Systematic Introduction. Graduate Texts in Computer Science, Springer (1998)
2. de Bakker, J., Meertens, L.: On the completeness of the inductive assertion method. J. Comput. Syst. Sci. 11(3), 323–357 (1975)
3. Blikle, A.: A comparative review of some program verification methods. In: Gruska, J. (ed.) Mathematical Foundations of Computer Science 1977, 6th Symposium, Tatranska Lomnica, Czechoslovakia, September 5-9, 1977, Proceedings. Lecture Notes in Computer Science, vol. 53, pp. 17–33. Springer (1977)
4. Brink, C., Rewitzky, I.: A Paradigm for Program Semantics: Power Structures and Duality. CSLI Publications (2001)
5. Conway, J.: Regular Algebra and Finite Machines. Chapman and Hall (1971)
6. Dang, H.H., Möller, B.: Concurrency and local reasoning under reverse exchange. Sci. Comput. Prog. 85, Part B, 204–223 (2013)
7. Dang, H., Glück, R., Möller, B., Roocks, P., Zelend, A.: Exploring modal worlds. J. Log. Algebr. Meth. Program. 83(2), 135–153 (2014)
8. Desharnais, J., Möller, B., Struth, G.: Modal Kleene algebra and applications - a survey. J. Relational Methods in Computer Science 1, 93–131 (2004)
9. Desharnais, J., Möller, B., Struth, G.: Kleene algebra with domain. ACM Trans. Comput. Log. 7(4), 798–833 (2006)
10. Desharnais, J., Möller, B.: Characterizing determinacy in Kleene algebras. Inf. Sci. 139(3-4), 253–273 (2001)
11. Dijkstra, R.M.: Computation calculus bridging a formalization gap. Sci. Comput. Program. pp. 3–36 (2000)
12. Emerson, E.A.: Temporal and modal logic. In: Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B), pp. 995–1072. Elsevier (1990)
13. Frias, M.F., Pombo, C.L.: Interpretability of first-order linear temporal logics in fork algebras. J. Log. Algebr. Program. 66(2), 161–184 (2006)
14. Gischer, J.L.: The equational theory of pomsets. Theoretical Computer Science 61(2–3), 199–224 (1988)
15. Hoare, T., Möller, B., Struth, G., Wehrman, I.: Concurrent Kleene algebra and its foundations. J. Log. Algebr. Program. 80(6), 266–296 (2011)
16. Hoare, T., van Staden, S.: The laws of programming unify process calculi. Sci. Comput. Program. 85, 102–114 (2014)

17. Hoare, T., van Staden, S., Möller, B., Struth, G., Zhu, H.: Developments in concurrent Kleene algebra. J. Log. Algebr. Program. (2015 (to appear))
18. Hoare, T., van Staden, S., Möller, B., Struth, G., Villard, J., Zhu, H., O'Hearn, P.W.: Developments in concurrent Kleene algebra. In: Höfner, P., Jipsen, P., Kahl, W., Müller, M.E. (eds.) Relational and Algebraic Methods in Computer Science - 14th International Conference RAMiCS 2014, Marienstatt, Germany, April 28-May 1, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8428, pp. 1–18. Springer (2014)
19. Jipsen, P.: Concurrent Kleene algebra with tests. In: Höfner, P., Jipsen, P., Kahl, W., Müller, M. (eds.) Relational and Algebraic Methods in Computer Science - 14th International Conference, RAMiCS 2014, Marienstatt, Germany, April 28-May 1, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8428, pp. 37–48. Springer (2014)
20. von Karger, B.: Temporal algebra. Mathematical Structures in Computer Science 8(3), 277–320 (1998)
21. von Karger, B., Berghammer, R.: A relational model for temporal logic. Logic Journal of the IGPL 6(2), 157–173 (1998)
22. Main, M.: A powerdomain primer — a tutorial for the Bulletin of the EATCS 33. Tech. Rep. CU-CS-375-87 (1987). Paper 360, Univ. Colorado at Boulder, Dept of Computer Science (1987), http://scholar.colorado.edu/csci_techreports/360
23. Möller, B., Höfner, P., Struth, G.: Quantales and temporal logics. In: Johnson, M., Vene, V. (eds.) Algebraic Methodology and Software Technology, 11th International Conference, AMAST 2006, Kuressaare, Estonia, July 5-8, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4019, pp. 263–277. Springer (2006)
24. O'Hearn, P.W., Reynolds, J.C., Yang, H.: Separation and information hiding. ACM Trans. Program. Lang. Syst. 31(3), 1–50 (2009)
25. Rosenthal, K.: Quantales and their applications, Pitman Research Notes in Mathematics Series, vol. 234. Longman Scientific & Technical (1990)
26. Schmidt, G.: Programme als partielle Graphen. TU Munich, FB Mathematik. Habilitation Thesis (1977)
27. Schmidt, G., Ströhlein, T.: Relations and Graphs: Discrete Mathematics for Computer Scientists. Springer (1993)
28. Solin, K., von Wright, J.: Enabledness and termination in refinement algebra. Sci. Comput. Program. 74(8), 654–668 (2009)
29. Tarlecki, A.: A language of specified programs. Sci. Comput. Program. 5(1), 59–81 (1985)
30. Wehrman, I., Hoare, C.A.R., O'Hearn, P.W.: Graphical models of separation logic. Inf. Process. Lett. 109(17), 1001–1004 (2009)
31. Winskel, G.: On powerdomains and modality. Theor. Comput. Sci. 36, 127–137 (1985)
32. Zhou, C., Hoare, C.A.R., Ravn, A.P.: A calculus of durations. Inf. Process. Lett. 40(5), 269–276 (1991)

Appendix: Deferred Proofs

Proof of Lm. 3.5.
We only show Part 1, since Part 2 is analogous. First,

$$\begin{array}{ll}
 & \mathit{in}(G \mid G') \\
= & \quad \{\![\ \text{by Def. 3.3}\ ]\!\} \\
 & (A \cup A') \cap \overline{E+E'} \times (E+E') \\
= & \quad \{\![\ \text{distributivity}\ ]\!\} \\
 & (A \cup A') \cap (\overline{E+E'} \times E + \overline{E+E'} \times E') \\
= & \quad \{\![\ \text{distributivity}\ ]\!\} \\
 & ((A \cap \overline{E+E'} \times E) + (A \cap \overline{E+E'} \times E')) \cup \\
 & ((A' \cap \overline{E+E'} \times E) + (A' \cap \overline{E+E'} \times E'))\ .
\end{array}$$

We simplify the first two summands; the other two are analogous. For summand number one we have

$$\begin{array}{ll}
 & A \cap \overline{E+E'} \times E \\
= & \quad \{\![\ \text{De Morgan}\ ]\!\} \\
 & A \cap (\overline{E} \cap \overline{E'}) \times E \\
= & \quad \{\![\ \cap/\times\text{-exchange using } E = E \cap E\ ]\!\} \\
 & A \cap \overline{E} \times E \cap \overline{E'} \times E \\
= & \quad \{\![\ \text{by Def. 3.3}\ ]\!\} \\
 & \mathit{in}(G) \cap \overline{E'} \times E\ .
\end{array}$$

For summand number two we have

$$\begin{array}{ll}
 & A \cap \overline{E+E'} \times E' \\
= & \quad \{\![\ \text{De Morgan, distributivity}\ ]\!\} \\
 & A \cap \overline{E} \times E' \cap \overline{E'} \times E' \\
\subseteq & \quad \{\![\ \text{by } E \cap E' = \emptyset \text{, hence } E' \subseteq \overline{E} \text{ and covariance of} \times\ ]\!\} \\
 & A \cap \overline{E} \times \overline{E} \cap \overline{E'} \times E' \\
= & \quad \{\![\ \text{by Def. 3.1.7}\ ]\!\} \\
 & \emptyset \cap \overline{E'} \times E' \\
= & \quad \{\![\ \text{set algebra}\ ]\!\} \\
 & \emptyset\ .
\end{array}$$

Altogether, the claim is shown. □

Proof of Th. 3.6.
First we observe that $(\mathsf{G}, |)$ is an aggregation algebra in the sense of [15] and CS and $\mathrm{CC}(G, G') =_{df}$ `TRUE` can be viewed as independence relations with CS $\subseteq$ CC. With this, the claim follows from Lms. 3.4 and 3.5 and Prop. 3.6 of [15] if we can show that the restricting predicates CS and CC are bilinear, i.e., satisfy.

$$\begin{array}{lcl}
\mathrm{CS}(G \mid G', G'') & \iff & \mathrm{CS}(G, G'') \wedge \mathrm{CS}(G', G'')\ , \\
\mathrm{CS}(G, G' \mid G'') & \iff & \mathrm{CS}(G, G') \wedge \mathrm{CS}(G, G'')
\end{array} \tag{11}$$

and the analogous property for CC, and CC is symmetric. For CC both claims are trivial. To show (11), we exploit that the definition of CS is symmetric in both arguments, so that it suffices to consider the first argument.

$$\begin{array}{cl}
 & \mathrm{CS}(G \mid G', G'') \\
\Leftrightarrow & \quad \{\!\![\text{ by Def. 3.4 }]\!\!\} \\
 & (A \cup A') \cap A'' \cap E'' \times (E + E') = \emptyset \\
\Leftrightarrow & \quad \{\!\![\text{ distributivity, set algebra }]\!\!\} \\
 & A \cap A'' \cap E'' \times E = \emptyset \ \wedge \\
 & A \cap A'' \cap E'' \times E' = \emptyset \ \wedge \\
 & A' \cap A'' \cap E'' \times E = \emptyset \ \wedge \\
 & A' \cap A'' \cap E'' \times E' = \emptyset \\
\Leftrightarrow & \quad \{\!\![\text{ by Def. 3.4, and since } E' \cap E = E \cap E'' = E' \cap E'' = \emptyset, \\
 & \quad\quad \text{hence } E'', E' \subseteq \overline{E} \text{ and } E'', E \subseteq \overline{E'}, \text{ therefore} \\
 & \quad\quad A \cap E'' \times E' = A' \cap E'' \times E = \emptyset \text{ by condition (7) of Def. 3.1 }]\!\!\} \\
 & \mathrm{CS}(G, G'') \wedge \mathsf{TRUE} \wedge \mathsf{TRUE} \wedge \mathrm{CS}(G', G'') \ .
\end{array}$$

Proof of Lm. 3.18.
By the definitions of $;$ and $\boldsymbol{;}$ their result, and its interfaces, coincides with that of parallel composition whenever it is defined.

We start by showing the first equation. By the above remark and Lm. 3.5,

$$\mathit{in}(G \boldsymbol{;} G') \ = \ (\mathit{in}(G) \cap \overline{E'} \times E) \ \cup \ (\mathit{in}(G') \cap \overline{E} \times E') \ .$$

For the second summand we calculate by $\mathit{in}(G') = \mathit{out}(G)$ and the definition of out, Eq. (6), and disjointness of E, E' with (6) and Boolean algebra:

$$\mathit{in}(G') \cap \overline{E} \times E' = A \cap \overline{E} \times E \cap \overline{E} \times E' = A \cap \overline{E} \times (E \cap E') = \emptyset \ .$$

Concerning the first summand we have by by $\mathit{in}(G') = \mathit{out}(G)$ and the definition of in, out, definition of CS in Def. 3.4 and shunting, Eq. (6), distributivity, second summand $= \emptyset$ by Eq. (6), definition of $\cap$,

$$\begin{gathered}
\mathit{in}(G) \subseteq A \cap A' \cap \overline{E} \times E \subseteq \overline{E' \times E} \cap \overline{E} \times E = (\overline{E'} \times E + \mathrm{EV} \times \overline{E}) \cap \overline{E} \times E = \\
(\overline{E'} \times E \cap \overline{E} \times E) + (\mathrm{EV} \times \overline{E} \cap \overline{E} \times E) = \overline{E'} \times E \cap \overline{E} \times E \subseteq \overline{E'} \times E \ .
\end{gathered}$$

Therefore the first summand reduces to $\mathit{in}(G)$, as required.

The equation $\mathit{in}(G) = \mathit{in}(G \downharpoonleft \mathit{in}(G'))$ is immediate, since the definition of $\downharpoonleft$ and $\mathit{in}(G') = \mathit{out}(G)$ entail $G \downharpoonleft \mathit{in}(G') = G$.

The equations for out are proved completely symmetrically.

Now we can show associativity of $\boldsymbol{;}$. Assume graphlets G, G, G''. If any of them is $\Box$ then the associativity equation is immediate from the definition of $\boldsymbol{;}$. Otherwise we only need to check the case where $\mathit{in}(G') = \mathit{out}(G)$ and $\mathit{in}(G'') = \mathit{out}(G')$ so that both $G \boldsymbol{;} G'$ and $G' \boldsymbol{;} G''$ are defined. By the just proved equations for in and out then also $\mathit{in}(G'') = \mathit{out}(G \boldsymbol{;} G')$ and $\mathit{in}(G' \boldsymbol{;} G'') = \mathit{out}(G)$, so that also $(G \boldsymbol{;} G') \boldsymbol{;} G''$ and $G \boldsymbol{;} (G' \boldsymbol{;} G'')$ are defined; by definition of $\boldsymbol{;}$ and associativity of $;$ they coincide.

Next, by the definition of the restriction operators, $G \lfloor C$ and $C \rfloor G'$ are defined iff $\mathit{out}(G) \subseteq C$ and $\mathit{in}(G') \subseteq C$. In that case $G \lfloor C = G$ and $C \rfloor G' = G'$ and therefore

$$(G \lfloor C) \mathbin{\boldsymbol{;}} G' = G \mathbin{\boldsymbol{;}} G' = G \mathbin{\boldsymbol{;}} (C \rfloor G') \ .$$

Finally, the last two claims are immediate from the definition of restriction and the first two claims. □

Proof of Lm. 3.24.
We only show the first equation, since the second is analogous. By the definition of $|||$ and Lm. 3.5 we have

$$\mathit{in}(G \mid G') \ = \ (\mathit{in}(G) \cap \overline{E'} \times E) \ \cup \ (\mathit{in}(G') \cap \overline{E} \times E') \ .$$

The first summand reduces to $\mathit{in}(G)$ if we can show $\mathit{in}(G) \subseteq \overline{E'} \times E$, equivalently, $\mathit{in}(G) \cap \overline{\overline{E'} \times E} = \emptyset$. We calculate, by the definition of in with Eq. (6), distributivity with Eq. (6), and the definition of $|||$:

$$\mathit{in}(G) \cap \overline{\overline{E'} \times E} = A \cap \overline{E} \times E \cap (\mathrm{EV} \times \overline{E} + E' \times E) = A \cap \overline{E} \times E \cap E' \times E = \emptyset \ .$$

Symmetrically, the second summand reduces to $\mathit{in}(G')$. □

Proof of Th. 4.6.

1. $$\begin{array}{ll} & \mathsf{X} \cdot \bigcap\limits_i Q_i \\ = & \quad \{\![\text{ De Morgan }]\!\} \\ & \mathsf{X} \cdot \overline{\bigcup\limits_i \overline{Q_i}} \\ = & \quad \{\![\text{ Eq. (10) }]\!\} \\ & \overline{\mathsf{X} \cdot \bigcup\limits_i \overline{Q_i}} \\ = & \quad \{\![\text{ quantale distributivity }]\!\} \\ & \overline{\bigcup\limits_i \mathsf{X} \cdot \overline{Q_i}} \\ = & \quad \{\![\text{ Eq. (10) }]\!\} \\ & \overline{\bigcup\limits_i \overline{\mathsf{X} \cdot Q_i}} \\ = & \quad \{\![\text{ De Morgan }]\!\} \\ & \bigcap\limits_i \mathsf{X} \cdot Q_i \ . \end{array}$$
2. We show by induction on i that $\bigcap\limits_{j \le i} \mathsf{X}^j \cdot [\![\psi]\!] = (S \rfloor \mathsf{X})^i \cdot [\![\psi]\!]$. The base case $i = 0$ is trivial. The induction step proceeds as follows:

$$\begin{array}{ll} & \bigcap\limits_{j \le i+1} \mathsf{X}^j \cdot [\![\psi]\!] \\ = & \quad \{\![\text{ set theory }]\!\} \\ & \bigcap\limits_{0 < j \le i+1} \mathsf{X}^j \cdot [\![\psi]\!] \cap [\![\psi]\!] \end{array}$$

$$\begin{array}{ll}
= & \quad \{\![\ \text{left distributivity of } \mathsf{X}\ ]\!\} \\
 & \mathsf{X} \cdot (\bigcap\limits_{0<j\leq i+1} \mathsf{X}^{j-1} \cdot [\![\psi]\!]) \cap [\![\psi]\!] \\
= & \quad \{\![\ \text{index transformation } k = j-1\ ]\!\} \\
 & \mathsf{X} \cdot (\bigcap\limits_{k\leq i} \mathsf{X}^{k} \cdot [\![\psi]\!]) \cap [\![\psi]\!] \\
= & \quad \{\![\ \text{induction hypothesis}\ ]\!\} \\
 & \mathsf{X} \cdot (S \downharpoonleft \mathsf{X})^{i} \cdot [\![\psi]\!] \cap [\![\psi]\!] \\
= & \quad \{\![\ \text{Lm. 3.11}\ ]\!\} \\
 & S \downharpoonleft (\mathsf{X} \cdot (S \downharpoonleft \mathsf{X})^{i} \cdot [\![\psi]\!]) \\
= & \quad \{\![\ \text{Lm. 3.18}\ ]\!\} \\
 & (S \downharpoonleft \mathsf{X}) \cdot (S \downharpoonleft \mathsf{X})^{i} \cdot [\![\psi]\!]) \\
= & \quad \{\![\ \text{definition of powers}\ ]\!\} \\
 & (S \downharpoonleft \mathsf{X})^{i+1} \cdot [\![\psi]\!]\ .
\end{array}$$

By this,

$$[\![\mathsf{G}\psi]\!] \ = \ \bigcap_{i\in\mathbb{N}} \mathsf{X}^{i} \cdot [\![\psi]\!] \ = \ \bigcap_{i\in\mathbb{N}} \bigcap_{j\leq i} \mathsf{X}^{j} \cdot [\![\psi]\!] \ = \ \bigcap_{i\in\mathbb{N}} \bigcap_{j\leq i} (S \downharpoonleft \mathsf{X})^{i} \cdot [\![\psi]\!]\ .$$

An easy induction shows $(S \downharpoonleft \mathsf{X})^{\omega} \subseteq (S \downharpoonleft \mathsf{X})^{i} \cdot [\![\psi]\!]$ for all i and hence $(S \downharpoonleft \mathsf{X})^{\omega} \subseteq [\![\mathsf{G}\psi]\!]$. For the reverse inclusion it suffices to show that $[\![\mathsf{G}\psi]\!] \ = \ (S \downharpoonleft \mathsf{X}) \cdot [\![\mathsf{G}\psi]\!]$. Indeed,

$$\begin{array}{ll}
 & (S \downharpoonleft \mathsf{X}) \cdot [\![\mathsf{G}\psi]\!] \\
= & \quad \{\![\ \text{Lm. 3.18 and Lm. 3.11}\ ]\!\} \\
 & \mathsf{X} \cdot [\![\mathsf{G}\psi]\!] \cap (S \downharpoonleft U) \\
= & \quad \{\![\ \text{above representation}\ ]\!\} \\
 & \mathsf{X} \cdot (\bigcap\limits_{i\in\mathbb{N}} \mathsf{X}^{i} \cdot [\![\psi]\!]) \cap [\![\psi]\!] \\
= & \quad \{\![\ \text{Part 1}\ ]\!\} \\
 & (\bigcap\limits_{i\in\mathbb{N}} \mathsf{X} \cdot \mathsf{X}^{i} \cdot [\![\psi]\!]) \cap [\![\psi]\!] \\
= & \quad \{\![\ \text{definition of powers}\ ]\!\} \\
 & (\bigcap\limits_{i\in\mathbb{N}} \mathsf{X}^{i+1} \cdot [\![\psi]\!]) \cap [\![\psi]\!] \\
= & \quad \{\![\ \text{combining cases}\ ]\!\} \\
 & \bigcap\limits_{j\in\mathbb{N}} \mathsf{X}^{j} \cdot [\![\psi]\!] \\
= & \quad \{\![\ \text{above representation}\ ]\!\} \\
 & [\![\mathsf{G}\psi]\!]\ .
\end{array}$$

□